Thomas Richter, Henry von Wahl y Thomas Wick

Introducción a la matemática numérica — Teoría, práctica y numerosos ejemplos

Traducción de Stefan Frei y Dandy Rueda

20 de febrero de 2026

This material is a first version of a chapter on orthogonalization methods to appear as part of the book *Introducción a la matemática numérica — Teoría, práctica y numerosos ejemplos* which is to be published in 2026 by the Springer publishing company.

The original German version of the book *Einführung in die Numerische Mathematik - Begriffe, Konzepte und zahlreiche Anwendungsbeispiele* has been published by Springer in 2024, https://doi.org/10.1007/978-3-662-69582-1 and is written by Thomas Richter (University of Magdeburg, Germany), Henry von Wahl (University of Jena, Germany) and Thomas Wick (University of Hannover, Germany). The Spanish translation is edited by Stefan Frei (University of Konstanz, Germany) and Dandy Rueda (University of Magdeburg and University of Hannover, Germany).

Contact: thomas.richter@ovgu.de

Capítulo 1
Métodos de ortogonalización y factorización QR

Hemos visto que la factorización LU de una matriz regular $A \in \mathbb{R}^{n\times n}$ en las dos matrices triangulares L y U puede obtenerse mediante matrices de Frobenius. En efecto, se tiene $U = FA$ y, por tanto, $L = F^{-1}$. Para estas matrices se cumple

$$\operatorname{cond}(U) = \operatorname{cond}(FA) \leq \operatorname{cond}(F)\ \operatorname{cond}(A) = \|F\|\,\|L\|\ \operatorname{cond}(A).$$

Con un pivoteo adecuado, todas las entradas de las matrices F y L satisfacen las cotas $|f_{ij}| \leq 1$ y $|l_{ij}| \leq 1$ (véase el Teorema **??**). Sin embargo, incluso bajo esta condición favorable, en general no es posible obtener mejores estimaciones para las normas de F y L que $\|F\|_\infty \leq n$ y $\|L\|_\infty \leq n$. En consecuencia, se obtiene una siguiente cota muy desfavorable:

$$\operatorname{cond}_\infty(U) \leq n^2\ \operatorname{cond}_\infty(A).$$

La matriz U, cuya inversa se requiere para la sustitución regresiva, puede estar mucho peor condicionada que la matriz A misma —la cual, de por sí, ya puede presentar un mal condicionamiento—.

En este capítulo analizaremos la factorización QR de una matriz A como una factorización estable de una matriz $A \in \mathbb{R}^{n\times n}$ en una matriz triangular superior R y una matriz ortogonal Q. La factorización QR es más estable que la factorización LU y puede emplearse para resolver sistemas de ecuaciones lineales. Además, extenderemos la factorización QR a matrices no cuadradas $A \in \mathbb{R}^{n\times m}$ con $m > n$, y la utilizaremos en la Sección 1.5 para resolver sistemas lineales sobredeterminados. Finalmente, la factorización QR también constituye la base de algoritmos eficientes para el cálculo de valores propios, lo cual se aborda en la Sección **??**.

Más allá de las diversas aplicaciones de la factorización QR, abordaremos la ortogonalización de vectores como un problema numérico en sí mismo. El algoritmo más sencillo es el método de Gram–Schmidt, véase el Teorema **??** en el Capítulo **??**. Asimismo, analizaremos alternativas más estables y robustas.

1.1. La factorización QR

Buscamos, para una matriz $A \in \mathbb{R}^{n\times n}$, un proceso de factorización $A = QR$ —donde R es una matriz triangular superior— que sea numéricamente estable y que realice la descomposición exclusivamente mediante matrices que estén bien condicionadas. En este contexto, las matrices ortogonales resultan especialmente adecuadas.

Definición 1.1 (Matriz ortogonal) Una matriz $Q \in \mathbb{R}^{n\times n}$ se denomina *ortogonal* si tanto sus vectores fila como sus vectores columna forman una base ortonormal de $\mathbb{R}^n$. En tal caso se verifica $Q^TQ = I$, de modo que la inversa de Q coincide con su traspuesta, $Q^{-1} = Q^T$.

Las matrices ortogonales tienen número de condición espectral $\operatorname{cond}_2(Q) = 1$. En efecto, para todo valor propio λ de una matriz ortogonal Q con vector propio ω se cumple

$$\|\omega\|_2^2 = (Q^TQ\omega, \omega)_2 = (Q\omega, Q\omega)_2 = (\lambda\omega, \lambda\omega)_2 = |\lambda|^2\|\omega\|_2^2 \quad \Rightarrow \quad |\lambda| = 1.$$

Para una factorización de la forma $A = QR$, donde Q es ortogonal, se tiene $R = Q^TA$ y

$$\operatorname{cond}_2(R) = \operatorname{cond}_2(Q^TA) \leq \operatorname{cond}_2(Q^T)\operatorname{cond}_2(A) = \operatorname{cond}_2(A).$$

Por tanto, el número de condición de la matriz triangular R es, a lo más, igual al de la matriz original A.

A continuación resumimos algunas propiedades de las matrices ortogonales.

Teorema 1.2 (Matriz ortogonal) *Sea $Q \in \mathbb{R}^{n\times n}$ una matriz ortogonal. Entonces Q es regular y se cumple*

$$Q^{-1} = Q^T, \quad Q^TQ = I, \quad \|Q\|_2 = 1, \quad \operatorname{cond}_2(Q) = 1.$$

Además, se cumple:

1. $\det(Q) = 1$ *o* $\det(Q) = -1$.
2. *Si* $Q_1, Q_2 \in \mathbb{R}^{n\times n}$ *son matrices ortogonales, entonces el producto* Q_1Q_2 *es también una matriz ortogonal.*
3. *Para cualquier matriz* $A \in \mathbb{R}^{n\times n}$ *se tiene* $\|QA\|_2 = \|A\|_2$.
4. *Para vectores arbitrarios* $x, y \in \mathbb{R}^n$ *se cumple*

$$\|Qx\|_2 = \|x\|_2, \quad (Qx, Qy)_2 = (x, y)_2.$$

Demostración. Solo probamos aquellas propiedades que son esenciales en el contexto de la estabilidad numérica. Principalmente, se trata de la invariancia de la norma al multiplicar por matrices ortogonales. En efecto,

$$\|QAx\|_2^2 = (QAx, QAx)_2 = (Q^TQAx, Ax)_2 = (Ax, Ax)_2 = \|Ax\|_2^2$$

y por tanto,

$$\|QA\|_2 = \sup_{x\neq 0} \frac{\|QAx\|_2}{\|x\|_2} = \sup_{x\neq 0} \frac{\|Ax\|_2}{\|x\|_2} = \|A\|_2.$$

□

Definición 1.3 (Factorización QR) La factorización de una matriz $A \in \mathbb{R}^{n\times n}$ en una matriz ortogonal $Q \in \mathbb{R}^{n\times n}$ y una matriz triangular superior $R \in \mathbb{R}^{n\times n}$, dada por

$$A = QR$$

se denomina *factorización QR*.

Definimos la factorización QR para matrices rectangulares en general. Una vez construida, la factorización QR puede utilizarse para resolver sistemas de ecuaciones lineales con la matriz A de forma eficiente. La matriz Q —a diferencia de la matriz L de la factorización LU— en general es densa y no triangular. No obstante, dado que su inversa se obtiene simplemente tomando la traspuesta, el sistema lineal puede resolverse de forma eficiente mediante

$$Ax = b \quad \Leftrightarrow \quad Q^TAx = Q^Tb \quad \Leftrightarrow \quad Rx = Q^Tb.$$

Teorema 1.4 (Factorización QR) *Sea* $A \in \mathbb{R}^{n\times n}$ *una matriz regular. Entonces existe una factorización* $A = QR$*, donde* $Q \in \mathbb{R}^{n\times n}$ *es una matriz ortogonal y* $R \in \mathbb{R}^{n\times n}$ *es una matriz triangular superior.*

Demostración. Sea $A = [a_1, \ldots, a_n]$ la matriz formada por los vectores columna a_i. Como A es regular, los vectores a_i son linealmente independientes. Sean ahora $q_1, \ldots, q_n \in \mathbb{R}^n$ un sistema ortonormal con las propiedades

$$(q_i, q_j)_2 = \delta_{ij} \text{ para } 1 \le i, j \le n, \text{ y } (q_i, a_j)_2 = 0 \text{ para } 1 \le j < i \le n.$$

Un sistema de este tipo puede construirse mediante el proceso de ortogonalización de Gram-Schmidt, véase Theorem **??**. La matriz $Q = [q_1, \ldots, q_n] \in \mathbb{R}^{n\times n}$ es ortogonal y, para las entradas de la matriz $R = Q^TA \in \mathbb{R}^{n\times n}$ con $R = (r_{ij})_{ij}$ para $i, j = 1, \ldots, n$, se cumple

$$r_{ij} = (q_i, a_j)_2 = 0 \text{ para } i > j. \tag{1.1}$$

Por tanto, R es una matriz triangular superior. Además, observando que $(Q^T)^{-1} = (Q^{-1})^T = (Q^T)^T = Q$, se concluye que

$$A = QR.$$

□

La factorización QR no es única. Si el sistema ortonormal de vectores $q_1, \ldots, q_n$ está dado, entonces al cambiar el signo de uno de ellos, por ejemplo $q_1, \ldots, q_{i-1}, -q_i, q_{i+1}, \ldots, q_n$, se obtiene otro sistema ortonormal de vectores. Según (1.1), esto produce coeficientes $\tilde{r}_{ij} = -r_{ij}$.

Esta observación permite introducir una normalización sencilla de la factorización QR. En cada paso, se elige el signo del vector q_i de modo que la entrada diagonal resulte positiva, es decir, $r_{ii} = (q_i, a_i) > 0$.

Teorema 1.5 (Unicidad de la factorización QR) *La factorización* QR *de una matriz regular* $A \in \mathbb{R}^{n\times n}$ *es única si la condición* $r_{ii} > 0$ *es impuesta.*

Demostración. Sean $A = Q_1R_1 = Q_2R_2$ dos factorizaciones QR de A. Entonces se cumple que

$$Q := Q_2^TQ_1 = R_2R_1^{-1} \in \mathbb{R}^{m\times m}, \quad Q^T := Q_1^TQ_2 = R_1R_2^{-1} \in \mathbb{R}^{n\times n}. \tag{1.2}$$

Las matrices Q y Q^T son ambas matrices triangulares superiores, por lo que Q debe ser una matriz diagonal. Además,

$$Q^TQ = Q_1^TQ_2Q_2^TQ_1 = R_1R_2^{-1}R_2R_1^{-1} = I,$$

lo que muestra que Q es ortogonal. Asimismo, de (1.2) se sigue que

$$QR_1 = R_2$$

por lo que para cada vector canónico e_i se cumple

$$q_{ii} r_{ii}^{(1)} = e_i^T (QR_1) e_i = e_i^T (R_2) e_i = r_{ii}^{(2)} > 0. \tag{1.3}$$

De modo que los elementos diagonales q_{ii} son positivos. Puesto que Q es diagonal y ortogonal, dichos elementos diagonales son sus valores propios y tienen módulo uno. Con estos hechos, se concluye que $q_{ii} = 1$ para todo i; por lo tanto, $Q = I$. Finalmente, $Q_1 = Q_2$ y, en consecuencia, $R_1 = R_2$. □

El paso fundamental en la construcción de una factorización QR es la ortogonalización de un sistema de n vectores independientes $a_1, \ldots, a_n \in \mathbb{R}^n$. Este problema será examinado en detalle en las secciones siguientes. En particular, desarrollaremos procedimientos de ortogonalización que son considerablemente más estables que proceso de ortogonalización de Gram–Schmidt estándar.

Observación 1.6 (Factorización QR para matrices rectangulares) Hasta ahora hemos considerado la factorización QR para matrices regulares $A \in \mathbb{R}^{n\times n}$. El algoritmo puede aplicarse directamente a matrices rectangulares $A \in \mathbb{R}^{n\times m}$ con $n > m$ y rango completo $\operatorname{rank}(A) = m$. Los vectores columna $a_1, \ldots, a_m \in \mathbb{R}^n$ siguen siendo linealmente independientes y pueden transformarse, por ejemplo, en un sistema ortonormal $q_1, \ldots, q_m \in \mathbb{R}^n$ que satisface

$$(q_i, q_j)_2 = \delta_{ij}, \quad (q_i, a_j)_2 = 0, \quad i, j = 1, \ldots, m. \tag{1.4}$$

La matriz $Q = (q_1, \ldots, q_m) \in \mathbb{R}^{n\times m}$, al no ser cuadrada, no es ortogonal; sin embargo, se cumple

$$Q^T Q = I \in \mathbb{R}^{m\times m}.$$

De manera análoga, a partir de la ortonormalidad (1.4), se deduce que $R = Q^T A \in \mathbb{R}^{m\times m}$ es una matriz triangular superior.

Si completamos los vectores $q_1, \ldots, q_m$ con $\tilde{q}_{m+1}, \ldots, \tilde{q}_n$ para obtener una base ortonormal de $\mathbb{R}^n$, y extendemos la matriz R con $n - m$ filas nulas hasta obtener $\tilde{R} \in \mathbb{R}^{n\times m}$, podemos escribir la factorización en la forma habitual:

$$\tilde{Q}\tilde{R} = A \quad \Leftrightarrow \quad \left(\begin{array}{c|c} & \\ Q & \tilde{Q} \\ & \end{array}\right)\left(\begin{array}{c} R \\ \hline 0 \end{array}\right) = \left(\begin{array}{c} \\ A \\ \\ \end{array}\right)$$

El enriquecimiento de los vectores $q_1, \ldots, q_m$ con $\tilde{q}_{m+1}, \ldots, \tilde{q}_n$ sirve únicamente para uniformar la notación. Estos $n-m$ vectores no necesitan calcularse en la práctica para formar la matriz $R = Q^T A$, ya que las filas inferiores de la matriz extendida R son, en cualquier caso, nulas. ♦

1.2. Ortogonalización de Gram–Schmidt

El método de ortogonalización de Gram–Schmidt, véase el Teorema **??**, es un procedimiento sencillo basado en proyectar un vector sobre los vectores ya ortogonales. La fórmula básica de iteración para ortonormalizar los vectores $a_1, a_2, \ldots, a_m$ es

$$q_1 := \frac{a_1}{\|a_1\|}, \qquad \tilde{q}_i = a_i - \sum_{j=1}^{i-1} (a_i, q_j)\, q_j, \qquad q_i := \frac{\tilde{q}_i}{\|\tilde{q}_i\|}, \quad i = 2, 3, \ldots, m.$$

Aquí, (x, y) denota un producto escalar y $\|x\| = (x, x)^{1/2}$ es la norma asociada. El costo computacional del proceso crece con el número de vectores de la base. En el paso i, es necesario calcular $i - 1$ productos escalares, $i - 1$ sumas de vectores y una norma. En el espacio $V = \mathbb{R}^n$, el número de operaciones en el paso n es, por tanto, de orden $O(n^2)$. En consecuencia, el costo total para ortogonalizar n vectores de $\mathbb{R}^n$ se comporta como $O(n^3)$, puesto que cada producto escalar requiere un costo $O(n)$.

Ejemplo 1.7 (Inestabilidad numérica del método de ortogonalización de Gram-Schmidt) Como ejemplo, consideramos los tres vectores

$$a_1 = \begin{pmatrix} 1 \\ \frac{1}{2} \\ \frac{1}{3} \end{pmatrix}, \quad a_2 = \begin{pmatrix} \frac{1}{2} \\ \frac{1}{3} \\ \frac{1}{4} \end{pmatrix}, \quad a_3 = \begin{pmatrix} \frac{1}{3} \\ \frac{1}{4} \\ \frac{1}{5} \end{pmatrix},$$

y realizamos el cálculo directamente en Python. Al usar `half` como representación en punto flotante, incluso esta pequeña tarea muestra una inestabilidad numérica. El método de ortogonalización de Gram-Schmidt se implementa de la siguiente manera:

Implementación 1.8: Gram-Schmidt Orthogonalization

```python
def gram_schmidt(A):
    n = A.shape[0]
    Q = np.zeros_like(A)

    for i in range(n):
        Q[:, i] = A[:, i]
        for j in range(i):
            Q[:, i] -= np.inner(A[:, i], Q[:, j]) * Q[:, j]
        Q[:, i] /= np.linalg.norm(Q[:, i])
    return Q
```

Cuando aplicamos esto a la matriz de Hilbert 3×3 en precisión `half`

```python
A = np.array([[1,     1 / 2, 1 / 3],
              [1 / 2, 1 / 3, 1 / 4],
              [1 / 3, 1 / 4, 1 / 5]], dtype=np.half)
Q = gram_schmidt(A)
print(Q)
```

obtenemos

```
[[ 0.857  -0.4995  0.274 ]
 [ 0.4285  0.569  -0.7905]
 [ 0.2856  0.653   0.548 ]]
```

Comprobamos su ortogonalidad, es decir, calculamos $Q^T Q - I$ y obtenemos:

```python
print(Q.T @ Q)
err = np.linalg.norm(Q.T @ Q - np.identity(Q.shape[0]), ord=2)
print(f'||Q * Q^T - I||_2 = {err}')
```

```
[[ 0.9995    0.002161  0.05252 ]
 [ 0.002161  0.9995   -0.2289  ]
 [ 0.05252  -0.2289    1.    ]]
||Q * Q^T - I||_2 = 0.33211513864862696
```

El valor obtenido indica un error del 33 %, lo que implica que la matriz resultante constituye una aproximación muy deficiente de la matriz identidad. ◀

Veamos el siguiente ejemplo:

Ejemplo 1.9 (Factorización QR mediante ortogonalización de Gram-Schmidt) Consideremos el sistema de ecuaciones lineales $Ax = b$ con

$$A := \begin{pmatrix} 1 & 1 & 1 \\ 0.01 & 0 & 0.01 \\ 0 & 0.01 & 0.01 \end{pmatrix}, \quad b := \begin{pmatrix} 1 \\ 0 \\ 0.02 \end{pmatrix},$$

y solución exacta

$$x = \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix}.$$

A partir de los vectores columna de $A = (a_1, a_2, a_3)$, determinamos una base ortonormal mediante el proceso de Gram–Schmidt. Con una precisión de tres cifras, se obtiene

$$q_1 = \frac{q_1}{\|q_1\|} \approx \begin{pmatrix} 1 \\ 0.01 \\ 0 \end{pmatrix}.$$

A continuación,

$$\tilde{q}_2 = a_2 - (a_2, q_1)q_1 \approx a_2 - q_1 = \begin{pmatrix} 0 \\ -0.01 \\ 0.01 \end{pmatrix}, \quad q_2 = \frac{\tilde{q}_2}{\|\tilde{q}_2\|} \approx \begin{pmatrix} 0 \\ -0.709 \\ 0.709 \end{pmatrix},$$

y finalmente

$$\tilde{q}_3 = a_3 - (a_3, q_1)q_1 - (a_3, q_2)q_2 \approx a_3 - q_1 - 0 = \begin{pmatrix} 0 \\ 0 \\ 0.01 \end{pmatrix}, \quad q_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

La factorización QR resultante da lugar a la "matriz ortogonal"

$$\tilde{Q} = \begin{pmatrix} 1 & 0 & 0 \\ 0.01 & -0.709 & 0 \\ 0 & 0.709 & 1 \end{pmatrix},$$

así como a la matriz triangular superior $\tilde{R}$ con $\tilde{r}_{ij} = (q_i, a_j)$ para $j \geq i$

$$\tilde{R} := \begin{pmatrix} (q_1, a_1) & (q_1, a_2) & (q_1, a_3) \\ 0 & (q_2, a_2) & (q_2, a_3) \\ 0 & 0 & (q_3, a_3) \end{pmatrix} \approx \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0.00709 & 0 \\ 0 & 0 & 0.01 \end{pmatrix}.$$

Para resolver el sistema mediante esta factorización, resolvemos el sistema $\tilde{R}\tilde{x} = \tilde{Q}^T b$:

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0.00709 & 0 \\ 0 & 0 & 0.01 \end{pmatrix} \begin{pmatrix} \tilde{x}_1 \\ \tilde{x}_2 \\ \tilde{x}_3 \end{pmatrix} = \tilde{b} = \tilde{Q}^T b \approx \begin{pmatrix} 1 \\ 0.0142 \\ 0.02 \end{pmatrix}$$

de donde se obtiene

$$\tilde{x} = \begin{pmatrix} -3 \\ 2 \\ 2 \end{pmatrix}$$

con un error relativo del 140 %:

$$\frac{\|\tilde{x} - x\|_2}{\|x\|_2} \approx 1.41$$

Al resolver el sistema con esta matriz perturbada, aparece un error enorme. Esto se debe a que la matriz $\tilde{Q}$ presenta una ortogonalidad fuertemente perturbada

$$I \overset{!}{=} \tilde{Q}^T \tilde{Q} \approx \begin{pmatrix} 1 & -0.00709 & 0 \\ -0.00709 & 1.01 & 0.709 \\ 0 & 0.709 & 1 \end{pmatrix}, \quad \|\tilde{Q}^T \tilde{Q} - I\|_2 \approx 0.7.$$

Adaptamos la implementación en Python del método de ortogonalización de Gram-Schmidt para crear la factorización QR:

Implementación 1.10: QR Factorization Using the Gram-Schmidt Method

```
def qr_gram_schmidt(A):
    n = A.shape[0]
    Q, R = np.zeros_like(A), np.zeros_like(A)

    for i in range(n):
        Q[:, i] = A[:, i]
        for j in range(i):
            Q[:, i] -= np.inner(A[:,i], Q[:, j]) * Q[:, j]
        Q[:, i] /= np.linalg.norm(Q[:, i])
        for j in range(i, n):
            R[i, j] = np.inner(Q[:, i], A[:, j])
    return Q, R
```

Con el código implementado para la sustitución regresiva en el Capítulo **??**, podemos resolver el sistema. Usando la representación en punto flotante `half`, obtenemos

```
A = np.array([[1,    1,    1   ],
              [0.01, 0,    0.01],
              [0,    0.01, 0.01]], dtype=np.half)
b = np.array([1, 0, 0.02], dtype=np.half)
x_ex = np.array([-1, 1, 1])

Q, R = qr_gram_schmidt(A)
b2 = np.dot(Q.transpose(), b)
x = backward(R, b2)

print('x =', x)
print('x_ex = ', x_ex)
```

```
x = [-3. 2. 2.]
x_ex =  [-1  1  1]
```

```
rel_err = np.linalg.norm(x - x_ex) / np.linalg.norm(x_ex)
print(f'||x - x_ex|| / ||x|| = {rel_err}')
```

```
||x - x_ex|| / ||x_ex|| = 1.414213562373095
```

Aquí, la ortogonalidad es similar a la factorización QR calculada manualmente:

```
err = np.linalg.norm(Q @ Q.transpose() - np.identity(Q.shape[0]), ord=2)
print(f'||Q * Q^T - I||_2 = {err}')
```

```
||Q * Q^T - I||_2 = 0.7072275245944134
```

◀

En principio, la factorización QR tiene mejores propiedades de estabilidad que la factorización LU, ya que la matriz R tiene, como máximo, el número de condición de la matriz A. Sin embargo, esta consideración solo se aplica si Q y R pueden generarse sin errores. El proceso estándar de ortogonalización de Gram-Schmidt no es adecuado para crear la matriz ortogonal Q. En las

secciones siguientes, introduciremos métodos de ortogonalización alternativos que se basan en la aplicación de operaciones bien condicionadas. No obstante, antes de ello discutimos brevemente una variante del método de Gram–Schmidt, en la cual el paso de ortogonalización

$$\tilde{q}_i := a_i - \sum_{j=1}^{i-1} (a_i, q_j) q_j,$$

se sustituye por una regla iterativa:

$$\tilde{q}_i^0 := a_i, \quad k = 1, \ldots, i-1: \quad \tilde{q}_i^{(k)} = \tilde{q}_i^{(k-1)} - (\tilde{q}_i^{(k-1)}, q_k) q_k, \quad \tilde{q}_i := \tilde{q}_i^{(i-1)}.$$

De este modo, los errores que van surgiendo se tienen siempre en cuenta durante la proyección.

Teorema 1.11 (Método de Gram-Schmidt modificado) *Sean $\{a_1, \ldots, a_n\}$ una base de un espacio vectorial V, y sea $(\cdot, \cdot)$ un producto escalar con norma inducida $\|\cdot\|$. El algoritmo*

(i) $q_1 := \dfrac{a_1}{\|a_1\|}$.

(ii) Para $i = 2, \ldots, n$:

- $\tilde{q}_i^{(0)} := a_i$,
- $\tilde{q}_i^{(k)} := \tilde{q}_i^{(k-1)} - (\tilde{q}_i^{(k-1)}, q_k)\, q_k, \quad k = 1, \ldots, i-1,$
- $q_i := \dfrac{\tilde{q}_i^{(i-1)}}{\|\tilde{q}_i^{(i-1)}\|}$.

genera una base ortonormal $\{q_1, \ldots, q_n\}$ de V. Además, se cumple que

$$(q_i, a_j) = 0 \quad \forall 1 \le j < i \le n.$$

Demostración. La demostración puede realizarse de forma similar al Teorema **??**. □

A continuación repetimos el Ejemplo 1.7 con esta modificación.

Ejemplo 1.12 (Método de Gram-Schmidt modificado) Volvemos a considerar los vectores

$$a_1 = \begin{pmatrix} 1 \\ \frac{1}{2} \\ \frac{1}{3} \end{pmatrix}, \quad a_2 = \begin{pmatrix} \frac{1}{2} \\ \frac{1}{3} \\ \frac{1}{4} \end{pmatrix}, \quad a_3 = \begin{pmatrix} \frac{1}{3} \\ \frac{1}{4} \\ \frac{1}{5} \end{pmatrix}$$

y calculamos la ortogonalización con el algoritmo modificado, que implementamos a continuación en Python:

Implementación 1.13: Método de Gram-Schmidt modificado

```python
def gram_schmidt_mod(A):
    n = A.shape[0]
    Q = np.zeros_like(A)
    for i in range(n):
        Q[:, i] = A[:, i]
        for j in range(i):
            Q[:, i] -= np.inner(Q[:, i], Q[:, j]) * Q[:, j]
        Q[:, i] /= np.linalg.norm(Q[:, i])
    return Q
```

Dado que en el paso k de la parte (ii) del procedimiento solo necesitamos acceder a $\tilde{q}_i^{(k-1)}$, podemos sobrescribir este valor directamente con $\tilde{q}_i^{(k)}$. Aplicamos nuevamente el método a la matriz de Hilbert 3×3 usando la representación en punto flotante `half`:

```python
Q = gram_schmidt_mod(A)
err = np.linalg.norm(Q @ Q.transpose() - np.identity(Q.shape[0]), ord=2)
print(f'Q =\n{Q}')
print(f'||Q * Q^T - I||_2 = {err}')
```

y obtenemos

```
Q =
[[ 0.857  -0.4995  0.1514]
 [ 0.4285  0.569  -0.661 ]
 [ 0.2856  0.653   0.733 ]]
||Q * Q^T - I||_2 = 0.06270300839082639
```

En comparación con el Ejemplo 1.7, obtenemos el (todavía significativo) error relativo del 6.3 %, que sin embargo es mucho menor que el 33 % alcanzado con el procedimiento original. ◀

A continuación repetimos la factorización repetimos ahora la factorización QR de la matriz A del Ejemplo 1.9, esta vez utilizando el método de Gram–Schmidt modificado:

Ejemplo 1.14 (Factorización QR con el método de Gram–Schmidt modificado) Volvamos a considerar el sistema de ecuaciones lineales

$$A := \begin{pmatrix} 1 & 1 & 1 \\ 0.01 & 0 & 0.01 \\ 0 & 0.01 & 0.01 \end{pmatrix}, \quad b := \begin{pmatrix} 1 \\ 0 \\ 0.02 \end{pmatrix},$$

cuya solución es

$$x = \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix}.$$

Omitimos la presentación detallada del cálculo con tres cifras decimales y pasamos directamente a la implementación en Python. Para calcular la factorización QR con el método de Gram–Schmidt modificado, solo necesitamos extender nuestro código anterior creando la matriz R en las líneas 10–11:

Implementación 1.15: QR Factorization With the Modified Gram-Schmidt Method

```
1  def qr_gram_schmidt_mod(A):
2      n = A.shape[0]
3      Q, R = np.zeros_like(A), np.zeros_like(A)
4      for i in range(n):
5          Q[:, i] = A[:, i]
6          for j in range(i):
7              Q[:, i] -= np.inner(Q[:, i], Q[:, j]) * Q[:, j]
8          Q[:, i] /= np.linalg.norm(Q[:, i])
9          for j in range(i, n):
10             R[i, j] = np.inner(Q[:, i], A[:, j])
11     return Q, R
```

Aplicado a la matriz

```
Q, R = qr_gram_schmidt_mod(A)
b3 = np.dot(Q.transpose(), b)
x2 = backward(R, b3)
print(f'x = {x}')
```

esto produce

```
x = [-2. 2. 1.]
```

De esto, obtenemos el error relativo en la solución y la ortogonalidad:

```
rel_err = np.linalg.norm(x2 - x_ex) / np.linalg.norm(x_ex)
print(f'||x - x_ex|| / ||x_ex|| = {rel_err}')

err = np.linalg.norm(Q @ Q.transpose() - np.identity(Q.shape[0]), ord=2)
print(f'||Q * Q^T - I||_2 = {err}')
```

```
||x - x_ex|| / ||x_ex|| = 0.8164965809277261
||Q * Q^T - I||_2 = 0.01000213623046875
```

Aunque la ortogonalidad ha mejorado en un factor 70, el error de la solución ni siquiera se ha reducido a la mitad. La razón del mal resultado radica en la estructura del método: en la iteración i, el algoritmo modificado ortogonaliza siempre con respecto a los vectores q_j ya calculados, con $j < i$. Esto conduce a una mejor ortogonalidad entre los q_i —es decir, $(q_i, q_j) \approx 0$— pero no a una mejor ortogonalidad con respecto a las columnas de la matriz A. En consecuencia, $R = Q^T A$ en general no tiene forma triangular. ◀

Debido a la ortogonalidad de la matriz Q, la factorización QR tiene el potencial de ser un método muy estable. Sin embargo, hasta ahora no ha sido posible llevar a cabo el proceso de factorización de manera suficientemente estable. El cálculo de las entradas de la matriz R resulta todavía problemático.

1.3. Transformaciones de Householder

El principio geométrico detrás del método de Gram–Schmidt consiste en proyectar los vectores a_i sobre la base ortogonal ya construida $q_1, \ldots, q_{i-1}$. Esta proyección está mal condicionada si a_i es casi paralelo a alguno de los q_j con $j < i$, es decir, si $a_i \approx q_j$. En tal caso, la cancelación es inminente. Podemos escribir un paso del método de Gram–Schmidt de manera compacta mediante un producto matriz–vector:

$$\tilde{q}_i = [I - G^{(i)}]\, a_i, \qquad G^{(i)} = \sum_{l=1}^{i-1} q_l q_l^T.$$

Aquí, $vv^T \in \mathbb{R}^{n\times n}$ es el *producto diádico* de dos vectores, es decir,

$$a, b \in \mathbb{R}^n \quad \Rightarrow \quad (ab^T)_{ij} = a_i b_j, \quad i, j = 1, \ldots, n.$$

La matriz $[I - G^{(i)}]$ es una proyección $\mathbb{R}^n \to \mathbb{R}^n$, porque

$$[I - G^{(i)}]^2 = I - 2\sum_{l=1}^{i-1} q_l q_l^T + \sum_{k,l=1}^{i-1} q_l \underbrace{(q_l^T q_k)}_{=\delta_{lk}} q_k^T = [I - G^{(i)}].$$

Además, se cumple:

$$[I - G^{(i)}]\, q_k = q_k - \sum_{l=1}^{i-1} q_l \underbrace{(q_l^T q_k)}_{=\delta_{lk}} = 0, \qquad k < i.$$

Por lo tanto, la matriz $I - G^{(i)}$ no es regular. Si en el paso i el vector a_i es casi paralelo a los vectores ya ortogonalizados, es decir, si

$$\tilde{a}_i \in \delta a_i + \operatorname{span}\{q_1, \ldots, q_{i-1}\},$$

entonces

$$\tilde{q}_i = [I - G^{(i)}]\, \tilde{a}_i = [I - G^{(i)}]\, \delta a_i \quad \Rightarrow \quad \frac{\|\delta q_i\|}{\|\tilde{q}_i\|} = \frac{\|\delta q_i\|}{\|[I - G^{(i)}]\, \delta a_i\|}.$$

Para $\delta a_i \to 0$, es posible una amplificación arbitrariamente grande del error.

En lo que sigue, buscamos una transformación de A hacia una matriz triangular superior R, basada en operaciones ortogonales y, por tanto, estables. Una matriz ortogonal $Q \in \mathbb{R}^{n\times n}$ con $\det(Q) = 1$ representa una rotación y, para $\det(Q) = -1$, una reflexión (o una combinación rotación–reflexión). Las transformaciones de Householder emplean una matriz de reflexión para transformar una matriz $A \in \mathbb{R}^{n\times n}$ en una matriz triangular superior.

Definición 1.16 (Transformación de Householder) Para un vector $v \in \mathbb{R}^n$ con $\|v\|_2 = 1$, el *producto diádico* está definido por vv^T, y la matriz

$$S := I - 2\, vv^T \in \mathbb{R}^{n\times n}$$

se denomina *transformación de Householder*.

Teorema 1.17 (Transformación de Householder) *Toda transformación de Householder $S = I - 2vv^T$ con $\|v\|_2 = 1$ es simétrica y ortogonal. El producto de dos transformaciones de Householder $S_1 S_2$ es nuevamente una matriz ortogonal.*

Demostración. (i) La simetría se verifica mediante

$$S^T = [I - 2vv^T]^T = I - 2(vv^T)^T = I - 2vv^T = S.$$

Además,

$$S^T S = I - 4vv^T + 4v \underbrace{v^T v}_{=1} v^T = I,$$

es decir, $S^{-1} = S^T$, y por tanto S es una matriz ortogonal.

(ii) Para dos matrices simétricas y ortogonales S_1 y S_2 se cumple

$$(S_1 S_2)^T = S_2^T S_1^T = S_2^{-1} S_1^{-1} = (S_1 S_2)^{-1},$$

por lo que el producto $S_1 S_2$ es nuevamente una matriz ortogonal. Esta propiedad requiere únicamente la ortogonalidad de S_1 y S_2; no se ha utilizado el hecho de que estas matrices sean transformaciones de Householder.

□

Concluimos el análisis básico de las transformaciones de Householder mediante una caracterización geométrica:

Observación 1.18 (Transformación de Householder como reflexión) Sea $v \in \mathbb{R}^n$ un vector con $\|v\|_2 = 1$. Sea además $x \in \mathbb{R}^n$ un vector con descomposición $x = \alpha v + w^\perp$, donde $w^\perp \in \mathbb{R}^n$ satisface $v^T w^\perp = 0$, es decir, pertenece al complemento ortogonal de v. Entonces, para la transformación de Householder $S = I - 2vv^T$ se cumple:

$$S(\alpha v + w^\perp) = [I - 2vv^T](\alpha v + w^\perp) = \alpha\big(v - 2v \underbrace{v^T v}_{=1}\big) + w^\perp - 2v \underbrace{v^T w^\perp}_{=0} = -\alpha v + w^\perp.$$

Por lo tanto, la transformación de Householder describe una reflexión en el hiperplano perpendicular a v. ♦

Usando transformaciones de Householder, se transforma paso a paso una matriz $A \in \mathbb{R}^{n\times n}$ en una matriz triangular superior:

$$A^{(0)} = A, \quad A^{(i)} = S^{(i)} A^{(i-1)}, \quad S^{(i)} = I - 2v^{(i)} (v^{(i)})^T,$$

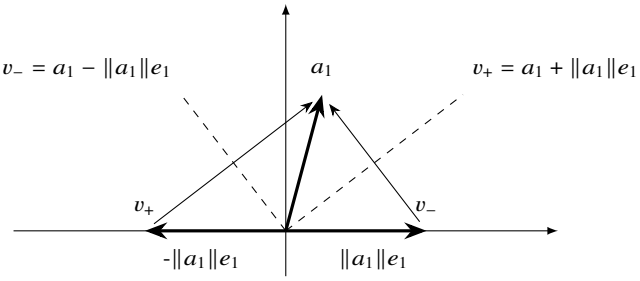


Figura 1.1 Ejes de reflexión (a trazos) y normales a la reflexión v_+ y v_- para la reflexión de a_1 sobre span$\{e_1\}$

de tal forma que $A^{(i)}$ tiene forma parcialmente triangular superior, es decir, $a_{kl}^{(i)} = 0$ para $l \le i$ y $k > l$. Finalmente, definimos $R := A^{(n-1)}$.

Describimos el primer paso del procedimiento: Dada la matriz regular $A^{(0)} := A \in \mathbb{R}^{n\times n}$, escrita en términos de sus vectores columna a_i para $i = 1, \ldots, n$, buscamos una transformación de Householder ortogonal $S^{(1)}$ tal que, al multiplicar por la izquierda, es decir, $A^{(1)} = S^{(1)}A^{(0)}$, la primera columna de $A^{(0)}$ se refleje en un múltiplo del primer vector unitario e_1. Denotamos por $a_i^{(1)}$, $i = 1, \ldots, n$, los vectores columna de $A^{(1)}$ y esperamos

$$a_1^{(1)} = S^{(1)}a_1^{(0)} \in \text{span}\{e_1\}.$$

Tenemos que reflejar con respecto al hiperplano perpendicular a $a_1 \pm \|a_1\|e_1$, véase la Figura 1.1. Para determinar el vector de reflexión disponemos de dos opciones, según la elección del signo. Por razones de estabilidad, definimos

$$v^{(1)} := \frac{a_1 + \text{sign}(a_{11})\|a_1\|e_1}{\left\|a_1 + \text{sign}(a_{11})\|a_1\|e_1\right\|}, \tag{1.5}$$

de modo que permite una elección adecuada del signo y con ello se evita una posible cancelación en la suma $a_1 \pm \|a_1\|e_1$.

Con esta elección se obtiene:

$$a_i^{(1)} = S^{(1)}a_i = a_i - 2(v^{(1)}, a_i)v^{(1)}, \quad i = 2, \ldots, n$$

$$\Rightarrow \quad a_1^{(1)} = -\,\text{sign}(a_{11})\|a_1\|e_1. \tag{1.6}$$

La matriz resultante $A^{(1)}$ es nuevamente regular y satisface $a_1^{(1)} \in \text{span}\{e_1\}$. Se tiene

$$A := \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & \ddots & & \vdots \\ a_{31} & a_{32} & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & \cdots & a_{nn} \end{pmatrix} \Rightarrow A^{(1)} := S^{(1)}A = \left(\begin{array}{c|ccc} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \cdots\ a_{1n}^{(1)} \\ \hline 0 & a_{22}^{(1)} & \ddots & \vdots \\ 0 & a_{32}^{(1)} & \ddots & \ddots\ \vdots \\ \vdots & & \ddots & \vdots \\ 0 & a_{n2}^{(1)} & \cdots & \cdots\ a_{nn}^{(1)} \end{array}\right)$$

Continuamos el procedimiento con la submatriz $\tilde{A}^{(1)} := A_{kl>1}^{(1)} \in \mathbb{R}^{(n-1)\times(n-1)}$, formada por las filas y columnas con índices mayores que 1. Para ello, consideramos el subvector de la segunda columna de $A^{(1)}$ definido por

$$\tilde{a}^{(1)} = (a_{22}^{(1)}, a_{32}^{(1)}, \ldots, a_{n2}^{(1)})^T \in \mathbb{R}^{n-1}.$$

Luego elegimos

$$\tilde{v}^{(2)} := \frac{\tilde{a}^{(1)} + \text{sign}(a_{22}^{(1)})\|\tilde{a}^{(1)}\|e_1}{\left\|\tilde{a}^{(1)} + \text{sign}(a_{22}^{(1)})\|\tilde{a}^{(1)}\|e_1\right\|}, \quad S^{(2)} = \left(\begin{array}{c|ccc} 1 & 0 & \cdots & 0 \\ \hline 0 & & & \\ 0 & & I - 2\tilde{v}^{(2)}(\tilde{v}^{(2)})^T & \\ \vdots & & & \\ 0 & & & \end{array}\right) \in \mathbb{R}^{n\times n}.$$

La multiplicación por la izquierda con $S^{(2)}$, es decir, $A^{(2)} := S^{(2)}A^{(1)}$, deja inalterada la primera fila. El bloque $A_{kl>1}^{(1)}$ se transforma de modo que sus entradas subdiagonales se anulan. La matriz resultante $A^{(2)}$ satisface $a_{kl}^{(2)} = 0$ para $l = 1, 2$ y $k > l$.

Tras $n - 1$ pasos se tiene

$$R := \underbrace{S^{(n-1)}S^{(n-2)}\cdots S^{(1)}}_{=:Q^T} A.$$

Todas las transformaciones son ortogonales; por lo tanto, $Q \in \mathbb{R}^{n\times n}$ es una matriz ortogonal —y, por supuesto, regular—, véase el Teorema 1.2.

Teorema 1.19 (Factorización QR con transformaciones de Householder) *Sea $A \in \mathbb{R}^{n\times n}$ una matriz regular. Entonces la factorización QR de A según Householder puede realizarse con*

$$\frac{2}{3}n^3 + O(n^2)$$

operaciones elementales numéricamente estables, esto es, cada una una multiplicación o división así como una suma.

Demostración. (i) La viabilidad de la factorización QR se deduce directamente del algoritmo. A partir de la regularidad de las matrices $A^{(i)}$, se sigue que el subvector $\tilde{a}^{(i)} \in \mathbb{R}^{n-i}$ no puede ser nulo. En consecuencia, $\tilde{v}^{(i)}$ queda bien definido según (1.5). La eliminación se lleva a cabo columna por columna mediante (1.6):

$$\tilde{a}_k^{(i+1)} = \underbrace{\left[I - 2\tilde{v}^{(i)}(\tilde{v}^{(i)})^T\right]}_{=\tilde{S}^{(i)}} \tilde{a}_k^{(i)} = \tilde{a}_k^{(i)} - 2\,(\tilde{v}^{(i)}, \tilde{a}_k^{(i)})\,\tilde{v}^{(i)}.$$

La estabilidad numérica está garantizada por $\mathrm{cond}_2(S^{(i)}) = 1$, véase la Observación **??**.

(ii) Procedemos ahora a estimar el costo computacional. En el paso $A^{(i)} \to A^{(i+1)}$ debe calcularse primero el vector $\tilde{v}^{(i)} \in \mathbb{R}^{n-i}$, lo que requiere $2(n-i)+2$ operaciones elementales.
A continuación, se lleva a cabo la eliminación por columnas: para cada uno de los $n-i$ vectores columna es necesario calcular un producto escalar $(\tilde{v}^{(i)}, \tilde{a}^{(i)})$ —que implica $n-i$ operaciones elementales— así como realizar una suma de vectores y la multiplicación de $(\tilde{v}^{(i)}, \tilde{a}^{(i)})$ por $\tilde{v}^{(i)}$ —otras $n-i$ operaciones elementales—. En total, se obtiene el siguiente costo computacional:

$$N_{QR} = 2\sum_{i=1}^{n-1}(n-i) + (n-i)^2 = n(n-1) + 2\,\frac{n(n-1)(2n-1)}{6} = \frac{2n^3}{3} + O(n^2).$$

□

Observación 1.20 (Factorización QR con transformaciones de Householder) Las matrices de Householder $\tilde{S}^{(i)} = I - 2\,\tilde{v}^{(i)}(\tilde{v}^{(i)})^T$ no se construyen ni almacenan de manera explícita. En una implementación eficiente, únicamente se guardan los vectores $\tilde{v}^{(i)} \in \mathbb{R}^{n-i+1}$. La matriz ortogonal

$$Q := (S^{(1)})^T \cdots (S^{(n-1)})^T$$

tampoco se forma ni almacena de forma explícita. Si se necesita aplicar Q^T, por ejemplo para calcular el vector del lado derecho $\tilde{b} = Q^T b$, esta operación se realiza paso a paso mediante las transformaciones de Householder:

$$\tilde{b} = Q^T b = S^{(n-1)} \cdots S^{(1)}\, b,$$

y el producto se evalúa iterativamente trabajando directamente con los vectores $v^{(i)}$:

Entrada: Vector b y vectores de Householder $v_1, \ldots, v_{n-1}$
$b^{(0)} = b$
para $i = 1$ **hasta** $n-1$ **hacer**
    $b^{(i+1)} = b^{(i)} - 2\,(v^{(i)}, b^{(i)})\,v^{(i)}$
Salida: $\tilde{b} = b^{(n)}$

Además de la matriz triangular superior R, deben almacenarse los vectores $\tilde{v}^{(i)} \in \mathbb{R}^{n+1-i}$. En total, esto requiere guardar $(n+1)n$ valores. A diferencia de la factorización LU, este almacenamiento no puede realizarse únicamente en el espacio de memoria de la matriz A, pues tanto R como los vectores $\tilde{v}^{(i)}$ ocupan posiciones en la diagonal. En implementaciones prácticas se debe reservar, por lo tanto, un vector diagonal adicional. ♦

Finalmente, calculamos la factorización QR para el Ejemplo 1.9 usando transformaciones de Householder:

Ejemplo 1.21 (Factorización QR con transformaciones de Householder) Consideremos nuevamente el sistema de ecuaciones lineales $Ax = b$ con

$$A := \begin{pmatrix} 1 & 1 & 1 \\ 0.01 & 0 & 0.01 \\ 0 & 0.01 & 0.01 \end{pmatrix}, \qquad b := \begin{pmatrix} 1 \\ 0 \\ 0.02 \end{pmatrix}, \qquad x = \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix}.$$

Omitimos el cálculo manual con tres cifras significativas y pasamos directamente a la implementación en Python para construir los vectores de Householder:

Implementación 1.22: Factorización QR con transformaciones de Householder

```
def qr_householder(A):
    n, m = A.shape
    V = np.zeros_like(A)
```

```
    for i in range(m):
        V[i:, i] = A[i:, i]
        ei = np.zeros(n - i, dtype=A.dtype)
        ei[0] = 1.0
        V[i:, i] += np.sign(A[i, i]) * np.linalg.norm(V[i:, i]) * ei
        V[i:, i] /= np.linalg.norm(V[i:, i])
        for k in range(i, m):
            A[i:, k] -= 2 * np.inner(V[i:, i], A[i:, k]) * V[i:, i]
    return V
```

Para utilizar estos vectores, necesitamos además implementar la aplicación de la matriz transpuesta Q^T al vector del lado derecho:

Implementación 1.23: Aplicación de las transformaciones de Householder

```
def QT_apply(V, b):
    for i in range(b.shape[0]):
        b[i:] -= 2 * np.inner(V[i], b[i:]) * V[i]
    return None
```

Aplicamos ahora la función a la matriz A usando aritmética de punto flotante de tipo `half`:

```
V = qr_householder(A)
```

Si calculamos la matriz Q^T obtenida a partir de estos vectores de Householder, obtenemos:

$$Q^T = \begin{pmatrix} -1.0 & -0.01 & 0.0 \\ 0.007053 & -0.705 & 0.7065 \\ 0.00707 & -0.7065 & -0.707 \end{pmatrix},$$

y

$$Q^T Q \approx \begin{pmatrix} 1.0 & -1.073 \times 10^{-6} & -1.669 \times 10^{-6} \\ -1.073 \times 10^{-6} & 0.9966 & -1.330 \times 10^{-3} \\ -1.669 \times 10^{-6} & -1.33 \times 10^{-3} & 0.9990 \end{pmatrix}.$$

Esto corresponde a un error relativo $\|Q^T Q - I\| \leq 4 \times 10^{-3}$.

Si resolvemos ahora el sistema utilizando los vectores ortogonales, obtenemos:

```
QT_apply(V, b)
x = backward(A, b)
print(f'x = {x}')
```

```
[-1.     0.999  1.001]
```

El error relativo de la solución es

```
rel_err = np.linalg.norm(x - x_ex) / np.linalg.norm(x_ex)
print(f'||x - x_ex|| / ||x_ex|| = {rel_err:.4e}')
```

```
||x - x_ex|| / ||x_ex|| = 7.974e-04
```

Comparamos estos resultados con los obtenidos anteriormente en los ejemplos 1.9 y 1.14. En comparación con la factorización QR mediante el método de Gram–Schmidt modificado, ahora obtenemos un error 17 veces menor utilizando precisión `half`. Además, la solución se determina con un error máximo por componente del orden de 10^{-3}. Implementada correctamente, la factorización QR con transformaciones de Householder es un procedimiento de alta estabilidad numérica. ◀

1.4. Rotaciones de Givens

El proceso de ortogonalización de Gram–Schmidt se basa en proyecciones, mientras que las transformaciones de Householder son reflexiones. Una alternativa consiste en emplear rotaciones como transformaciones ortogonales para realizar la eliminación.

Definición 1.24 (Rotación de Givens) Una *rotación de Givens* en $\mathbb{R}^n$ es una rotación en el plano generado por los vectores unitarios e_i y e_j mediante un ángulo θ. Su matriz de transformación viene dada por:

$$G(i,j,\theta) = \begin{pmatrix} 1 & & & & & & & & \\ & \ddots & & & & & & & \\ & & 1 & & & & & & \\ & & & c & & & -s & & \\ & & & & 1 & & & & \\ & & & & & \ddots & & & \\ & & & & & & 1 & & \\ & & & s & & & c & & \\ & & & & & & & 1 & \\ & & & & & & & & \ddots \\ & & & & & & & & & 1 \end{pmatrix}, \quad \begin{aligned} c &= \cos(\theta), \\ s &= \sin(\theta). \end{aligned}$$

Teorema 1.25 (Rotación de Givens) *La rotación de Givens $G(i,j,\theta)$ es una matriz ortogonal y satisface* $\det(G) = 1$. *Además, se cumple*

$$G(i,j,\theta)^{-1} = G(i,j,-\theta).$$

Demostración. La afirmación se comprueba mediante un cálculo directo. □

Al igual que en las transformaciones de Householder, las rotaciones de Givens son matrices ortogonales. La multiplicación por la izquierda, ya sea GA o Gx, sólo afecta a las filas i y j de A o a las componentes i y j de x:

$$G(i,j,\theta)A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{i-1,1} & \cdots & a_{i-1,n} \\ ca_{i1} - sa_{j1} & \cdots & ca_{in} - sa_{jn} \\ a_{i+1,1} & \cdots & a_{i+1,n} \\ \vdots & \ddots & \vdots \\ a_{j-1,1} & \cdots & a_{j-1,n} \\ sa_{i1} + ca_{j1} & \cdots & sa_{in} + ca_{jn} \\ a_{j+1,1} & \cdots & a_{j+1,n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}.$$

La factorización QR basada en rotaciones de Givens transforma la matriz A de manera sucesiva en una matriz triangular superior R. Sin embargo, la aplicación de una rotación de Givens solo permite eliminar un único elemento subdiagonal y no una columna completa. Por ello, para la primera columna se requieren $n-1$ rotaciones, para la segunda $n-2$, y así sucesivamente. Aunque esto pueda parecer costoso, cada rotación involucra únicamente cuatro entradas no triviales, y su aplicación es mucho menos costosa que un paso de Householder.

$$\begin{pmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix} \to \begin{pmatrix} * & * & * & * \\ 0 & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix} \to \begin{pmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \\ * & * & * & * \end{pmatrix} \to \begin{pmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{pmatrix}$$

$$\to \begin{pmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \\ 0 & * & * & * \end{pmatrix} \to \begin{pmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \\ 0 & 0 & * & * \end{pmatrix} \to \begin{pmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \\ 0 & 0 & 0 & * \end{pmatrix}$$

Examinemos un paso del procedimiento. Para ello, sea la matriz A dada en la forma siguiente:

$$A = \begin{pmatrix} a_{11} & \cdots & \cdots & \cdots & \cdots & a_{1n} \\ 0 & \ddots & & \ddots & & \vdots \\ \vdots & \ddots & a_{ii} & & & \vdots \\ \vdots & & 0 & a_{i+1,i+1} & & \vdots \\ \vdots & & \vdots & \vdots & & \vdots \\ \vdots & & 0 & \vdots & & \vdots \\ \vdots & & a_{ji} & \vdots & \ddots & \vdots \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \cdots & a_{ni} & a_{n,i+1} & \cdots & a_{nn} \end{pmatrix}$$

Queremos hallar la rotación $G(i, j, \theta)$ que anule el elemento a_{ji}. Para GA se tiene:

$$(G(i, j, \theta)A)_{ji} = sa_{ii} + ca_{ji}, \qquad c = \cos\theta,\ s = \sin\theta.$$

En lugar de determinar explícitamente el ángulo θ, calculamos directamente los coeficientes c y s resolviendo el sistema

$$s\,a_{ii} + c\,a_{ji} = 0, \qquad c^2 + s^2 = 1.$$

De esta forma se obtiene

$$c = \frac{a_{ii}}{\sqrt{a_{ii}^2 + a_{ji}^2}}, \qquad s = -\frac{a_{ji}}{\sqrt{a_{ii}^2 + a_{ji}^2}}.$$

Al aplicar $G(i, j, \theta)$ a la matriz A se obtiene

$$(G(i, j, \theta)A)_{ji} = \frac{-a_{ji}a_{ii} + a_{ii}a_{ji}}{\sqrt{a_{ii}^2 + a_{ji}^2}} = 0,$$

$$(G(i, j, \theta)A)_{ii} = \frac{a_{ii}^2 + a_{ji}^2}{\sqrt{a_{ii}^2 + a_{ji}^2}} = \sqrt{a_{ii}^2 + a_{ji}^2}.$$

De este modo, el elemento a_{ji} queda en efecto anulado. Para eliminar todos los elementos de la columna i por debajo de la diagonal se requieren $(n - i)$ rotaciones. Cada una de ellas modifica $(n - i)$ entradas en dos filas, lo que conduce a $4(n - i)^2$ operaciones elementales por columna. Si ignoramos las $O(n - i)$ operaciones necesarias para calcular c y s, obtenemos:

$$\sum_{i=1}^{n-1} 4(n - i)^2 = 4\sum_{i=1}^{n-1} i^2 = \frac{2}{3}(n - 1)n(2n - 1) = \frac{4}{3}n^3 + O(n^2).$$

Aunque este costo sea aproximadamente el doble del requerido por el método de Householder, las rotaciones de Givens resultan muy ventajosas cuando la matriz A es dispersa: solo es necesario anular los elementos subdiagonales no nulos. Para las matrices de *Hessenberg* —triangulares superiores con una única subdiagonal— la factorización QR mediante rotaciones de Givens puede realizarse en $O(n^2)$ operaciones. Este tipo de factorización desempeña un papel central en algoritmos para el cálculo de autovalores, véase el Capítulo **??**.

En Python, la factorización QR mediante rotaciones de Givens puede implementarse de la siguiente manera. En esta versión permitimos asimismo el caso en que la matriz tenga más filas que columnas.

Implementación 1.26: Factorización QR mediante rotaciones de Givens

```
def qr_givens(A):
    n, m = A.shape
    QT = np.identity(n, dtype=A.dtype)

    for i in range(m):
        for j in range(i + 1, n):
            c, s = A[i, i], -A[j, i]
            nrm = np.sqrt(c**2 + s**2)
            c, s = c / nrm, s / nrm
            for k in range(i, m):
                t1, t2 = A[i, k], A[j, k]
                A[i, k] = c * t1 - s * t2
                A[j, k] = s * t1 + c * t2
            for k in range(n):
                t1, t2 = QT[i, k], QT[j, k]
                QT[i, k] = c * t1 - s * t2
                QT[j, k] = s * t1 + c * t2
    return QT
```

Ejemplo 1.27 (Factorización QR mediante rotaciones de Givens) Al igual que en los Ejemplos 1.9 y 1.21, consideramos nuevamente la siguiente matriz y el siguiente vector del lado derecho:

$$A := \begin{pmatrix} 1 & 1 & 1 \\ 0.01 & 0 & 0.01 \\ 0 & 0.01 & 0.01 \end{pmatrix}, \qquad b := \begin{pmatrix} 1 \\ 0 \\ 0.02 \end{pmatrix}.$$

Si aplicamos nuestra implementación en Python con precisión `half`,

```
QT = qr_givens(A)
QTb = np.dot(QT, b)
x = backward(A, QTb)
```

obtenemos la solución con un error relativo

```
||x - x_ex|| / ||x|| = 2.8191e-04
```

Si examinamos ahora con más detalle la matriz Q^T, obtenemos

$$Q^T = \begin{pmatrix} 1.0 & 0.01 & 0.0 \\ 0.007072 & -0.707 & 0.707 \\ 0.007072 & -0.707 & -0.707 \end{pmatrix}.$$

Este resultado es prácticamente idéntico al de la matriz del Ejemplo 1.21. Las propiedades de aproximación son, por tanto, excelentes, y observamos que se cumple $QR \approx A$ y $QQ^T \approx I$:

```
err = np.linalg.norm(A2 - QT.transpose() @ A, ord=2) / np.linalg.norm(A2, ord=2)
print(f'||A - QR||_2 / ||A||_2 = {err:.4e}')
```

```
||A - QR||_2 / ||A||_2 = 7.2264e-07
```

```
Id = np.identity(QT.shape[0], dtype=np.single)
err = np.linalg.norm(Id - QT @ QT.T, ord=2)
print(f'||I - Q^T*Q||_2 = {err:.4e}')
```

```
||I - Q*Q^T||_2 = 5.0022e-05
```

La matriz `A2` es una copia de `A` en representación de punto flotante `single`, de modo que podamos calcular la norma 2 del error. Vemos así que hemos determinado la factorización QR hasta la precisión de máquina de `half`, que es aproximadamente $\varepsilon \approx 4 \times 10^{-4}$, y que la matriz Q es ortogonal con la precisión de máquina utilizada. ◀

1.5. Sistemas sobredeterminados

En un gran número de aplicaciones surgen sistemas de ecuaciones lineales en los que el número de ecuaciones no coincide con el número de incógnitas:

$$Ax = b, \quad A \in \mathbb{R}^{n\times m}, \quad x \in \mathbb{R}^m, \quad b \in \mathbb{R}^n, \quad n \neq m.$$

En el caso $n > m$ —es decir, más ecuaciones que incógnitas— nos referimos a sistemas *sobredeterminados*. En cambio, si $n < m$ hablamos de sistemas *subdeterminados*. Un método eficiente para determinar soluciones de sistemas sobredeterminados se basa en la factorización QR.

El estudio de la unicidad de la solución de sistemas generales con una matriz A rectangular ya no puede abordarse mediante su regularidad. En su lugar, sabemos que un sistema tiene solución si y sólo si el rango de la matriz A coincide con el rango de la matriz ampliada $[A|b]$. Si, además, se cumple $\operatorname{rank}(A) = m$, entonces la solución es única. Por tanto, un sistema lineal subdeterminado con $n < m$ nunca puede tener solución única.

En esta sección nos centraremos exclusivamente en sistemas sobredeterminados, es decir, en el caso $n > m$. En la práctica, este tipo de sistemas aparece con mucha frecuencia en problemas de mínimos cuadrados y en probemas de optimización. En general, los sistemas sobredeterminados no admites solución.

Ejemplo 1.28 (Sistemas sobredeterminados) Consideremos el problema de determinar el polinomio de interpolación cuadrático

$$p(x) = a_0 + a_1 x + a_2 x^2,$$

que satisfaga las condiciones

$$p\left(-\frac{1}{4}\right) = 0, \quad p\left(\frac{1}{2}\right) = 1, \quad p(2) = 0, \quad p\left(\frac{5}{2}\right) = 1.$$

Resulta evidente que las condiciones anteriores dan lugar al siguiente sistema formado por cuatro ecuaciones y sólo tres incógnitas:

$$
\begin{aligned}
a_0 - \tfrac{1}{4}a_1 + \tfrac{1}{16}a_2 &= 0\\
a_0 + \tfrac{1}{2}a_1 + \tfrac{1}{4}a_2 &= 1\\
a_0 + 2a_1 + 4a_2 &= 0\\
a_0 + \tfrac{5}{2}a_1 + \tfrac{25}{4}a_2 &= 1
\end{aligned}
$$

Al intentar resolver el sistema mediante eliminación de Gauss, obtenemos

$$
\left(\begin{array}{ccc|c} 1 & -\frac{1}{4} & \frac{1}{16} & 0\\ 1 & \frac{1}{2} & \frac{1}{4} & 1\\ 1 & 2 & 4 & 0\\ 1 & \frac{5}{2} & \frac{25}{4} & 1 \end{array}\right) \rightarrow \left(\begin{array}{ccc|c} 1 & -\frac{1}{4} & \frac{1}{16} & 0\\ 0 & 3 & \frac{3}{4} & 4\\ 0 & 9 & \frac{63}{4} & 0\\ 0 & 11 & \frac{99}{4} & 4 \end{array}\right) \rightarrow \left(\begin{array}{ccc|c} 1 & -\frac{1}{4} & \frac{1}{16} & 0\\ 0 & 3 & 3 & 4\\ 0 & 0 & \frac{27}{2} & -12\\ 0 & 0 & 22 & -\frac{32}{4} \end{array}\right)
$$

Como consecuencia, obtendríamos simultáneamente $a_2 = -8/9 \approx -0.89$ y $a_2 = -4/11 \approx -0.36$, lo que evidencia que el sistema es incompatible. ◂

En el Capítulo **??** examinaremos en detalle el problema de interpolación. Allí se mostrará que este resultado era completamente de esperar: la interpolación única de $n+1$ puntos requiere un polinomio de grado n.

En el caso de sistemas sobredeterminados debemos replantear el objetivo del problema, pues ya no podemos hablar de una solución exacta del sistema. En su lugar, caracterizaremos un vector $x \in \mathbb{R}^m$ como una *mejor aproximación* en un sentido adecuado. La calidad de dicha aproximación se evalúa mediante el residual

$$r(x) = b - Ax \in \mathbb{R}^n.$$

Una mejor aproximación $x \in \mathbb{R}^m$ es aquella que minimiza el residual en cierta norma:

Definición 1.29 (Mejor aproximación) Sean $A \in \mathbb{R}^{n\times m}$ con $n > m$, $b \in \mathbb{R}^n$, y $\|\cdot\|$ una norma en $\mathbb{R}^n$. Un vector $x \in \mathbb{R}^m$ se denomina *mejor aproximación* al problema $Ax = b$ si cumple

$$\|b - Ax\| \;\leq\; \|b - Ay\| \qquad \forall\, y \in \mathbb{R}^m.$$

Distintas normas definen diferentes mejores aproximaciones. Para el siguiente teorema es esencial considerar la norma euclidiana, inducida por el producto interno estándar.

Definición 1.30 (Método de mínimos cuadrados) Sean $A \in \mathbb{R}^{n\times m}$ con $n > m$ y $b \in \mathbb{R}^n$. El *método de mínimos cuadrados* define la mejor aproximación $x \in \mathbb{R}^m$ como la aproximación de la solución de $Ax = b$ cuyo residual tiene la menor norma euclidiana:

$$\|b - Ax\|_2 = \min_{y\in\mathbb{R}^m} \|b - Ay\|_2. \tag{1.7}$$

La solución $x \in \mathbb{R}^m$ se denomina también *solución de mínimos cuadrados*.

La búsqueda de la mejor aproximación en la norma euclidiana está estrechamente relacionada con la mejor aproximación de funciones, es decir, con la aproximación de valores medidos mediante funciones sencillas (el llamado "ajuste"), tarea que estudiaremos en el Capítulo **??**.

Teorema 1.31 (Mínimos cuadrados) *Supongamos que $A \in \mathbb{R}^{n\times m}$ con $n > m$ satisface* $\operatorname{rank}(A) = m$. *Entonces la matriz $A^TA \in \mathbb{R}^{m\times m}$ es definida positiva, y la mejor aproximación $x \in \mathbb{R}^m$ queda determinada de manera única como solución del* sistema de ecuaciones normales

$$A^TAx = A^Tb.$$

Este sistema se denomina sistema de ecuaciones normales asociado al sistema de ecuaciones $Ax = b$.

Demostración. (i) Como $\operatorname{rank}(A) = m$, la ecuación $Ax = 0$ implica necesariamente que $x = 0$. De ello se deduce que la matriz A^TA es definida positiva:

$$(A^TAx, x)_2 = (Ax, Ax)_2 = \|Ax\|_2^2 > 0 \qquad \forall\, x \neq 0,$$

y, en consecuencia, el sistema $A^TAx = A^Tb$ admite una única solución para cualquier $b \in \mathbb{R}^n$.

(ii) Sea ahora $x \in \mathbb{R}^m$ la solución del sistema de ecuaciones normales. Para todo $y \in \mathbb{R}^m$ se tiene

$$
\begin{aligned}
\|b - A(x+y)\|_2^2 &= \|b - Ax\|_2^2 + \|Ay\|_2^2 - 2(b - Ax, Ay)_2\\
&= \|b - Ax\|_2^2 + \underbrace{\|Ay\|_2^2}_{\geq 0} - 2(\underbrace{A^Tb - A^TAx}_{=0}, y)_2 \geq \|b - Ax\|_2^2.
\end{aligned}
$$

(iii) Recíprocamente, supongamos que x es un minimizador de (1.7). Entonces, para cualquier $y \in \mathbb{R}^m$ se cumple

$$\begin{aligned}\|b - Ax\|_2^2 &\leq \|b - A(x+y)\|_2^2 \\ &= \|b - Ax\|_2^2 + \|Ay\|_2^2 - 2(b - Ax, Ay)_2.\end{aligned}$$

Por tanto,

$$2(A^T b - A^T Ax, y)_2 \leq \|Ay\|_2^2.$$

Tomando $y = se_i$, donde e_i es el i-ésimo vector canónico y $s \in \mathbb{R}$, obtenemos

$$2s(A^T b - A^T Ax)_i \leq |s|^2 \|A\|^2 \qquad \Rightarrow \qquad |(A^T b - A^T Ax)_i| \leq |s| \|A\|^2.$$

Al tomar el límite cuando $s \to 0$, se concluye que $A^T Ax = A^T b$.

□

La mejor aproximación —en la norma euclidiana— de un sistema sobredeterminado puede obtenerse resolviendo el sistema de ecuaciones normales. Sin embargo, la estrategia directa de formar explícitamente la matriz $A^T A$ y resolver después dicho sistema mediante el método de Cholesky no es recomendable desde el punto de vista numérico. Por un lado, el costo computacional de construir $A^T A$ es considerable; por otro, el propio cálculo del producto de matrices es una operación mal condicionada. Además, la estimación

$$\operatorname{cond}(A^T A) = \|A^T A\| \, \|(A^T A)^{-1}\|$$

muestra que el número de condición de $A^T A$ puede ser significativamente mayor. De hecho, para una matriz regular se cumple la cota $\operatorname{cond}(A^T A) \leq \operatorname{cond}(A)^2$.

En el siguiente ejemplo analizaremos este método, previsiblemente inestable. Retomemos el sistema sobredeterminado del Ejemplo 1.28.

Ejemplo 1.32 (Resolución de las ecuaciones normales) La solución exacta del sistema de ecuaciones normales $A^T Ax = A^T b$ está dada por

$$x \approx \begin{pmatrix} 0.3425 \\ 0.3840 \\ -0.1131 \end{pmatrix} \quad \Rightarrow \quad p(x) = 0.3425 + 0.3740x - 0.1131x^2. \tag{1.8}$$

Se cumple

$$\|b - Ax\|_2 \approx 0.947.$$

Planteamos el sistema de ecuaciones normales con cálculo a tres cifras significativas:

$$A^T A = \begin{pmatrix} 4 & 4.75 & 10.6 \\ 4.75 & 10.6 & 23.7 \\ 10.6 & 23.7 & 55.1 \end{pmatrix}, \quad A^T b = \begin{pmatrix} 2 \\ 3 \\ 6.5 \end{pmatrix}.$$

Utilizando aritmética de tres cifras significativas determinamos la factorización de Cholesky mediante el método directo del Algoritmo **??**:

$$\begin{aligned} l_{11} &= \sqrt{4} = 2 \\ l_{21} &= 4.75/2 \approx 2.38 \\ l_{31} &= 10.6/2 = 5.3 \\ l_{22} &= \sqrt{10.6 - 2.38^2} \approx 2.22 \\ l_{32} &= (23.7 - 2.38 \cdot 5.3)/2.22 \approx 5 \\ l_{33} &= \sqrt{55.1 - 5.3^2 - 5^2} \approx 1.42. \end{aligned} \quad , \quad L := \begin{pmatrix} 2 & 0 & 0 \\ 2.38 & 2.22 & 0 \\ 5.3 & 5 & 1.42 \end{pmatrix}.$$

Resolvemos

$$L \underbrace{L^T x}_{=y} = A^T b \quad \Rightarrow \quad y \approx \begin{pmatrix} 1 \\ 0.279 \\ -0.137 \end{pmatrix} \quad \Rightarrow \quad \tilde{x} \approx \begin{pmatrix} 0.348 \\ 0.345 \\ -0.0972 \end{pmatrix}$$

con el polinomio

$$p(x) = 0.348 + 0.345x - 0.0972x^2$$

y el residual $\|b - A\tilde{x}\| \approx 0.95$, así como el error relativo respecto de la solución exacta x de (1.8)

$$\frac{\|\tilde{x} - x\|}{\|x\|} \approx 0.08.$$

Una desviación del 8 % para una matriz de tamaño 4×3 ya indica una amplificación del error considerable. ◂

Desarrollaremos ahora un método alternativo que evita la construcción del sistema de ecuaciones normales $A^TAx = A^Tb$ y que, en su lugar, utiliza la factorización QR para matrices no cuadradas, introducida en la Observación 1.6. Sea $A \in \mathbb{R}^{n\times m}$ y consideremos la factorización

$$A = QR,$$

donde $Q \in \mathbb{R}^{n\times m}$ y $R \in \mathbb{R}^{m\times m}$. Alternativamente, se puede prolongar R a una matriz $\tilde{R} \in \mathbb{R}^{n\times m}$ añadiendo $n-m$ filas nulas y prolongar Q a una matriz $\tilde{Q} \in \mathbb{R}^{n\times n}$ con $n-m$ vectores adicionales ortogonales entre sí y a las columnas existentes.

Con esta factorización QR generalizada obtenemos

$$A^TAx = A^Tb \quad \Leftrightarrow \quad (QR)^TQRx = (QR)^Tb \quad \Leftrightarrow \quad R^TRx = R^TQ^Tb.$$

La (pequeña) matriz $R \in \mathbb{R}^{m\times m}$ es regular y la solución del sistema de ecuaciones normales viene dada por

$$Rx = Q^Tb.$$

Ejemplo 1.33 (Mejor aproximación de un sistema sobredeterminado mediante factorizaciones QR) Partimos del sistema lineal sobredeterminado del Ejemplo 1.28.

$$A = \begin{pmatrix} 1 & -\frac{1}{4} & \frac{1}{16} \\ 1 & \frac{1}{2} & \frac{1}{4} \\ 1 & 2 & 4 \\ 1 & \frac{5}{2} & \frac{25}{4} \end{pmatrix}, \quad b = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}.$$

En el primer paso de la transformación de Householder —con redondeo a tres cifras significativas— elegimos

$$v^{(1)} = \frac{a_1 + \|a_1\|e_1}{\| a_1 + \|a_1\|e_1 \|} \approx \begin{pmatrix} 0.866 \\ 0.289 \\ 0.289 \\ 0.289 \end{pmatrix}.$$

Entonces, con $a_i^{(1)} = a_i - 2(a_i, v^{(1)})v^{(1)}$ obtenemos

$$\tilde{A}^{(1)} \approx \begin{pmatrix} -2 & -2.38 & -5.29 \\ 0 & -0.210 & -1.54 \\ 0 & 1.29 & 2.21 \\ 0 & 1.79 & 4.46 \end{pmatrix} =: (a_1^{(1)}, a_2^{(1)}, a_3^{(1)})$$

y el subvector $\tilde{a}_2^{(1)} = (-0.21, 1.29, 1.79)^T$ conduce a

$$\tilde{v}^{(2)} = \frac{\tilde{a}_2^{(1)} - \|\tilde{a}_2^{(1)}\|\tilde{e}_2}{\| \tilde{a}_2^{(1)} - \|\tilde{a}_2^{(1)}\|\tilde{e}_2 \|} \approx \begin{pmatrix} -0.740 \\ 0.393 \\ 0.546 \end{pmatrix}.$$

Con ello obtenemos

$$\tilde{A}^{(2)} \approx \begin{pmatrix} -2 & -2.38 & -5.29 \\ 0 & 2.22 & 5.04 \\ 0 & 0 & -1.28 \\ 0 & 0 & -0.392 \end{pmatrix} =: (a_1^{(2)}, a_2^{(2)}, a_3^{(2)}).$$

Aún es necesario un tercer paso. Con $\tilde{a}_3^{(2)} = (-1.28, -0.392)$, obtenemos por el mismo procedimiento

$$\tilde{v}^{(3)} = \begin{pmatrix} -0.989 \\ -0.148 \end{pmatrix}.$$

Finalmente, obtenemos

$$\tilde{R} = \tilde{A}^{(3)} \approx \begin{pmatrix} -2 & -2.38 & -5.29 \\ 0 & 2.22 & 5.04 \\ 0 & 0 & 1.34 \\ 0 & 0 & 0 \end{pmatrix}.$$

Para resolver el sistema de ajuste

$$A^TAx = A^Tb \quad \Leftrightarrow \quad Rx = \tilde{b}$$

determinamos primero el segundo miembro según la regla $b^{(i)} = b^{(i-1)} - 2(b^{(i-1)}, v^{(i)})v^{(i)}$:

$$\tilde{b} = \begin{pmatrix} -1 \\ 0.28 \\ -0.155 \end{pmatrix}.$$

Finalmente, resolvemos por sustitución regresiva $Rx = \tilde{b}^{(3)}$:

$$\tilde{x} \approx \begin{pmatrix} 0.343 \\ 0.389 \\ -0.116 \end{pmatrix}$$

y obtenemos el polinomio de interpolación

$$p(x) = 0.343 + 0.389x - 0.116x^2$$

con residual

$$\|b - A\tilde{x}\|_2 \approx 0.948$$

y el error relativo respecto de la solución exacta de mínimos cuadrados

$$\frac{\|\tilde{x} - x\|}{\|x\|} \approx 0.01,$$

es decir, un error de alrededor del 1 % en lugar de casi un 8 % al resolver directamente el sistema normal.

Resumimos los resultados: la mejor aproximación exacta x, la solución de Cholesky del sistema de ecuaciones normales x_{LL} y la solución mediante la factorización QR ampliada x_{QR} vienen dadas por

$$x \approx \begin{pmatrix} 0.343 \\ 0.384 \\ -0.113 \end{pmatrix}, \quad x_{LL} \approx \begin{pmatrix} 0.348 \\ 0.345 \\ -0.0972 \end{pmatrix}, \quad x_{QR} \approx \begin{pmatrix} 0.343 \\ 0.389 \\ -0.116 \end{pmatrix}.$$

El vector residual es

$$b - Ax \approx \begin{pmatrix} -0.240 \\ 0.493 \\ -0.659 \\ 0.403 \end{pmatrix}, \quad b - Ax_{LL} \approx \begin{pmatrix} -0.256 \\ 0.503 \\ -0.649 \\ 0.397 \end{pmatrix}, \quad b - Ax_{QR} \approx \begin{pmatrix} -0.238 \\ 0.492 \\ -0.657 \\ 0.410 \end{pmatrix}.$$

En Python podemos reutilizar nuestra implementación de la factorización QR mediante transformaciones de Householder. Sin embargo, como necesitamos un segundo miembro distinto $\tilde{b}$, también debemos adaptar la manera de calcular $Q^T b$.

```
def b_tilde(V, b):
    b = b.copy()
    for i in range(len(V)):
        b[i:] -= 2 * np.inner(V[i], b[i:]) * V[i]
    return np.array(b[:len(V)])
```

Si aplicamos esto al ejemplo anterior, obtenemos

```
V = qr_householder(A)
bt = b_tilde(V, b)
x_h = backward(A[:len(V),:], bt)
print(f'x = {x_h}')
```

```
x = [ 0.3423  0.3804 -0.1113]
```

De ello resulta el residual

```
print(f'||Ax - b|| = {np.linalg.norm(np.inner(A2, x_h) - b):.4e}')
```

```
||Ax - b|| = 9.4727e-01
```

Alternativamente, podemos utilizar nuestra factorización QR con rotaciones de Givens. Esto conduce a la solución

```
QT = qr_givens(A)
bt = np.dot(QT, b)[:A.shape[1]]
x_g = backward(A[:A.shape[1],:], bt)
print(f'x = {x_g}')
```

```
x = [ 0.3425  0.384  -0.113 ]
```

con el residual

```
print(f'||Ax - b|| = {np.linalg.norm(np.inner(A2, x_g) - b):.4e}')
```

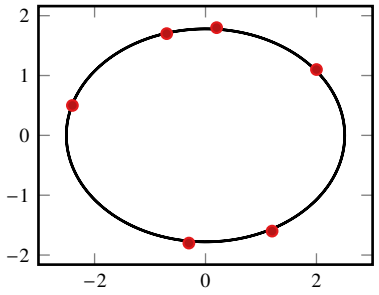
```
||Ax - b|| = 9.4727e-01
```

◀

En la Sección **??** consideraremos la aproximación de Gauss discreta como un método para ajustar funciones a valores discretos. Se trata de una aplicación del método de los mínimos cuadrados. La regresión lineal constituye un caso particular de la aproximación de Gauss discreta. El problema consiste en representar lo mejor posible los pares de datos medidos (x_i, y_i), para $i = 1, \ldots, n$, mediante una relación lineal $y(x) = ax + b$ (véase también el Excursus **??**). Como aplicación de la mejor aproximación en sistemas sobredeterminados, esto conduce a matrices de tamaño $A \in \mathbb{R}^{n\times 2}$.

Otra aplicación de la factorización QR se presentará en el Capítulo **??**. Uno de los métodos más potentes para aproximar los valores propios de una matriz $A \in \mathbb{R}^{n\times n}$ se basa en la aplicación repetida de la factorización QR.

Figura 1.2 Posiciones del cuerpo celeste sobre una elipse en distintos instantes



1.6. Excursus: Determinación de los ejes principales de un cuerpo celeste

En este excursus presentamos una aplicación histórica de los sistemas de ecuaciones sobredeterminados. El desarrollo original del método por Carl Friedrich Gauss surgió precisamente a partir de este problema. En 1801, el astrónomo italiano Piazzi observó el planeta Ceres durante 40 días y posteriormente lo perdió de vista [37]. A partir de los datos disponibles, Gauss aplicó el método de los mínimos cuadrados y calculó numéricamente la órbita del planeta—por supuesto, sin una computadora—. Gracias a ello, los astrónomos pudieron encontrar nuevamente al planeta.

Para ilustrar el procedimiento matemático de este problema, consideremos el modelo simplificado del movimiento de un cuerpo celeste sobre una órbita elíptica alrededor del origen:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1.$$

Consideremos la posición del cuerpo en distintos instantes:

t	1	2	3	4	5	6
x	0.2	-0.7	-0.3	1.2	2.0	-2.4
y	1.8	1.7	-1.8	-1.6	1.1	0.5

A partir de estos datos queremos determinar las longitudes de los dos ejes mayores, a y b. Como estas cantidades aparecen de forma no lineal, introducimos primero las variables auxiliares

$$A = \frac{1}{a^2}, \qquad B = \frac{1}{b^2}.$$

Esto conduce al sistema sobredeterminado

$$\begin{pmatrix} 0.04 & 3.24 \\ 0.49 & 2.56 \\ 0.09 & 3.24 \\ 1.44 & 2.56 \\ 4.00 & 1.21 \\ 5.76 & 0.25 \end{pmatrix} \begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}.$$

Para resolver este problema sobredeterminado, formulamos directamente las ecuaciones normales y las resolvemos. En efecto, se tiene

$$A^T A \approx \begin{pmatrix} 51.50 & 11.64 \\ 11.64 & 35.63 \end{pmatrix}, \qquad A^T b \approx \begin{pmatrix} 11.82 \\ 13.06 \end{pmatrix}.$$

Así, la solución de $A^T Ax = A^T b$ resulta:

$$x = \begin{pmatrix} A \\ B \end{pmatrix} \approx \begin{pmatrix} 0.158 \\ 0.315 \end{pmatrix}, \qquad a = \frac{1}{\sqrt{A}} \approx 2.51, \qquad b = \frac{1}{\sqrt{B}} \approx 1.78.$$

Una vía alternativa consiste en construir la factorización QR de A. Para $A \in \mathbb{R}^{6\times 2}$ son necesarios dos pasos, a partir de los cuales obtenemos

$$A = QR, \qquad R \approx \left(\begin{array}{cc} -7.18 & -1.62 \\ 0 & -5.74 \\ \hline 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{array}\right)$$

y para el segundo miembro

$$\tilde{b} = Q^T b \approx \begin{pmatrix} -1.65 \\ -1.81 \end{pmatrix}.$$

Así, la solución del sistema se obtiene como antes:

$$\tilde{R}x = \tilde{b} \quad \Rightarrow \quad x \approx \begin{pmatrix} 0.158 \\ 0.315 \end{pmatrix}.$$

En la Figura 1.2 se muestra el resultado gráficamente. Para el residual de la solución de mínimos cuadrados se tiene

$$\|b - Ax\|_2 \approx 0.13.$$

La solución $x \in \mathbb{R}^2$ constituye la mejor aproximación, pero no una solución del sistema lineal sobredeterminado. El residual de 0.13 no proporciona información sobre la calidad de la solución numérica, sino sobre la calidad de los datos. Si la ley elíptica postulada para la órbita del planeta fuera exacta, el residual ofrecería una medida del error en las observaciones.

Referencias

1. M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Yangqing Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
2. L. V. Ahlfors. *Complex Analysis. An Introduction to the Theory of Analytic Functions of One Complex Variable*. McGraw-Hill, Inc., 3rd edition, 1979.
3. G. P. Akilov and L. V. Kantorovich. *Functional Analysis in Normed Spaces*. Pergamon Press, 1964.
4. G. S. Almasi and A. Gottlieb. *Highly parallel computing*. Benjamin-Cummings Publishing Co., 2 edition, 1994.
5. H. Amann and J. Escher. *Analysis I*. Birkhäuser, 2006.
6. K. Bach and F. Otto, editors. *Seifenblasen. Eine Forschungsarbeit des Instituts für leichte Flächenwerke über Minimalflächen = Forming bubbles.* Mitteilungen des Instituts für Leichte Flächentragwerke. Kämer, Stuttgart, 1980. ISBN 3-7828-2018-5.
7. C. Bär. *Elementare Differentialgeometrie*. De Gruyter, Berlin, Boston, 2 edition, 2010.
8. R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, Philadelphia, PA, 1994.
9. U. Becciani, E. Sciacca, M. Bandieramonte, A. Vecchiato, B. Bucciarelli, and M. G. Lattanzi. Solving a very large-scale sparse linear system with a parallel algorithm in the gaia mission. In *2014 International Conference on High Performance Computing Simulation (HPCS)*, pages 104–111, July 2014.
10. P. Benner, A. Cohen, M. Ohlberger, and K. Willcox. *Model Reduction and Approximation: Theory and Algorithms*. SIAM Philadelphia, 2015.
11. J. Bewersdorff. *Algebra für Einsteiger: Von der Gleichungsauflösung zur Galois-Theorie*. Vieweg+Teubner, Wiesbaden, 2007.
12. C. M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.
13. C. M. Bishop and H. Bishop. *Deep Learning. Foundations and Concepts*. Springer Cham, 2024.
14. M. Bonnet. Lecture notes on numerical linear algebra. Méthodes numériques matricielles avancées: Analyse et expérimentaiton, 2022.
15. Bronstein. Formelsammlung. auf CD.
16. S. L. Brunton and J. N. Kutz. *Daten-driven science and engineering: Machine learning, Dynamical Systems, and Control*. Cambridge University Press, 2019.
17. W. Burkhart. *Steuerrecht in der Imkerei*. Steuerbuero Burkhart, 2015.
18. A. Burkov. *The hundred-page machine learning book*. Andriy Burkov, 2019.
19. T. F. Chan. An improved algorithm for computing the singular value decomposition. *ACM Transactions on Mathematical Software*, 8(1):72–83, 1982.
20. F. Cucker and A. G. Corbolan. An alternate proof of the continuity of the roots of a polynomial. *The American Mathematical Monthly*, 96(4):342–345, 1989.
21. G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989.
22. W. Dahmen and A. Reusken. *Numerik für Ingenieure und Naturwissenschaftler*. Springer, 2008.
23. J. Demmel, M. Gu, S. Eisenstat, I. Slapničar, K. Veselić, and Z. Drmač. Computing the singular value decomposition with high relative accuracy. *Linear Algebra and its Applications*, 299:21–80, 1999.
24. Bundesministerium der Justiz und Verbraucherschutz. Einkommensteuergesetz. 2002.
25. P. Deuflhard. *Newton Methods for Nonlinear Problems. Affine Invariance and Adaptive Algorithms*, volume 35 of *Computational Mathematics*. Springer, 2011.
26. P. Deuflhard and F. Bornemann. *Gewöhnliche Differentialgleichungen*. De Gruyter, 2008.
27. Duden. *Rechnen und Mathematik: Das Lexikon für Schule und Praxis*. Dudenverlag, 5. ueberarbeitete Auflage, 1994.
28. C. Eck, H. Garcke, and P. Knabner. *Mathematische Modellierung*. Springer, 2008.
29. J. F. Epperson. *An introduction to numerical methods and analysis*. John Wiley & Sons, 2007.
30. D. Etling. *Theoretische Meteorologie*. Springer, 2008.
31. L. C. Evans. *Partial Differential Equations*. Graduate Studies in Mathematics. American Mathematical Society, 2010.
32. J. D. Faires and R. L. Burdon. *Numerische Methoden*. Spektrum Akademischer Verlag, 1994.
33. G. Fischer. *Lineare Algebra*. Vieweg + Teubner, 2010.
34. A. Fog. Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for intel, amd and via cpus. Technical report, 2016. Zugriff am 22.05.2016.
35. E. Freitag and R. Busam. *Funktionentheorie 1*. Springer, 2006.
36. R. Freund and R. Hoppe. *Stoer/Bulirsch: Numerische Mathematik 1*. Springer, 2007.
37. S. Froeba and A. Wassermann. *Die bedeutendsten Mathematiker*. Marixverlag, 2007.
38. C. Gambella, B. Ghaddar, and J. Naoum-Sawaya. Optimization problems for machine learning: A survey. *European Journal of Operational Research*, 290(3):807–828, 2021.
39. C. Geiger and C. Kanzow. *Numerische Verfahren zur Lösung unrestringierter Optimierungsaufgaben*. Springer, 1999.
40. C. Geiger and C. Kanzow. *Theorie und Numerik restringierter Optimierungsaufgaben*. Springer, 2002.
41. G. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *SIAM Journal of Numerical Analysis*, 2(2):205–224, 1965.
42. G. H. Golub and C. Reinsch. Singular value decomposition and least squares solutions. *Numer. Math.*, 14:403–420, 1970.
43. G. H. Golub and C. F. van Loan. *Matrix computations*. Johns Hopkins University Press, 3rd edition, 1996.
44. I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

45. A. Grama, A. Gupta, G. Karypis, and V. Kumar. *Introduction to Parallel Computing*. Addison-Wesley, 2003. 2nd edition.
46. Benedikt Großer and Bruno Lang. Efficient parallel reduction to bidiagonal form. *Parallel Computing*, 25(8):969–986, 1999.
47. C. Großmann and H. G. Roos. *Numerische Behandlung partieller Differentialgleichungen*. Teubner, 2005.
48. W. Hackbusch. *Iterative Lösung großer schwachbesetzter Gleichungssysteme*. Number 69 in Leitfäden der angewandten Mathematik und Mechanik. Springer, 1993.
49. G. H"ammerlin and K.-H. Hoffmann. *Numerische Mathematik*. Springer Verlag, 1992.
50. M. Hanke-Bourgeois. *Grundlagen der numerischen Mathematik und des Wissenschaftlichen Rechnens*. Vieweg-Teubner Verlag, 2009.
51. C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.
52. H. Heuser. *Lehrbuch der Analysis Teil 2*. Vieweg+Teubner, 14 edition, 2012.
53. C. F. Higham and D. J. Higham. Deep learning: An introduction for applied mathematicians. *SIAM Review*, 61(4):860–891, 2019.
54. K. Hornik. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.
55. M. Huber, B. Gmeiner, U. Rüde, and B. Wohlmuth. Resilience for massively parallel multigrid solvers. *SIAM Journal on Scientific Computing*, 38(5):S217–S239, 2016.
56. D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the I.R.E.*, pages 1098–1101, 1952. Aufgerufen 09/2016.
57. IEEE. IEEE 754-2008: Standard for floating-point arithmetic. Technical report, 2008.
58. W. Keiper, A. Milde, and S. Volkwein. *Reduced-Order Modeling (ROM) for Simulation and Optimization*. Springer Verlag, 2018.
59. J. Kepler. *Nova stereometria doliorum vinariorum*. online by courtesy of Carnegie Mellon University Libraries, 1615.
60. H. Kielhöfer. *Variationsrechnung: Eine Einführung in die Theorie einer unabhängigen Variablen mit Beispielen und Anwendungen*. Vieweg+Teubner Verlag, 2010.
61. M. Koecher. *Klassische elementare Analysis*. Springer, 1987.
62. K. Königsberger. *Analysis 1*. Springer, 6 edition, 2004.
63. R. Kress. *Numerical Analysis*. Springer Verlag, 1998.
64. W. Ma, Y. Ao, C. Yang, and S. Williams. Solving a trillion unknowns per second with hpgmg on sunway taihulight. *Cluster Computing*, 23(2):493–507, June 2020.
65. G. Maess. *Vorlesungen über numerische Mathematik II, Analysis*. Birkhäuser Verlag, 1988.
66. S. Mandal, A. Ouazzi, and S. Turek. Modified Newton solver for yield stress fluids. In *Proceedings of ENUMATH 2015, the 11th European Conference on Numerical Mathematics and Advanced Applications*, volume 112 of *Lecture Notes in Computational Science and Engineering*. Springer, 2016.
67. Marvin Marcus and Henryk Minc. *Introduction to Linear Algebra*. Dover Publications Inc., 1988.
68. C. McEniry. The mathematics behind the fast inverse square root funciton code. Aufgerufen am 22.05.2016, August 2007.
69. M. A. Nielsen. Neural networks and deep learning, 2018.
70. J. Nocedal and S. J. Wright. *Numerical optimization*. Springer Ser. Oper. Res. Financial Engrg., 2006.
71. B. N. Parlett. *The Symmetric Eigenvalue Problem*. Classics in Applied Mathematics. SIAM, 1998.
72. A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, K. A., E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.
73. D. Piekenbrock. *Einfuehrung in die Volkswirtschaftslehre und Mikrooekonomie*. Physica-Verlag HD, 2008.
74. G. Pólya. über die konvergenz von quadraturverfahren. *Mathematische Zeitschrift*, 37:264–286, 1933.
75. R. Rannacher. *Einführung in die Numerische Mathematik*. Heidelberg University Publishing, 2017.
76. R. Rannacher. *Numerik gewöhnlicher Differentialgleichungen*. Heidelberg University Publishing, 2017.
77. R. Rannacher. *Numerik partieller Differentialgleichungen*. Heidelberg University Publishing, 2017.
78. R. Rannacher. *Probleme der Kontinuumsmechanik und ihre numerische Behandlung*. Heidelberg University Publishing, 2017.
79. T. Rauber and G. Rünger. *Parallele Programmierung*. Springer, 2 edition, 2007.
80. H.-J. Reinhardt. *Numerik gewöhnlicher Differentialgleichungen. Anfangs- und Randwertprobleme.* Berlin, Boston: De Gruyter, 2008.
81. C. Reinsch and M. Richter. Singular value decomposition in extended double precision arithmetic. *Numer. Algorithms*, 93(3):1137–1155, 2023.
82. Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, Philadelphia, PA, 2 edition, 2003.
83. Y. Saad. *Numerical methods for large eigenvalue problems*, volume 66 of *Classics in Applied Mathematics*. SIAM publications, 2011.
84. R. Schaback and H. Wendland. *Numerische Mathematik*. Springer, 2005.
85. J. Schüle. *Paralleles Rechnen*. de Gruyter, 2010.
86. H. R. Schwarz and N. Köckler. *Numerische Mathematik*. Vieweg-Teubner Verlag, 2011.
87. G. W. Stewart. *Matrix Algorithms. Volume II: Eigensystems*. SIAM publications, 2001.
88. Gilbert Strang. *Introduction to linear algebra*. Wellesley-Cambridge Press, Wellesley, MA, sixth edition, 2023.
89. T. Strutz. *Bilddatenkompression. Grundlagen, Codierung, Wavelets, JPEG, MPEG, H.264*. Praxiswissen. Vieweg+Teubner Verlag, 2005.
90. Terence Tao. *Analysis I*. Springer, 2022.
91. Terence Tao. *Analysis II*. Springer, 2022.
92. L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM, 1997.
93. E. E. Tyrtyshnikov. *A Brief introduction to Numerical Analysis*. Birkhäuser, 1997.
94. M. Ulbrich and S. Ulbrich. *Nichtlineare Optimierung*. Mathematik Kompakt. Birkhäuser, 2011.
95. D. Werner. *Funktionalanalysis*. Springer, 2004.
96. Wikipedia. Fast inverse square root. https://en.wikipedia.org/wiki/Fast_inverse_square_root, Zuletzt aufgesucht am 24.12.2016.
97. J. H. Wilkinson. *The algebraic eigenvalue problem*. Numerical Mathematics and Scientific Computation. Oxford Science Publications, 1965.
98. J. Wloka. *Partielle Differentialgleichungen*. Teubner, Stuttgart, 1982.
99. W. Zulehner. *Numerische Mathematik: Ein Einfuehrung anhand von Differentialgleichungsproblemen; Band 2: Instationaere Probleme*. Mathematik Kompakt. Birkhaeuser Verlag, 2011.

Índice alfabético