Chapter 6

Regression and Least Squares

Regression refers to the problem of finding a function that best fits a set of data points. That is, given m data points $(t_1, y_1), (t_2, y_2), \ldots, (t_m, y_m)$, one is looking for a function f that best approximates $f(t_i) \approx y_i$, where "best" has to be mathematically defined and corresponds to an optimization problem. Unlike interpolation, the number of data points in *classical* regression is larger than the number of parameters in the function used to fit the data[16], and thus in general, one cannot (or does not even want to) obtain $f(t_i) = y_i$ for all $i = 1, \ldots, n$. Examples of regression problems are illustrated in Figure 6.1.

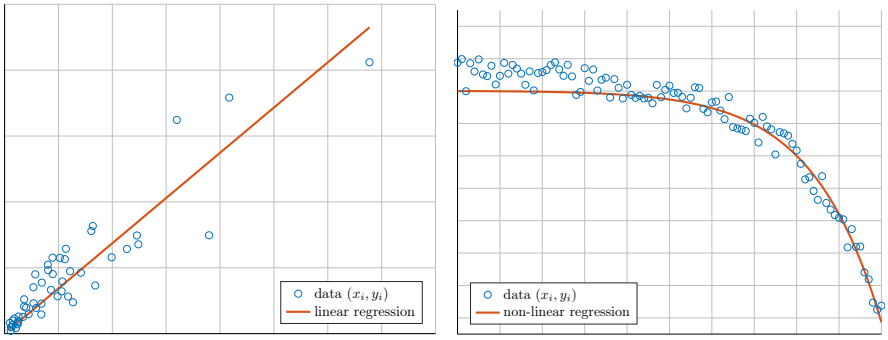
6.1 Regression: Introduction

To carry out regression, we need to specify two ingredients: The class of functions that is believed to fit the data well and the error that quantifies the fit.

6.1.1 Model function

In *parametric* regression, one considers a class of functions containing a model function that depends on n parameters. Typically, $n \ll m$ in order to avoid overfitting. Some examples are presented below; we call x_i, $i = 1, \ldots, n$, the parameters determined by fitting the function to the given data, t the problem variable, and g the model function.

[16] In regimes (statistical/machine learning) not covered in this lecture, the number of parameters is often (much) larger than the number of data points. In this case, the problem needs to be regularized, either explicitly by adding penalties/constraints or implicitly by terminating optimization methods early.



(a) Linear regression (b) Non-linear regression

Figure 6.1: Examples of regression.

- Linear Regression: g depends (affine) linearly on the *parameters*

$$\begin{aligned} g(t) &= x_1 + x_2 t \\ g(t) &= x_1 + x_2 t + x_3 t^2 \\ g(t) &= x_1 + x_2 x + \ldots + x_n t^{n-1} \\ g(t) &= x_1 e^t + x_2 e^{-t} \end{aligned} \tag{6.1}$$

- Nonlinear Regression:

$$\begin{aligned} g(t) &= x_1 + x_2 e^{x_3 t} && \text{asymptotic regression function} \\ g(t) &= \frac{x_1 t}{x_2 + t} && \text{Michaelis-Menten function} \\ g(t) &= \frac{x_1}{1 + x_2 e^{-x_3 t}} && \text{logistic function} \end{aligned} \tag{6.2}$$

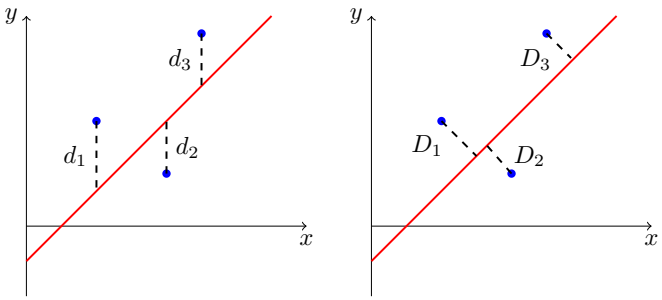
It is the application that determines *a priori* which model function to use, either by inspecting the data or by theoretical considerations. For example, in Figure 6.1a, the data looks to be approximately linear, so one would be inclined to use (6.1), while theoretical linear relationships can be given by Hooke's law, Ohm's law, or Fick's law for example. Nonlinear functions can also come from theoretical considerations; for example, the model function (6.2) is called logistic function (leading to logistic regression) because it is the solution of the logistic equation given by

$$\frac{\mathrm{d}g}{\mathrm{d}t} = rg(1-g) \quad \text{in } \mathbb{R}^+; \qquad f(0) = \alpha,$$

for some $r, \alpha > 0$.

6.1.2 Error function

The error function, also called objective or loss function, is used to assess the quality of the approximation of the data, which in turn drives the parameter selection.



(a) Vertical distance. (b) Euclidean distance.

Figure 6.2: Examples of different measures for the error.

For measuring the error at individual data points t_i, one can for example use vertical distances

$$d_i := |g(t_i) - y_i|\,, \quad i = 1, \ldots, m, \tag{6.3}$$

as in the left plot of Figure 6.2, or Euclidean distances as in the right plot of Figure 6.2. One disadvantage of the Euclidean distance is that it depends on the parameters, which complicates fitting. For example, when considering the linear function (6.1), one can show that the Euclidean distance is

$$D_i := \frac{d_i}{\sqrt{1 + x_1^2}}, \quad i = 1, \ldots, m. \tag{6.4}$$

In the following, we will therefore focus on vertical distances.

By choosing a norm on $\mathbb{R}^m$, one combines the m individual distances in a single quantity for measuring the error. The most common choices are:

1. Maximum error:

$$\max_{i=1,\ldots,m} d_i = \max_{i=1,\ldots,m} |g(t_i) - y_i| = \|\mathbf{g} - \mathbf{y}\|_\infty\,, \tag{6.5}$$

   where $\mathbf{g} := \big(g(t_1), \ldots, g(t_m)\big)$ and $\mathbf{y} := (y_1, \ldots, y_m)$.

2. Error in 1-norm:

$$\sum_{i=1}^{m} d_i = \sum_{i=1}^{m} |g(t_i) - y_i| = \|\mathbf{g} - \mathbf{y}\|_1\,. \tag{6.6}$$

3. (Squared) Error in 2-norm:

$$\sum_{i=1}^{m} d_i^2 = \sum_{i=1}^{m} |g(t_i) - y_i|^2 = \|\mathbf{g} - \mathbf{y}\|_2^2\,. \tag{6.7}$$

   When this norm is used, the resulting regression method is called least-squares. This is the most common choice because the error function has the advantage

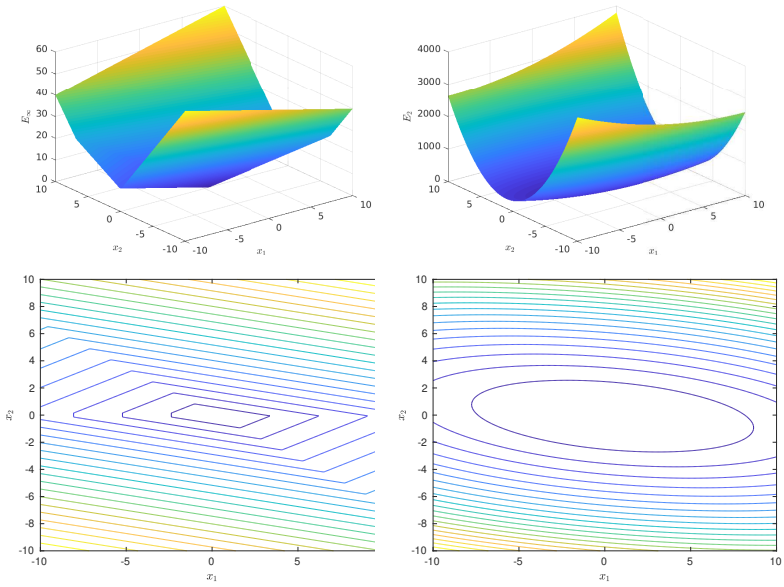


Figure 6.3: Examples of maximum (left) and 2-norm (right) error functions, respectively, both as surfaces and contour plots, for an example of linear regression.

of being differentiable if $g(t_i)$ is a differentiable function of the parameters. It also has a meaninful statistical intepretation.

Example 6.1 *Consider the linear regression defined in (6.1), and the maximum and 2-norm error functions. Figure 6.3 gives the contour plots of these error functions in an example. We can easily see the smoothness of the 2-norm while the maximum norm results in edges and corners. Note that the parameters that minimize E_2 are (usually) not equal to the ones that minimize E_∞.*

6.2 Linear least-squares

In this section, we will focus on the error function given by the 2-norm, defined in (6.7). We assume that the model function g depends linearly on the parameters to be fitted to the data, that is we can write

$$g(t) = x_1\phi_1(t) + x_2\phi_2(t) + \ldots + x_n\phi_n(t), \tag{6.8}$$

where the functions ϕ_j, $j = 1, \ldots, n$, are given functions, and x_j, $j = 1, \ldots, n$, are the parameters. For example, if $g(t) = x_1 + x_2 t$, then $\phi_1(t) = 1$ and $\phi_2(t) = t$. It will always be assumed that $n < m$, that is, there are more data points than unknown parameters.

The parameters $x_1, \ldots, x_n$ are determined by minimizing the error

$$f(\mathbf{x}) := \sum_{i=1}^{m} \left(g(t_i) - y_i\right)^2 = \|A\mathbf{x} - \mathbf{y}\|_2^2, \tag{6.9}$$

where

$$A := \begin{pmatrix} \phi_1(t_1) & \phi_2(t_1) & \cdots & \phi_n(x_1) \\ \phi_1(t_2) & \phi_2(t_2) & \cdots & \phi_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(t_m) & \phi_2(t_m) & \cdots & \phi_n(x_m) \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}.$$

In the following, we will study two different ways of finding the values of the parameters $\mathbf{x}$ that minimize (6.9): the normal equations and the QR factorization. We will assume that the matrix A has full rank n, that is, its columns are linearly independent.

6.3 Normal equations

We start by computing the gradient of f at $\mathbf{x}$ by perturbing $\mathbf{x}$:

$$\begin{aligned} f(\mathbf{x}+\mathbf{h}) - f(\mathbf{x}) &= \|A(\mathbf{x}+\mathbf{h}) - \mathbf{y}\|_2^2 - \|A\mathbf{x} - \mathbf{y}\|_2^2 \\ &= \big(A(\mathbf{x}+\mathbf{h}) - \mathbf{y}\big)^\top \big(A(\mathbf{x}+\mathbf{h}) - \mathbf{y}\big) - \big(A\mathbf{x} - \mathbf{y}\big)^\top \big(A\mathbf{x} - \mathbf{y}\big) \\ &= 2\mathbf{h}^\top A^\top A\mathbf{x} - 2\mathbf{h}^\top A^\top \mathbf{y} + O(\|\mathbf{h}\|_2^2). \end{aligned}$$

Hence,

$$\nabla f(\mathbf{x}) = 2(A^\top A\mathbf{x} - A^\top \mathbf{y})$$

and thus the gradient is zero if and only if the so called **normal equations** are satisfied:

$$A^\top A\mathbf{x} = A^\top \mathbf{y}. \tag{6.10}$$

Lemma 6.2 *If $A \in \mathbb{R}^{m\times n}$ has rank n then $A^\top A$ is symmetric positive definite.*

Proof. EFY. ∎

As a consequence of Lemma 6.2, the matrix $A^\top A$ is invertible and (6.10) has a unique solution. Since zero gradient is a necessary condition, this implies that zero gradient is also a sufficient condition for being a global minimizer of f.[17]

Theorem 6.3 *Suppose that $A \in \mathbb{R}^{m\times n}$ has rank n. Then $\mathbf{x}$ is a minimizer of $f(\mathbf{x}) = \|A\mathbf{x} - \mathbf{y}\|_2^2$ if and only if it solves the normal equations* (6.10).

The linear system (6.10) can be solved using the Cholesky factorization of $A^\mathsf{T}A$, leading to Algorithm 6.4.

[17] Another way of seeing this is to note that f is (strictly) convex.

Computation of $A^\mathsf{T}A$	$n(n+1)m$
Cholesky factorization of $A^\mathsf{T}A$	$\frac{1}{3}n^3$
Computation of $A^\mathsf{T}\mathbf{y}$	$2mn$
Forward substitution	n^2
Backward substitution	n^2

Table 6.1: Number of floating point operations for each step of Algorithm 6.4.

Algorithm 6.4 (Least-squares via normal equations)

1. $C := A^\mathsf{T}A$,
2. Compute Cholesky factor R of C using Alg. 4.17 (that is, $R^\mathsf{T}R = C$).
3. $\mathbf{b} := A^\mathsf{T}\mathbf{y}$.
4. Solve $R^\mathsf{T}\mathbf{z} = \mathbf{b}$ using forward substitution (Alg. 4.3).
5. Solve $R\mathbf{x} = \mathbf{z}$ using backward substitution (Alg. 4.4).

The computational complexity of Algorithm 6.4 is summarized in Table 6.1. The computation of $A^\mathsf{T}A$ exploits that $A^\mathsf{T}A$ is symmetric. One observes that for $m \gg n$ the computations of $A^\mathsf{T}A$ and $A^\mathsf{T}\mathbf{y}$ dominate the cost.

Remark 6.5 The normal equations (6.10) have a simple geometric interpretation. From

$$A^\mathsf{T}\left(\mathbf{b} - A\mathbf{x}\right) = 0$$

it follows that the residual $\mathbf{r} = \mathbf{y} - A\mathbf{x}$ is orthogonal to the columns of A, that is, the the residual $\mathbf{r}$ is *normal* to the span of A. It is instructive to connect this interpretation to the discussion in Section 3.3.

6.4 Method of Orthogonalization

The normal equations have a significant disadvantage; they are numerically unstable when $\kappa_2(A) \gg 1$.[18]

Example 6.6 For $m = n$ and $\phi_i(t) = t^i$ and uniformly distributed points t_i, we obtain the Vandermonde matrix

$$A = \begin{pmatrix} t_0^0 & t_0^1 & \cdots & t_0^{n-1} \\ t_1^0 & t_1^1 & \cdots & t_1^{n-1} \\ \vdots & \vdots & & \vdots \\ t_{n-1}^0 & t_{n-1}^1 & \cdots & t_{n-1}^{n-1}, \end{pmatrix}, \qquad t_i = i/(n-1).$$

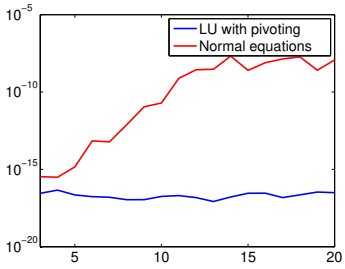
[18]The 2-norm condition number of a rectangular matrix is defined as the ration between the largest and the smallest singular value.

The linear system $A\mathbf{x} = \mathbf{y}$ with randomly chosen right-hand side $\mathbf{y}$ is solved using (a) the LU factorization with column pivoting and (b) the Cholesky factorization applied to the normal equations (6.10), For each computed solution $\widehat{\mathbf{x}}$ we measured the relative residual norm

$$\frac{\|\mathbf{r}\|_2}{\|A\|_2\|\widehat{\mathbf{x}}\|_2 + \|\mathbf{y}\|_2},$$

with

$$\mathbf{r} = \mathbf{y} - A\widehat{\mathbf{x}}.$$



It is clearly visible that the normal equations are numerically unstable; the relative residual is significantly larger than 10^{-16}. ◇

The effect observed in Example 6.6 is due to $\kappa(A^\mathsf{T}A) = \kappa(A)^2$ and hence this effect is unavoidable when working with normal equations. Note that the LU factorization has no meaningful extension to $m > n$ for least-squares problems and cannot be used as an alternative. Instead, we will use orthogonalization to reduce A to simpler form.

First, we consider the special case that the matrix $A \in \mathbb{R}^{m\times n}$ is already **upper triangular**, that is,

$$A = \begin{pmatrix} R \\ 0 \end{pmatrix} \tag{6.11}$$

for an upper triangular matrix $R \in \mathbb{R}^{n\times n}$. The vector $\mathbf{y} \in \mathbb{R}^m$ is partitioned analogously:

$$\mathbf{y} = \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix}, \qquad \mathbf{y}_1 \in \mathbb{R}^n, \quad \mathbf{y}_2 \in \mathbb{R}^{m-n}.$$

We compute

$$\begin{aligned} f(\mathbf{x}) &= \|\mathbf{y} - A\mathbf{x}\|_2^2 = \left\| \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix} - \begin{pmatrix} R \\ 0 \end{pmatrix} \mathbf{x} \right\|_2^2 \\ &= \left\| \begin{pmatrix} \mathbf{y}_1 - R\mathbf{x} \\ \mathbf{y}_2 - 0\mathbf{x} \end{pmatrix} \right\|_2^2 = \|\mathbf{y}_1 - R\mathbf{x}\|_2^2 + \|\mathbf{y}_2\|_2^2. \end{aligned}$$

If $R \in \mathbb{R}^{n\times n}$ is invertible then f is obviously minimized by the solution $\mathbf{x} \in \mathbb{R}^n$ of $R\mathbf{x} = \mathbf{y}_1$, which can be determined using backward substitution.

Using orthogonal matrices we transform $A \in \mathbb{R}^{m\times n}$ successively to the upper triangular form (6.11).

Theorem 6.7 *Consider $m \geq n$, $A \in \mathbb{R}^{m\times n}$ with rank$(A) = n$, $\mathbf{y} \in \mathbb{R}^m$. Let $Q \in \mathbb{R}^{m\times m}$ be orthogonal and $R \in \mathbb{R}^{n\times n}$ be upper triangular such that*

$$A = Q \begin{pmatrix} R \\ 0 \end{pmatrix}. \tag{6.12}$$

Partition $\widetilde{\mathbf{y}} = Q^\mathsf{T}\mathbf{y} \in \mathbb{R}^m$ as follows:

$$Q^\mathsf{T}\mathbf{y} = \begin{pmatrix} \widetilde{\mathbf{y}}_1 \\ \widetilde{\mathbf{y}}_2 \end{pmatrix}, \qquad \mathbf{y}_1 \in \mathbb{R}^n, \quad \mathbf{y}_2 \in \mathbb{R}^{m-n}.$$

Then the solution $\mathbf{x} \in \mathbb{R}^n$ of

$$R\mathbf{x} = \widetilde{\mathbf{y}}_1$$

minimizes $\|A\mathbf{x} - \mathbf{y}\|_2^2$.

Proof. Because the application of an orthogonal matrix Q^T does not change the Euclidean norm of a vector we obtain

$$\|\mathbf{y} - A\mathbf{x}\|_2 = \|Q^\mathsf{T}(\mathbf{y} - A\mathbf{x})\|_2 = \|Q^\mathsf{T}\mathbf{y} - Q^\mathsf{T}A\mathbf{x}\|_2 = \left\| \begin{pmatrix} \widetilde{\mathbf{y}}_1 \\ \widetilde{\mathbf{y}}_2 \end{pmatrix} - \begin{pmatrix} R \\ 0 \end{pmatrix} \mathbf{x} \right\|_2.$$

Hence the least-squares problem is equivalent to one in the upper triangular form discussed above. Because the rank of A is n, it follows that R is invertible, which completes the proof. □

It remains to actually compute the matrix Q from Theorem 6.7. A factorization of the form (6.12) is called **QR factorization**. In PYTHON, this factorization can be computed using `Q,R = numpy.linalg.qr(A,'complete')`. For $m \gg n$, having to compute and store the $m \times m$ matrix Q creates substantial overhead, which can be avoided. The so called **economic QR factorization** takes the form

$$A = Q_1 R,$$

where $Q_1 \in \mathbb{R}^{m\times n}$ satisfies $Q_1^\mathsf{T}Q_1 = I_n$ and $R \in \mathbb{R}^{n\times n}$ is upper triangular. This factorization completely suffices to solve the linear least-squares problem, which only requires to know R and $\widetilde{\mathbf{y}}_1 = Q_1^\mathsf{T}\mathbf{y}$. In PYTHON, this economic QR factorization is computed by `Q1,R = numpy.linalg.qr(A,'reduced')` with `numpy` package. Note that the default mode for `numpy.linalg.qr` is actually 'reduced', that is, the economic QR.

6.5 QR factorization via Gram-Schmidt

The Gram-Schmidt process is a simple way to compute the economic QR factorization.

Given $n \geq 1$ linearly independent vectors

$$\mathbf{a}_1, \ldots, \mathbf{a}_n \in \mathbb{R}^m,$$

the aim of the Gram-Schmidt process is to find n vectors

$$\mathbf{q}_1, \ldots, \mathbf{q}_n \in \mathbb{R}^m$$

such that $\|\mathbf{q}_i\|_2 = 1$ for $i = 1, \ldots, n$ and

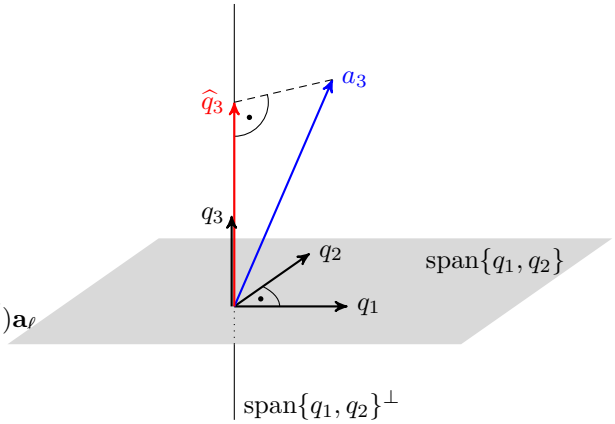
$$\operatorname{span}\{\mathbf{a}_1, \ldots, \mathbf{a}_\ell\} = \operatorname{span}\{\mathbf{q}_1, \ldots, \mathbf{q}_\ell\}, \quad \forall 1 \le \ell \le n. \tag{6.13}$$

Moreover, all vectors $\mathbf{q}_i$ are *mutually orthogonal*:

$$\mathbf{q}_i \perp \mathbf{q}_j, \quad i \neq j\,. \tag{6.14}$$

The idea of the Gram-Schmidt process is as follows. Suppose we have already computed $\ell - 1$ vectors $\mathbf{q}_1, \ldots, \mathbf{q}_{\ell-1}$ satisfying the conditions above. Then the ℓth vector $\mathbf{q}_\ell$ is obtained by projecting $\mathbf{a}_\ell$ onto the orthogonal complement of $\operatorname{span}\{\mathbf{q}_1, \ldots, \mathbf{q}_{\ell-1}\}$ and normalizing:

$$\begin{aligned} \widehat{\mathbf{q}}_\ell &= (I - (\mathbf{q}_1, \ldots, \mathbf{q}_{\ell-1})(\mathbf{q}_1, \ldots, \mathbf{q}_{\ell-1})^\mathsf{T})\mathbf{a}_\ell \\ \mathbf{q}_\ell &= \widehat{\mathbf{q}}_\ell / \|\widehat{\mathbf{q}}_\ell\|_2. \end{aligned}$$



This leads to the following algorithm.

Algorithm 6.8 (Gram-Schmidt)
Input: Linearly independent vectors $\mathbf{a}_1, \ldots, \mathbf{a}_n \in \mathbb{R}^m$.
Output: Orthonormal basis $\mathbf{q}_1, \ldots, \mathbf{q}_n$, such that (6.13) is satisfied.

for $\ell = 1, \ldots, n$ **do**
$\widehat{\mathbf{q}}_\ell := \mathbf{a}_\ell - \sum_{j=1}^{\ell-1} (\mathbf{q}_j^\mathsf{T} \mathbf{a}_\ell)\, \mathbf{q}_j$
$\mathbf{q}_\ell := \frac{\widehat{\mathbf{q}}_\ell}{\|\widehat{\mathbf{q}}_\ell\|_2}$
end for

PYTHON

```
% The columns of A containing a_1, ..., a_n are replaced by
%                             q_1, ..., q_n.
import numpy as np
for l in range(n):
    qlhat = np.zeros((n,1))
    if l!=0:
        sl = A[:,:l].T @ A[:,l]
        qlhat = A[:,l]-A[:,:l]@sl
    else:
        qlhat = A[:,l]
```

```
    rll = np.linalg.norm(qlhat)
    A[:,l] = qlhat / rll
```

The Gram-Schmidt process produces an economic QR factorization of the $m \times n$ matrix $A = (\mathbf{a}_1, \ldots, \mathbf{a}_n)$. This can be seen as follows. Let $r_{\ell\ell} := \|\widehat{\mathbf{q}}_\ell\|_2$ and $r_{j\ell} := q_j^\mathsf{T}\mathbf{a}_\ell$. Then Algorithm 6.8 implies the relation

$$\mathbf{a}_\ell = \widehat{\mathbf{q}}_\ell + \sum_{j=1}^{\ell-1} \langle \mathbf{q}_j, \mathbf{a}_\ell \rangle \, \mathbf{q}_j = \sum_{j=1}^{\ell} r_{j\ell}\mathbf{q}_j$$

for $\ell = 1, \ldots, n$. Collecting these relations into a single matrix yields

$$A = Q_1 R, \quad \text{with} \quad R = \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ 0 & r_{22} & \cdots & r_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & r_{nn} \end{pmatrix}. \tag{6.15}$$

But this happens to be an economic QR factorization, as

$$Q_1^\mathsf{T} Q_1 = \begin{pmatrix} \mathbf{q}_1^\mathsf{T}\mathbf{q}_1 & \mathbf{q}_1^\mathsf{T}\mathbf{q}_2 & \cdots & \mathbf{q}_1^\mathsf{T}\mathbf{q}_n \\ \mathbf{q}_2^\mathsf{T}\mathbf{q}_1 & \mathbf{q}_2^\mathsf{T}\mathbf{q}_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{q}_{n-1}^\mathsf{T}\mathbf{q}_n \\ \mathbf{q}_n^\mathsf{T}\mathbf{q}_1 & \cdots & \mathbf{q}_n^\mathsf{T}\mathbf{q}_{n-1} & \mathbf{q}_n^\mathsf{T}\mathbf{q}_n \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{pmatrix}$$

Remark 6.9 *Due to roundoff error, the basis produced by the Gram-Schmidt process may lose orthogonality to a large extent, especially if the columns of A are nearly linearly dependent. A simple but effective cure is to perform orthogonalization twice:*

PYTHON

```
import numpy as np
for l in range(n):
    qlhat = np.zeros((n,1))
    if l!=0:
        sl = A[:,:l].T @ A[:,l]
        A[:,l] = A[:,l] - A[:,:l] @ sl # Standard orth step.
        sl = A[:,:l].T @ A[:,l]
        qlhat = A[:,l] - A[:,:l] @ sl  # Extra orth step.
    else:
        qlhat = A[:,l]
    rll = np.linalg.norm(qlhat)
    A[:,l] = qlhat / rll
```

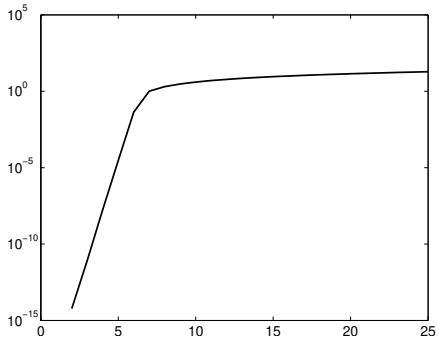
Example 6.10 Let us consider the $25 \times n$ matrix

$$A = \begin{pmatrix} 1 & t_1^1 & \cdots & t_1^n \\ 1 & t_2^1 & \cdots & t_2^n \\ \vdots & \vdots & & \vdots \\ 1 & t_{25}^1 & \cdots & t_{25}^n, \end{pmatrix}, \qquad t_i = (i-1)/24,$$

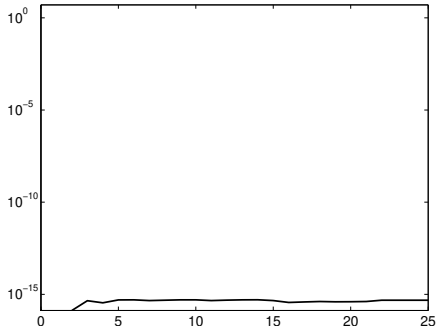
which arises when fitting a polynomial of degree n. We apply the Gram-Schmidt process (Algorithm 6.8) as well as the modification discussed in Remark 6.9. To measure the loss of orthogonality in the computed matrix $\widehat{Q}_1$, we compute

$$\|I_n - \widehat{Q}_1^{\mathsf{T}} \widehat{Q}_1\|_2.$$

Ideally, this quantity should not be much larger than 10^{-16}. The following two figures show this quantity for $n = 1, \ldots, 25$:



Gram-Schmidt
Algorithm 6.8



Improved Gram-Schmidt
Remark 6.9

It turns out that orthogonality is completely lost in the Gram-Schmidt process for $n = 7$ or larger. On the other hand, the improved Gram-Schmidt process maintains orthogonality nearly perfectly, with a loss of only $\approx 10^{-16}$. ◇

6.6 QR factorization via Householder reflectors

The MATLAB command `qr` does not use Gram-Schmidt but a different approach for computing the QR factorization, which completely avoids any issue with

In the Householder based QR factorization one constructs the matrix Q as a composition of simple orthogonal matrices. There are two types of elementary matrices, Givens rotations and Householder reflectors, and both are suitable for constructing QR factorizations. We will focus our discussion on Householder reflectors.

6.6.1 Construction

Theorem 6.11 (Properties of Householder reflectors) *Let* $\mathbf{0} \neq \mathbf{v} \in \mathbb{R}^m$. *Then the* **Householder reflector**

$$Q := I_m - \frac{2}{\mathbf{v}^{\mathsf{T}}\mathbf{v}} \mathbf{v}\mathbf{v}^{\mathsf{T}}$$

has the following properties:

1. *Q is symmetric,*
2. *Q is orthogonal,*
3. *$Q^2 = I_m$.*

Proof. EFY. □

Remark 6.12 *Householder reflectors have a geometric interpretation. When applied to a vector, they correspond to reflecting the vector at the hyperplane $\{\mathbf{x} \in \mathbb{R}^m \,|\, \mathbf{x}^\mathsf{T}\mathbf{v} = 0\}$.*

A QR factorization of $A \in \mathbb{R}^{m\times n}$ is produced by successively reducing the columns of the matrix using Householder reflections. The following result is essential for this purpose.

Lemma 6.13 *Let $\mathbf{0} \neq \mathbf{a} \in \mathbb{R}^m$ and let $\mathbf{e}_1 \in \mathbb{R}^m$ denote the first unit vector. Let*

$$\alpha = \|\mathbf{a}\|_2 \quad \text{or} \quad \alpha = -\|\mathbf{a}\|_2.$$

(If $\mathbf{a} = \beta\mathbf{e}_1$ one uses $\alpha = -\|\mathbf{a}\|_2 = -|\beta|$.) Then with $\mathbf{v} := \mathbf{a} - \alpha\mathbf{e}_1$ it holds that

$$Q = I_m - \frac{2}{\mathbf{v}^\mathsf{T}\mathbf{v}}\mathbf{v}\mathbf{v}^\mathsf{T} \qquad \Rightarrow \qquad Q\mathbf{a} = \alpha\mathbf{e}_1. \tag{6.16}$$

Proof. The choice of α implies that $\mathbf{v} \neq \mathbf{0}$ and thus Q is well defined. The claim $Q\mathbf{a} = \alpha\mathbf{e}_1$ is verified by direct calculation. □

Remark 6.14 *In practice, one chooses $\alpha = -\text{sign}(a_1)\sqrt{\mathbf{a}^\mathsf{T}\mathbf{a}}$, $\mathbf{v} = (a_1 - \alpha, a_2, \ldots, a_k)^\mathsf{T}$ to avoid numerical cancellation*[19]*.*

Now, let $\mathbf{a}_1$ denote the first column of $A \in \mathbb{R}^{m\times n}$. In the first step of the Householder based QR factorization A we use Lemma 6.13 to construct a Householder reflector $Q^{(1)}$ such that

$$Q^{(1)}\mathbf{a}_1 = \begin{pmatrix} r_{11} \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

for some $r_{11} \in \mathbb{R}$. Applied to A we obtain

$$Q^{(1)}A = \left(\begin{array}{c|ccc} r_{11} & r_{12} & \cdots & r_{1n} \\ \hline 0 & & & \\ \vdots & & A^{(1)} & \\ 0 & & & \end{array}\right), \tag{6.17}$$

[19] We define $\text{sign}(x) = 1$ for $x \geq 0$ and $\text{sign}(x) = -1$ for $x < 0$

with $A^{(1)} \in \mathbb{R}^{(m-1)\times(n-1)}$.

In the next step, we repeat this procedure for the submatrix $A^{(1)}$ and embed the transformations as follows:

$$Q^{(2)} := \left(\begin{array}{c|ccc} 1 & 0 & \cdots & 0 \\ \hline 0 & & & \\ \vdots & & \widetilde{Q}^{(2)} & \\ 0 & & & \end{array}\right),$$

and

$$\left(\begin{array}{c|ccc} 1 & 0 & \cdots & 0 \\ \hline 0 & & & \\ \vdots & & \widetilde{Q}^{(2)} & \\ 0 & & & \end{array}\right)\left(\begin{array}{c|ccc} r_{11} & r_{12} & \cdots & r_{1n} \\ \hline 0 & & & \\ \vdots & & A^{(1)} & \\ 0 & & & \end{array}\right) = \left(\begin{array}{c|ccc} r_{11} & r_{12} & \cdots & r_{1n} \\ \hline 0 & & & \\ \vdots & & \widetilde{Q}^{(2)}A^{(1)} & \\ 0 & & & \end{array}\right).$$

Letting $\mathbf{a}_2$ denote the first column of $A^{(1)}$, we choose $\widetilde{Q}^{(2)}$ as a Householder reflector that maps $\mathbf{a}_2$ to a scalar multiple of $\mathbf{e}_1 \in \mathbb{R}^{m-1}$. Then

$$Q^{(2)}Q^{(1)}A = \left(\begin{array}{cc|ccc} r_{11} & r_{12} & r_{13} & \cdots & r_{1n} \\ 0 & r_{22} & r_{23} & \cdots & r_{2n} \\ \hline 0 & 0 & & & \\ \vdots & \vdots & & A^{(2)} & \\ 0 & 0 & & & \end{array}\right),$$

It is now clear how to continue this process until the matrix $Q^{(n)}Q^{(n-1)}\cdots Q^{(1)}A$ is in upper triangular form (if $m = n$ one can skip the last step, $Q^{(n)}$).

In summary, the QR factorization of $A \in \mathbb{R}^{n\times n}$ is given by

$$A = (Q^{(1)})^{\mathsf{T}}\cdots(Q^{(n)})^{\mathsf{T}}R = \underbrace{Q^{(1)}\cdots Q^{(n)}}_{=:Q}\begin{pmatrix} R \\ 0 \end{pmatrix},$$

where $R \in \mathbb{R}^{n\times n}$ us upper triangular and the orthogonal matrices $Q^{(k)} \in \mathbb{R}^{m\times m}$ take the following form:

$$Q^{(k)} = \left(\begin{array}{c|c} I_{k-1} & 0\cdots 0 \\ \hline 0 & \\ \vdots & I_{m-k+1} - \dfrac{2}{\mathbf{v}_k^{\mathsf{T}}\mathbf{v}_k}\mathbf{v}_k\mathbf{v}_k^{\mathsf{T}} \\ 0 & \end{array}\right), \quad \mathbf{v}_k \in \mathbb{R}^{m-k+1}. \tag{6.18}$$

Algorithm 6.15 summarizes the described procedure.

Algorithm 6.15 (QR factorization)
Input: Matrix $A \in \mathbb{R}^{m\times n}$.
Output: QR factorization $A = Q\begin{pmatrix} R \\ 0 \end{pmatrix}$ with Q orthogonal and R upper triangular.

$Q = I_m$
for $k = 1, \ldots, \min\{n, m-1\}$ **do**
  Determine (embedded) Householder reflector $Q^{(k)}$ (see (6.18)) such that the trailing $m-k$ entries of the kth column of $Q^{(k)}A$ are zero.
  Replace $A \leftarrow Q^{(k)}A$.
  Replace $Q \leftarrow Q^{(k)}Q$.
end for
Return R as the first n rows of A.

Example 6.16 For

$$A = \begin{pmatrix} -4 & -2-2\sqrt{6} & -6-3\sqrt{2}-\sqrt{6} \\ 0 & -2\sqrt{3} & 9-\sqrt{3} \\ -4\sqrt{2} & -2\sqrt{2}+2\sqrt{3} & 3-6\sqrt{2}+\sqrt{3} \end{pmatrix}$$

we compute its QR factorization using Algorithm 6.15. The first column of A is

$$\mathbf{a}_1 = \begin{pmatrix} -4 \\ 0 \\ -4\sqrt{2} \end{pmatrix}, \qquad \|\mathbf{a}_1\|_2 = \sqrt{16+16\cdot 2} = \sqrt{48} = 4\sqrt{3}.$$

For the corresponding Householder reflector, we compute $\alpha = -\operatorname{sign}(a_1)\|\mathbf{a}\|_2 = -\operatorname{sign}(-4)4\sqrt{3} = 4\sqrt{3}$ and hence

$$\mathbf{v}_1 = \mathbf{a}_1 - \alpha\mathbf{e}_1 = \begin{pmatrix} -4 \\ 0 \\ -4\sqrt{2} \end{pmatrix} - 4\sqrt{3}\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} -4-4\sqrt{3} \\ 0 \\ -4\sqrt{2} \end{pmatrix}, \quad \|\mathbf{v}_1\|_2^2 = 16(6+2\sqrt{3}),$$

$$\begin{aligned} Q^{(1)} &= I_3 - \frac{2}{\|\mathbf{v}_1\|_2^2}\mathbf{v}_1\mathbf{v}_1^{\mathsf{T}} = I_3 - \frac{2}{16(6+2\sqrt{3})}\begin{pmatrix} -4-4\sqrt{3} \\ 0 \\ -4\sqrt{2} \end{pmatrix}\cdot\begin{pmatrix} -4-4\sqrt{3}, & 0, & -4\sqrt{2} \end{pmatrix} \\ &= I_3 - \frac{1}{3+\sqrt{3}}\begin{pmatrix} -1-\sqrt{3} \\ 0 \\ -\sqrt{2} \end{pmatrix}\cdot\begin{pmatrix} -1-\sqrt{3}, & 0, & -\sqrt{2} \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} - \frac{1}{\sqrt{3}(1+\sqrt{3})}\begin{pmatrix} 4+2\sqrt{3} & 0 & \sqrt{2}(1+\sqrt{3}) \\ 0 & 0 & 0 \\ \sqrt{2}(1+\sqrt{3}) & 0 & 2 \end{pmatrix} \\ &= \frac{1}{\sqrt{3}}\begin{pmatrix} -1 & 0 & -\sqrt{2} \\ 0 & \sqrt{3} & 0 \\ -\sqrt{2} & 0 & 1 \end{pmatrix}. \end{aligned}$$

Evidently, $Q^{(1)}$ is – as expected – an orthogonal matrix. We obtain

$$\begin{aligned} Q^{(1)}A &= \frac{1}{\sqrt{3}}\begin{pmatrix} -1 & 0 & -\sqrt{2} \\ 0 & \sqrt{3} & 0 \\ -\sqrt{2} & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} -4 & -2-2\sqrt{6} & -6-3\sqrt{2}-\sqrt{6} \\ 0 & -2\sqrt{3} & 9-\sqrt{3} \\ -4\sqrt{2} & -2\sqrt{2}+2\sqrt{3} & 3-6\sqrt{2}+\sqrt{3} \end{pmatrix} \\ &= \frac{1}{\sqrt{3}}\begin{pmatrix} 12 & 6 & 18 \\ 0 & -6 & -3+9\sqrt{3} \\ 0 & 6\sqrt{3} & 9+3\sqrt{3} \end{pmatrix} = \sqrt{3}\begin{pmatrix} 4 & 2 & 6 \\ 0 & -2 & -1+3\sqrt{3} \\ 0 & 2\sqrt{3} & 3+\sqrt{3} \end{pmatrix}. \end{aligned}$$

The first column $Q^{(1)}A$ equals $\alpha\mathbf{e}_1$, as desired.

We repeat this procedure for the submatrix

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

It is first column is

$$\mathbf{a}_2 = \begin{pmatrix} -2\sqrt{3} \\ 6 \end{pmatrix}, \qquad \|\mathbf{a}_2\|_2 = \sqrt{12+36} = 4\sqrt{3}.$$

In turn, $\alpha = -\text{sign}(a_1)\|\mathbf{a}_2\|_2 = -\text{sign}(-2\sqrt{3})\cdot 4\sqrt{3} = 4\sqrt{3}$ and for $\mathbf{v}_2 = \mathbf{a}_2 - \alpha\mathbf{e}_1$ we obtain

$$\mathbf{v}_2 = \begin{pmatrix} -2\sqrt{3} \\ 6 \end{pmatrix} - 4\sqrt{3}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} -6\sqrt{3} \\ 6 \end{pmatrix}, \qquad \|\mathbf{v}_2\|_2^2 = 144.$$

Therefore $\widetilde{Q}^{(2)}$ is given by

$$\begin{aligned} \widetilde{Q}^{(2)} &= I_2 - \frac{2}{\|\mathbf{v}\|_2^2}\mathbf{v}\cdot\mathbf{v}^{\mathsf{T}} = I_2 - \frac{2}{144}\cdot\begin{pmatrix} -6\sqrt{3} \\ 6 \end{pmatrix}\cdot\begin{pmatrix} -6\sqrt{3}, & 6 \end{pmatrix} \\ &= I_2 - \frac{1}{2}\cdot\begin{pmatrix} -\sqrt{3} \\ 1 \end{pmatrix}\cdot\begin{pmatrix} -\sqrt{3}, & 1 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \frac{1}{2}\begin{pmatrix} 3 & -\sqrt{3} \\ -\sqrt{3} & 1 \end{pmatrix} = \frac{1}{2}\begin{pmatrix} -1 & \sqrt{3} \\ \sqrt{3} & 1 \end{pmatrix} \end{aligned}$$

and we obtain

$$\widetilde{Q}^{(2)}A^{(1)} = \frac{1}{2}\begin{pmatrix} -1 & \sqrt{3} \\ \sqrt{3} & 1 \end{pmatrix}\cdot\sqrt{3}\begin{pmatrix} -2 & -1+3\sqrt{3} \\ 2\sqrt{3} & 3+\sqrt{3} \end{pmatrix} = \frac{\sqrt{3}}{2}\begin{pmatrix} 8 & 4 \\ 0 & 12 \end{pmatrix}.$$

This already gives the upper triangular matrix R:

$$R = \sqrt{3}\begin{pmatrix} 4 & 2 & 6 \\ 0 & 4 & 2 \\ 0 & 0 & 6 \end{pmatrix}.$$

The orthogonal matrix Q^{T} with $Q^{\mathsf{T}}A = R$ is given by

$$\begin{aligned} Q^{\mathsf{T}} &= \left(\begin{array}{c|cc} 1 & 0 & 0 \\ \hline 0 & \multicolumn{2}{c}{\multirow{2}{*}{$\widetilde{Q}^{(2)}$}} \\ 0 & & \end{array}\right) Q^{(1)} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -\frac{1}{2} & \frac{1}{2}\sqrt{3} \\ 0 & \frac{1}{2}\sqrt{3} & \frac{1}{2} \end{pmatrix}\cdot\frac{1}{\sqrt{3}}\begin{pmatrix} -1 & 0 & -\sqrt{2} \\ 0 & \sqrt{3} & 0 \\ -\sqrt{2} & 0 & 1 \end{pmatrix} \\ &= \frac{1}{2\sqrt{3}}\begin{pmatrix} -2 & 0 & -2\sqrt{2} \\ -\sqrt{6} & -\sqrt{3} & \sqrt{3} \\ -\sqrt{2} & 3 & 1 \end{pmatrix}. \end{aligned}$$

For verification we compute

$$\begin{aligned} QR &= \frac{1}{2\sqrt{3}} \begin{pmatrix} -2 & -\sqrt{6} & -\sqrt{2} \\ 0 & -\sqrt{3} & 3 \\ -2\sqrt{2} & \sqrt{3} & 1 \end{pmatrix} \cdot \sqrt{3} \begin{pmatrix} 4 & 2 & 6 \\ 0 & 4 & 2 \\ 0 & 0 & 6 \end{pmatrix} \\ &= \begin{pmatrix} -4 & -2-2\sqrt{6} & -6-3\sqrt{2}-\sqrt{6} \\ 0 & -2\sqrt{3} & 9-\sqrt{3} \\ -4\sqrt{2} & -2\sqrt{2}+2\sqrt{3} & 3-6\sqrt{2}+\sqrt{3} \end{pmatrix} = A. \end{aligned}$$

◇

It would be quite inefficient to implement Algorithm 6.15 literally. In particular, the matrix $Q^{(k)}$ is never formed but only applied implicitly via the vector $\mathbf{v}_k$. To apply $Q^{(k)}$ to matrix $\begin{pmatrix} B_1 \\ B_2 \end{pmatrix}$ with $B_1 \in \mathbb{R}^{(k-1)\times\ell}$ and $B_2 \in \mathbb{R}^{(m-k+1)\times\ell}$, one notes that

$$Q^{(k)} \begin{pmatrix} B_1 \\ B_2 \end{pmatrix} = \begin{pmatrix} B_1 \\ B_2 - \frac{2}{\mathbf{v}_k^\mathsf{T}\mathbf{v}_k} \mathbf{v}_k \mathbf{v}_k^\mathsf{T} B_2 \end{pmatrix}.$$

One first computes the vector $\mathbf{w} = \frac{2}{\mathbf{v}_k^\mathsf{T}\mathbf{v}_k} B_2^\mathsf{T}\mathbf{v}_k$ ($\approx 2(m-k+1)\ell$ flops) and then the rank-one update $B_2 - \mathbf{v}_k\mathbf{w}^\mathsf{T}$ ($\approx 2(m-k+1)\ell$ flops). Using the structure of A the operation $A \leftarrow Q^{(k)}A$ in Algorithm 6.15 only costs $4(m-k+1)(n-k+1)$ flops. Altogether the updates of A cost

$$4\sum_{k=1}^{n}(m-k+1)(n-k+1) \approx 2mn^2 - \frac{2}{3}n^3 \tag{6.19}$$

flops. The computation of Q is executed analogously.

6.6.2 Application to linear least-squares problems

In principle, Algorithm 6.15 together with Theorem 6.7 provide all the tools needed to solve least-squares problems. However, one can avoid the (expensive) computation of Q if one instead directly computes $Q^\mathsf{T}\mathbf{y}$; see Algorithm 6.17.

Algorithm 6.17 (Least-squares problems via QR factorization)
Input: Matrix $A \in \mathbb{R}^{m\times n}$ with $\operatorname{rank}(A) = n$, $\mathbf{y} \in \mathbb{R}^m$
Output: Solution $\mathbf{x} \in \mathbb{R}^n$ of least-square problem $\min \|A\mathbf{x} - \mathbf{y}\|_2^2$.

for $k = 1, \ldots, \min\{n, m-1\}$ **do**
Determine (embedded) Householder reflector $Q^{(k)}$ (see (6.18)) such that the trailing $m-k$ entries of the kth column of $Q^{(k)}A$ are zero.
Replace $A \leftarrow Q^{(k)}A$.
Replace $\mathbf{y} \leftarrow Q^{(k)}\mathbf{y}$.
end for
Partition $A = \begin{pmatrix} R \\ 0 \end{pmatrix}$ and $\mathbf{y} = \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix}$.
Compute $\mathbf{x} = R^{-1}\mathbf{y}_1$ using Algorithm 4.4.

Since all other costs are negligible, the overall cost of Algorithm 6.17 is given by the cost for updating A (see (6.19)), that is, $2mn^2 - \frac{2}{3}n^3$ flops. Compared with the approach via normal equations, see Table 6.1, the cost is at most a factor 2 larger.