FDU 数值算法 2. 稠密线性最小二乘问题的解法

本文根据邵老师授课内容整理而成，并参考了以下教材:

- 数值线性代数 (第二版) 徐树方, 高立, 张平文 第 3 章
- Applied Numerical Linear Algebra (James W. Demmel) 第 3 章
- Iterative Methods for Sparse Linear Systems (2nd Edition, Yousef Saad) 第 1 章
- Matrix Computations (4th Edtion, G. Golub, C. Van Loan) 第 $6, 9$ 章

欢迎批评指正!

2.1 线性最小二乘问题

2.1.1 基本定义

最小二乘问题 (Least Squares) 多产生于数据拟合问题.

给定 m 个点 $t_1, \dots, t_n$ 和观测数据 $y_1, \dots, y_m$ 以及 n 个已知函数 $\phi_1(\cdot), \dots, \phi_n(\cdot)$
考虑 ϕ_i 的线性组合 $f(x;t) = \sum_{k=1}^{n} x_k \phi_k(t)$,
我们希望它在点 $t_1, \dots, t_n$ 上能最佳地逼近 $y_1, \dots, y_m$
为此我们定义残差 $r_i(x) = y_i - f(x;t_i) = y_i - \sum_{k=1}^{n} x_k \phi_k(t_i) \ \ (i = 1, \dots, m)$
并定义以下记号:

$$A = [\phi_1(t), \dots, \phi_n(t)] = \begin{bmatrix} \phi_1(t_1) & \cdots & \phi_n(t_1) \\ \vdots & & \vdots \\ \phi_1(t_m) & \cdots & \phi_n(t_m) \end{bmatrix} \qquad b = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} \qquad x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \qquad r(x) = \begin{bmatrix} r_1(x) \\ \vdots \\ r_m(x) \end{bmatrix}$$

因此 $r(x) = b - Ax$

- 当 $m = n$ 时，我们通常可以要求 $r(x) = 0$，即等价于求解线性方程组 $Ax = b$
- 当 $m > n$ 时，$r(x) = 0$ 通常是无解的 (它称为**超定方程组**或**矛盾方程组**)
  此时我们只能要求残差向量 $r(x)$ 在某种范数意义下最小，
  而线性最小二乘问题就是要求残差向量 $r(x)$ 在 2 范数意义下最小.

给定 $\begin{cases} A \in \mathbb{R}^{m \times n} \\ b \in \mathbb{R}^{m} \end{cases}$ 我们称 $\min\limits_{x \in \mathbb{R}^n} \|r(x)\|_2 = \min\limits_{x \in \mathbb{R}^n} \|b - Ax\|_2$ 为**最小二乘问题**.
最小二乘问题 $\min\limits_{x \in \mathbb{R}^n} \|Ax - b\|_2$ 的解 x_{ls} 称为线性方程组 $Ax = b$ 的**最小二乘解**.

- 除非另有规定，本课程中我们**默认假设** $\operatorname{rank}(A) = n < m$.
  后面我们可以看到这个假设保证了最小二乘解的唯一性.
- 在我们的假设中，残差向量 $r(x)$ 线性地依赖于 x，因此称为**线性最小二乘问题**.
  更一般地，若 $r(x)$ 非线性地依赖于 x，则称其为非线性最小二乘问题.

另一种视角:
考虑线性方程组 $Ax = b$
假设 $A \in \mathbb{C}^{m \times n}$ 的奇异值分解为 $A = U \Sigma V^{\text{H}}$，其中 Σ 的结构为:

$$\Sigma = \begin{bmatrix} \Sigma_r & 0_{r \times (n-r)} \\ 0_{(m-r) \times r} & 0_{(m-r) \times (n-r)} \end{bmatrix} \text{ where diagonal matrix } \Sigma_r \succ 0$$

则线性方程组 $Ax = b$ 可化为:

$$\begin{gathered} Ax = U \Sigma V^{\text{H}} x = b \\ \Leftrightarrow \\ \text{denote } y := V^{\text{H}} x \\ \Sigma y = U^{\text{H}} b = c \\ \Leftrightarrow \\ \begin{bmatrix} \Sigma_r & 0_{r \times (n-r)} \\ 0_{(m-r) \times r} & 0_{(m-r) \times (n-r)} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \end{gathered}$$

显然当且仅当 $c_2 = 0_{m-r}$ 时上述方程组有解，解的结构为:

$$\begin{cases} y_1 = \Sigma_r^{-1} c_1 \\ y_2 \in \mathbb{C}^{m-r} \end{cases}$$

其中 $y_1 = \Sigma_r^{-1} c_1$ 和 $y_2 \in \mathbb{C}^{m-r}$ (不严格地来讲) 分别负责特解和通解.

当 $c_2 = 0_{m-r}$ 时，上述方程组无解，我们转而估计残差的 l_2 范数 $\|b - Ax\|_2$:

$$\begin{aligned} \|b - Ax\|_2^2 &= \|b - U\Sigma V^{\mathrm{H}} x\|_2^2 \\ &= \|U^{\mathrm{H}} b - \Sigma V^{\mathrm{H}} x\|_2^2 \quad \left(\text{note that } \begin{cases} c = U^{\mathrm{H}} b \\ y = V^{\mathrm{H}} x \end{cases}\right) \\ &= \|c - \Sigma y\|_2^2 \\ &= \left\| \begin{bmatrix} c_1 - \Sigma_r y_1 \\ c_2 \end{bmatrix} \right\|_2^2 \\ &= \|c_1 - \Sigma_r y_1\|_2^2 + \|c_2\|_2^2 \\ &\geq \|c_2\|_2^2 \end{aligned}$$

上述不等式当且仅当 $c_1 = \Sigma_r y_1$ (即 $y_1 = \Sigma_r^{-1} c_1$) 时取等 (这预示着最小二乘解是一定存在的).
因此 $\|c_1 - \Sigma_r y_1\|_2^2$ 我们是管得了的，而 $\|c_2\|_2^2$ 我们是管不了的.

奇异值分解作为理论分析的强力手段可以让我们看得更清楚，但实际应用中它还是太昂贵了.
我们发现 V 仅用于 $y = V^{\mathrm{H}} x$ 的变量替换，它不需要是酉矩阵，
但最小二乘的变换中我们用到了 l_2 范数的酉不变性，因此 U 必须是一个酉矩阵.
我们想到用 QR 分解代替奇异值分解来求解最小二乘问题，参见 2.3.1 节的内容.

2.1.2 最小二乘解

我们首先介绍一下线性方程组 $Ax = b$ 的基本性质.
(数值线性代数, 定理 3.1.1)
给定 $\begin{cases} A \in \mathbb{R}^{m \times n} \\ b \in \mathbb{R}^m \end{cases}$ 记 $A = [a_1, \ldots, a_n]$
$Ax = b$ 有解的充要条件是 $\mathrm{rank}([A, b]) = \mathrm{rank}(A)$，即 $b \in \mathrm{Range}(A)$

- **必要性:**
  若 $Ax = b$ 存在解 x_0，则 $b \in \mathrm{Range}(A)$，说明 $\mathrm{rank}([A, b]) = \mathrm{rank}(A)$
- **充分性:**
  若 $\mathrm{rank}([A, b]) = \mathrm{rank}(A)$，则 $b \in \mathrm{Range}(A)$
  即存在 $c_1, \ldots, c_n \in \mathbb{R}$ 使得 $b = \sum_{i=1}^{n} c_i a_i$
  令 $x_0 = (c_1, \ldots, c_n)^{\mathrm{T}}$，即有 $b = Ax$ 成立，表明 $Ax = b$ 有解.

(数值线性代数, 定理 3.1.2)
给定 $\begin{cases} A \in \mathbb{R}^{m \times n} \\ b \in \mathbb{R}^m \end{cases}$ 记 $A = [a_1, \ldots, a_n]$
若 x_0 是 $Ax = b$ 的任意给定的解，则 $Ax + b$ 的解集为 $x_0 + \mathrm{Ker}(A) = \{x_0 + x : Ax = 0_m\}$

- 这个定理表明:
  只要知道了 $Ax = b$ 的一个解，便可以对 $\mathrm{Ker}(A)$ 位移得到 $Ax = b$ 的全部解.
  因此 $Ax = b$ 存在唯一解，当且仅当 $\begin{cases} b \in \mathrm{Range}(A) \\ \mathrm{Ker}(A) = \{0_n\} \end{cases}$
- **证明:**
  一方面，对于任意 $y \in x_0 + \mathrm{Ker}(A)$，
  我们有 $Ay = A(x_0 + y - x_0) = Ax_0 + A(y - x_0) = b + 0_m = b$，
  表明 y 是 $Ax = b$ 的解.

  另一方面，对于 $Ax = b$ 的任意解 y，
  我们有 $A(y - x_0) = Ay - Ax_0 = b - b = 0_m$，
  说明 $y - x_0 \in \mathrm{Ker}(A)$，即有 $y \in x_0 + \mathrm{Ker}(A)$

  综上所述，$Ax + b$ 的解集为 $x_0 + \mathrm{Ker}(A)$

下面我们讨论最小二乘解的存在性和唯一性问题.

给定 $\begin{cases} A \in \mathbb{R}^{m\times n} \\ b \in \mathbb{R}^m \end{cases}$

向量 $b \in \mathbb{R}^m$ 可以唯一地分解为 $\begin{cases} b = b_1 + b_2 \\ b_1 \in \mathrm{Range}(A) \\ b_2 \in \mathrm{Range}(A)^\perp \end{cases}$ $(\mathbb{R}^m = \mathrm{Range}(A) \oplus \mathrm{Range}(A)^\perp)$

当 x 取遍 $\mathbb{R}^n$ 时，$y = Ax$ 会取遍整个 $\mathrm{Range}(A)$

当残差向量 $r = b - y = b - Ax$ 垂直于 $\mathrm{Range}(A)$，即 $r = b_2 \in \mathrm{Range}(A)^\perp$ 时，$\|b - y\|_2$ 达到极小.

此时 $y_{\min} = b_1$，对应的残差向量 $b_2 = b - y_{\min}$

有了 $y_{\min}$，只要求解 $Ax = y_{\min}$ 即可得到最小二乘解 x_{ls}

(数值线性代数, 定理 3.1.3)

线性最小二乘问题 $\min\limits_{x\in\mathbb{R}^n} \|Ax - b\|_2$ 的解总是存在的.

最小二乘解唯一的充要条件是 $\mathrm{Ker}(A) = \{0_n\}$

- 记最小二乘解集 $\mathcal{X}_{\mathrm{ls}} := \arg\min\limits_{x\in\mathbb{R}^n} \|b - Ax\|_2^2$，由上述定理可知 $\mathcal{X}_{\mathrm{ls}}$ 总是非空的.

  除非另有规定，本课程中我们**默认假设** $\mathrm{rank}(A) = n < m$.

  这个假设保证了 $\begin{cases} \mathrm{Range}(A) = \mathbb{R}^m \\ \mathrm{Ker}(A) = \{0_n\} \end{cases}$，即保证了 $\mathcal{X}_{\mathrm{ls}}$ 只有唯一的元素.

  在此假设下，我们记这个唯一的最小二乘解为 x_{ls}

- **证明:**

  向量 $b \in \mathbb{R}^m$ 可以唯一地分解为 $\begin{cases} b = b_1 + b_2 \\ b_1 \in \mathrm{Range}(A) \\ b_2 \in \mathrm{Range}(A)^\perp \end{cases}$ $(\mathbb{R}^m = \mathrm{Range}(A) \oplus \mathrm{Range}(A)^\perp)$

  于是对于任意 $x \in \mathbb{R}^n$，都有 $b_1 - Ax \in \mathrm{Range}(A)$，从而有:

  $$\begin{aligned} \|r(x)\|_2^2 &= \|b - Ax\|_2^2 \\ &= \|(b_1 - Ax) + b_2\|_2^2 \quad (b_1 - Ax \perp b_2) \\ &= \|b_1 - Ax\|_2^2 + \|b_2\|_2^2 \end{aligned}$$

  根据 $b_1 \in \mathrm{Range}(A)$ 可知 $Ax = b_1$ 是可解的

  因此 $\|b_1 - Ax\|_2^2 \geq 0$，当且仅当 x 是 $Ax = b_1$ 的解时取等.

  于是我们有 $\|r(x)\|_2^2 \geq \|b_2\|_2^2$，当且仅当 x 是 $Ax = b_1$ 的解时取等.

  所以最小二乘解总是存在的，其唯一的充要条件是 $\mathrm{Ker}(A) = \{0_n\}$.

(数值线性代数, 定理 3.1.4)

给定 $\begin{cases} A \in \mathbb{R}^{m\times n} \\ b \in \mathbb{R}^m \end{cases}$ 考虑线性最小二乘问题 $\min\limits_{x\in\mathbb{R}^n} \|Ax - b\|_2$

$x \in \mathcal{X}_{\mathrm{ls}}$ 当且仅当 $A^{\mathrm{T}}Ax = A^{\mathrm{T}}b$

- 向量 $b \in \mathbb{R}^m$ 可以唯一地分解为 $\begin{cases} b = b_1 + b_2 \\ b_1 \in \mathrm{Range}(A) \\ b_2 \in \mathrm{Range}(A)^\perp \end{cases}$ $(\mathbb{R}^m = \mathrm{Range}(A) \oplus \mathrm{Range}(A)^\perp)$
- **必要性证明:**

  若 $x \in \mathcal{X}_{\mathrm{ls}}$，则我们有 $Ax = b_1$ 成立，于是 $r(x) = b - Ax = b - b_1 = b_2 \in \mathrm{Range}(A)^\perp = \mathrm{Ker}(A^{\mathrm{T}})$

  因此 $A^{\mathrm{T}}r(x) = A^{\mathrm{T}}(b - Ax) = 0$，即有 $A^{\mathrm{T}}Ax = A^{\mathrm{T}}b$
- **充分性证明:**

  若 $A^{\mathrm{T}}Ax = A^{\mathrm{T}}b$，则对于任意 $y \in \mathbb{R}^n$ 有:

  $$\begin{aligned} \|b - A(x + y)\|_2^2 &= \|b - Ax\|_2^2 - 2y^{\mathrm{T}}A^{\mathrm{T}}(b - Ax) + \|Ay\|_2^2 \\ &= \|b - Ax\|_2^2 + \|Ay\|_2^2 \\ &\geq \|b - Ax\|_2^2 \end{aligned}$$

  即有 $x \in \mathcal{X}_{\mathrm{ls}}$ 成立.

方程组 $A^{\mathrm{T}}Ax = A^{\mathrm{T}}b$ 称为最小二乘问题的**正则化方程组**或**法方程组**，

它是一个含有 n 个变量和 n 个方程的线性方程组.

在 A 列线性无关 (即 $\mathrm{Range}(A) = n$) 的条件下，$A^{\mathrm{T}}A$ 是正定阵，

因此我们可以根据 **Cholesky 分解 (平方根法)** 求解方程组 $A^{\mathrm{T}}Ax = A^{\mathrm{T}}b$

这样我们就得到了求解最小二乘问题最古老的算法——**正则化方法**.

- 计算 $\begin{cases} C = A^{\mathrm{T}}A \\ d = A^{\mathrm{T}}b \end{cases}$

  值得注意的是，在 $C = A^{\mathrm{T}}A$ 的计算中，若不使用足够的精度，则矩阵 A 的一些信息可能会丢失.

- 使用平方根法计算 C 的 Cholesky 分解 $C = LL^{\mathrm{T}}$
- 求解三角方程组 $\begin{cases} Ly = d \\ L^{\mathrm{T}}x = y \end{cases}$

注意正则化方程组 $A^{\mathrm{T}}Ax = A^{\mathrm{T}}b$ 的解 x 可以表示为 $x = (A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}}b$
我们定义 $A^{\dagger} = (A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}} \in \mathbb{R}^{n\times m}$ (它是 A 的 **Moore-Penrose 广义逆**的特殊情况)
在 A 列线性无关 (即 $\mathrm{Range}(A) = n$) 的条件下，$A^{\dagger}$ 满足 Penrose 方程组 $\begin{cases} AXA = A \\ XAX = X \\ (AX)^{\mathrm{T}} = AX \\ (XA)^{\mathrm{T}} = XA \end{cases}$
于是这个解又可以记为 $x = A^{\dagger}b$

下面的定理给出了 b 的扰动引起的最小二乘解 x 的相对误差的界.
(A, b 的同时扰动对最小二乘解 x 的影响是一个非常复杂的问题，我们不进行讨论)

(数值线性代数, 定理 3.1.5)
给定 $\begin{cases} A \in \mathbb{R}^{m\times n} \\ b \in \mathbb{R}^m \end{cases}$ 考虑线性最小二乘问题 $\min\limits_{x\in\mathbb{R}^n} \|Ax - b\|_2^2$ (默认假设 $\mathrm{rank}(A) = n < m$)
假定 b 有扰动 δb 且 $\begin{cases} x = \arg\min\limits_{x\in\mathbb{R}^n} \|b - Ax\|_2^2 \\ x + \delta x = \arg\min\limits_{x\in\mathbb{R}^n} \|(b+\delta b) - Ax\|_2^2 \end{cases}$
即有 $\begin{cases} x = A^{\dagger}b \\ x + \delta x = A^{\dagger}(b + \delta b) \end{cases}$

设 b_1 和 $\tilde{b}_1$ 分别是 b 和 $b + \delta b$ 在 $\mathrm{Range}(A)$ 上的正交投影.
若 $b_1 \neq 0_m$，则 $\frac{\|\delta x\|_2}{\|x\|_2} \leq \kappa_2(A)\frac{\|b_1 - \tilde{b}_1\|_2}{\|b_1\|_2}$ (其中 $\kappa_2(A) = \|A\|_2\|A^{\dagger}\|_2$ 是广义条件数)

- **证明:**
  在 $\mathrm{rank}(A) = n < m$ 的假设下，$A^{\mathrm{T}}A$ 正定.
  因此 $\mathrm{Ker}(A^{\dagger}) = \mathrm{Ker}((A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}}) = \mathrm{Ker}(A^{\mathrm{T}}) = \mathrm{Range}(A)^{\perp}$
  于是我们有 $\begin{cases} A^{\dagger}b = A^{\dagger}b_1 \\ A^{\dagger}(b+\delta b) = A^{\dagger}\tilde{b}_1 \end{cases}$
  从而有:

$$\begin{aligned} \|\delta x\|_2 &= \|A^{\dagger}b - A^{\dagger}(b+\delta b)\|_2 \\ &= \|A^{\dagger}b_1 - A^{\dagger}\tilde{b}_1\|_2 \\ &\leq \|A^{\dagger}\|_2\|b_1 - \tilde{b}_1\|_2 \end{aligned}$$

  若 x 是最小二乘解，则我们有 $Ax = b_1$ 成立，即有 $\|b_1\|_2 = \|Ax\|_2 \leq \|A\|_2\|x\|_2$
  联立 $\begin{cases} \|\delta x\|_2 \leq \|A^{\dagger}\|_2\|b_1 - \tilde{b}_1\|_2 \\ \|x\|_2 \geq \frac{\|b_1\|_2}{\|A\|_2} \end{cases}$ 即得 $\frac{\|\delta x\|_2}{\|x\|_2} \leq \|A\|_2\|A^{\dagger}\|_2\frac{\|b_1 - \tilde{b}_1\|_2}{\|b_1\|_2}$
- 这个定理表明:
  只有 b 在 $\mathrm{Range}(A)$ 上的投影 b_1 的变化会对最小二乘解产生影响.
  此外，该影响的敏感性依赖于**最小二乘问题的条件数** $\kappa_2(A) = \|A\|_2\|A^{\dagger}\|_2$.
  若 $\kappa_2(A)$ 很小，则我们称该最小二乘问题是良态的; 否则称为病态的.

下面的定理给出了 $\kappa_2(A) = \|A\|_2\|A^{\dagger}\|_2$ 与 $\kappa_2(A^{\mathrm{T}}A)$ 之间的关系.
(数值线性代数, 定理 3.1.6)
若 A 列线性无关 (这保证了 $A^{\dagger} = (A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}}$)，则 $(\kappa_2(A))^2 = \kappa_2(A^{\mathrm{T}}A)$

- **证明:**
  根据谱范数的性质，我们有 $\begin{cases} \|A\|_2^2 = \|A^{\mathrm{T}}A\|_2 \\ \|A^{\dagger}\|_2^2 = \|A^{\dagger}(A^{\dagger})^{\mathrm{T}}\|_2 = \|(A^{\mathrm{T}}A)^{-1}\|_2 \end{cases}$
  于是 $(\kappa_2(A))^2 = \|A\|_2^2\|A^{\dagger}\|_2^2 = \|A^{\mathrm{T}}A\|_2\|(A^{\mathrm{T}}A)^{-1}\|_2 = \kappa_2(A^{\mathrm{T}}A)$
- 这个定理表明:
  最小二乘问题 $\min\limits_{x\in\mathbb{R}^n} \|Ax - b\|_2^2$ 化为正则化方程 $(A^{\mathrm{T}}A)x = A^{\mathrm{T}}b$ 会导致条件数变为原来的平方.
  这增加了求解过程对舍入误差的敏感性.
  因此在使用正则化方法时，要特别注意这一点.

2.2 初等正交变换

为给出求解最小二乘问题的更实用的算法，我们来介绍两个最基本的初等正交变换.
它们是数值线性代数中许多重要算法的基础.

邵老师提到的一些有趣的事实:

- 行列式等于 1 的就是旋转变换，行列式等于 -1 的就是镜像变换 (偶数次镜像变换的复合就是旋转变换)
- 一次空间旋转 (由行列式为 1 的 3 阶方阵表示) 可以分解为三次平面旋转 (由行列式为 1 的 2 阶方阵表示)
- 旋转变换的本身可以不断逼近单位阵 (而镜像变换做不到)，这在矩阵序列收敛中非常有用.

2.2.1 Householder 变换

回顾 Gauss 变换:

$$l_k = \frac{1}{x_k}\begin{bmatrix} 0 \\ \vdots \\ 0 \\ x_{k+1} \\ \vdots \\ x_n \end{bmatrix} \qquad L_k = I - l_k e_k^{\mathrm{T}} = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -\frac{x_{k+1}}{x_k} & 1 & \\ & & \vdots & & \ddots \\ & & -\frac{x_n}{x_k} & & 1 \end{bmatrix} \quad \Rightarrow \quad L_k x = \begin{bmatrix} x_1 \\ \vdots \\ x_k \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

初等下三角阵 L_k 可将 x 的第 $k+1$ 个至第 n 个分量置为零 (前提是 $x_k \neq 0$)

现在我们来讨论如何求一个初等正交矩阵，来代替 Gauss 变换矩阵的功能.
这样，一个矩阵的上三角化任务便可以由一系列的初等正交变换来完成.

(Householder 变换)
设 $w \in \mathbb{R}^n$ 为单位向量 $(\|w\|_2 = 1)$,
我们定义对应的 Householder 变换 $H := I - 2ww^{\mathrm{T}} \in \mathbb{R}^{n\times n}$ (又称**初等反射矩阵**或**镜像变换**)

(Householder 变换的基本性质, 数值线性代数, 定理 $3.2.1$)
设 $H = I - 2ww^{\mathrm{T}}$ 是单位向量 w 对应的 Householder 变换，则我们有:

- **(对称性)** $H^{\mathrm{T}} = H$
  $H^{\mathrm{T}} = (I—2ww^{\mathrm{T}})^{\mathrm{T}} = I - 2ww^{\mathrm{T}} = H$
- **(正交性)** $H^{\mathrm{T}}H = I$
  $H^{\mathrm{T}}H = H^2 = (I - 2ww^{\mathrm{T}})^2 = I - 4ww^{\mathrm{T}} + 4ww^{\mathrm{T}}ww^{\mathrm{T}} = I - 4ww^{\mathrm{T}} + 4 \cdot 1 \cdot ww^{\mathrm{T}} = I$
- **(对合性)** $H^2 = I$
  $H^2 = H^{\mathrm{T}}H = I$
- **(反射性)** 对于任意 $x \in \mathbb{R}^n$，Hx 是 x 关于 w 的垂直超平面 $\mathrm{span}\{w\}^{\perp} = \{x \in \mathbb{R}^n : w^{\mathrm{T}}x = 0\}$ 的镜像.
  x 可分解为 $\begin{cases} x = x_1 + x_2 \\ x_1 \in \mathrm{span}(w) \\ x_2 \in \mathrm{span}(w)^{\perp} \end{cases}$ (于是有 $\begin{cases} \exists\, \alpha \in \mathbb{R} \text{ such that } x_1 = \alpha w \\ w^{\mathrm{T}}x_2 = 0 \\ x = x_1 + x_2 = \alpha w + x_2 \end{cases}$)
  我们有:

$$\begin{aligned} Hx &= (I - 2ww^{\mathrm{T}})(x_1 + x_2) \\ &= (I - 2ww^{\mathrm{T}})(\alpha w + x_2) \\ &= \alpha w - 2ww^{\mathrm{T}} \cdot \alpha w + x_2 - 2ww^{\mathrm{T}}x_2 \\ &= \alpha w - 2\alpha w \cdot 1 + x_2 - 2w \cdot 0 \\ &= -\alpha w + x_2 \\ &= -x_1 + x_2 \end{aligned}$$

  因此 Hx 是 x 关于 w 的垂直超平面 $\mathrm{span}\{w\}^{\perp} = \{x \in \mathbb{R}^n : w^{\mathrm{T}}x = 0\}$ 的镜像.
  取 $\begin{cases} P_w := ww^{\mathrm{T}} \\ P_{w^{\perp}} := I - ww^{\mathrm{T}} \end{cases}$ 则我们可以将 $H = I - 2ww^{\mathrm{T}}$ 分解为 $H = -P_w + P_{w^{\perp}}$
  其中 $P_{w^{\perp}}$ 是从 $\mathbb{R}^n$ 向 $\mathrm{span}\{w\}^{\perp}$ 的投影算子，而 $P_w = I - P_{w^{\perp}}$ 可提取向量中垂直于 $\mathrm{span}\{w\}^{\perp}$ 的部分.

$$\begin{aligned} x &= (ww^{\mathrm{T}})x + (I - ww^{\mathrm{T}})x \\ &= P_w x + P_{w^{\perp}}x \\ &= x_1 + x_2 \end{aligned}$$

Householder 变换的主要用途在于，它和 Gauss 变换一样，
可以通过适当选取单位向量 w，将给定向量的若干个指定分量置为零.

(数值线性代数, 定理 $3.2.2$)
若 $x \neq 0_n \in \mathbb{R}^n$，则可构造 $\begin{cases} \alpha = \pm\|x\|_2 \\ w = \frac{x-\alpha e_1}{\|x-\alpha e_1\|_2} \\ H = I - 2ww^{\mathrm{T}} \end{cases}$ 使得 $Hx = \alpha e_1$ (其中 e_1 是 $\mathbb{R}^n$ 的第 1 个标准单位基向量)

- **证明:**
  记 $\alpha = \pm\|x\|_2$
  为使 $Hx = (I - 2ww^{\mathrm{T}})x = x - 2(w^{\mathrm{T}}x)w = \alpha e_1$，我们有 $w = \frac{x-\alpha e_1}{\|x-\alpha e_1\|_2}$

一个自然的问题是，实际计算中，α 应当取 $\|x\|_2$ 还是 $-\|x\|_2$?
我们通常取 $\alpha = \|x\|_2$，但这可能造成**相消**的问题:
当 x 的第 1 个分量为正值 $(x_1 > 0)$ 且占主导地位 (即 $\|x\|_2 \approx x_1$) 时，
$x - \|x\|_2 e_1$ 的第 1 个分量便会出现 $x_1 - \|x\|_2$ 这种两个相近的数相减的情况，从而严重地损失有效数字.

但幸运的是，我们可通过等价变形来规避相消问题的出现:

$$x_1 - \|x\|_2 = \frac{x_1^2 - \|x\|_2^2}{x_1 + \|x\|_2} = -\frac{x_2^2 + \cdots + x_n^2}{x_1 + \|x\|_2}$$

当 $x_1 > 0$ 时，上述等价变形便可规避相消问题的出现.
(对于 $x_1 \leq 0$ 的情况，相消问题并不会出现)

其次，记 $\begin{cases} v = x - \alpha e_1 \\ w = \frac{v}{\|v\|_2} \\ \beta = \frac{2}{v^{\mathrm{T}}v} \end{cases}$，我们有:

$$H = I - 2ww^{\mathrm{T}} = I - \frac{2}{v^{\mathrm{T}}v}vv^{\mathrm{T}} = I - \beta vv^{\mathrm{T}}$$

因此我们没必要求出 w，只需求出 v 和 β 即可.
在实际运算中，我们可以将 v 的第一个分量规格化为 1 (这样就无需储存了)
然后将 v 的后 $n-1$ 个分量保存在 x 的后 $n-1$ 个置为 0 的分量上.

最后，$v^{\mathrm{T}}v$ 的上溢和下溢也是计算中需要考虑的问题.
为避免溢出，我们可用 $\frac{x}{\|x\|_\infty}$ 代替 x 来构造 v
因为理论上，正数乘是不影响向量单位化结果的，
即对于任意 $\gamma > 0$，向量 γv 和 v 的单位化结果是相同的.

基于上述讨论，我们得到如下算法:
(计算 Householder 变换, 数值线性代数, 算法 $3.2.1$)

$$
\begin{aligned}
&\text{function: } [v,\beta]=\text{Householder}(x)\\
&\quad n=\text{length}(x)\\
&\quad x=\frac{x}{\|x\|_\infty}\\
&\quad v(2:n)=x(2:n)\\
&\quad \sigma=x(2:n)^{\mathrm{T}}x(2:n)\\
&\quad \text{if } \sigma=0\\
&\qquad \beta=0\\
&\quad \text{else}\\
&\qquad \alpha=\sqrt{x(1)^2+\sigma}\\
&\qquad \text{if } x(1)>0 \quad (\text{规避相消})\\
&\qquad\quad v(1)=-\frac{\sigma}{x(1)+\alpha}\\
&\qquad \text{else} \quad (x(1)\le 0 \text{ 时无需规避相消})\\
&\qquad\quad v(1)=x(1)-\alpha\\
&\qquad \text{end}\\
&\qquad \beta=\frac{2v(1)^2}{v(1)^2+\sigma}\\
&\qquad v=\frac{v}{v(1)}\\
&\quad \text{end}
\end{aligned}
$$

上述算法是数值稳定的.
假定计算结果是 $\tilde{v}$ 和 $\tilde{\beta}$，定义 $\tilde{H}=I-\tilde{\beta}\tilde{v}\tilde{v}^{\mathrm{T}}$
我们可以证明 $\|H-\tilde{H}\|_2=O(\text{eps})$

上述算法拓展到复数域上的结果参见 Homework 4 的 Problem 2

实际上 Householder 变换可以将向量中任意若干个相邻的分量置为零.
例如，欲将 $x\in\mathbb{R}^n$ 的第 $k+1$ 个分量至第 j 个分量置为零，
只需定义 $\begin{cases}\alpha=\pm\sqrt{\sum_{i=1}^{j}x_i^2}\\ v=(0,\dots,0,x_k-\alpha,x_{k+1},\dots,x_j,0,\dots,0)\\ \beta=\frac{2}{v^{\mathrm{T}}v}\\ H=I-2\beta vv^{\mathrm{T}}\end{cases}$

在使用 Householder 变换 $H=I-\beta vv^{\mathrm{T}}\in\mathbb{R}^{m\times m}$ 转化给定矩阵 $A\in\mathbb{R}^{m\times n}$ 的过程中，
主要的计算量不是确定 v 和 β，而是计算矩阵乘积.
在实际计算时，H 无需显式给出，而是根据如下的公式计算矩阵乘积 HA:

$$HA=(I-\beta vv^{\mathrm{T}})A=A-(\beta v)(v^{\mathrm{T}}A)$$

- 确定 v 和 β，并计算 βv
- 计算 $v^{\mathrm{T}}A$ (浮点运算量为 $O(2mn)$)
- 计算 $(\beta v)(v^{\mathrm{T}}A)$ (浮点运算量为 $O(mn)$) (存疑)
- 计算 $A-vu^{\mathrm{T}}$ (即为所求的乘积 HA) (浮点运算量为 $O(mn)$)

总浮点运算量为 $O(4mn)$

2.2.2 Givens 变换

若要将一个向量中许多相邻的分量置为零，则可以使用 Householder 变换.
若只要将其中一个分量置为零，则应使用 Givens 变换.

(Givens 变换)

$$G(i,k,\theta)=I+\sin(\theta)(e_ie_k^{\mathrm{T}}-e_ke_i^{\mathrm{T}})+(\cos(\theta)-1)(e_ie_i^{\mathrm{T}}+e_ke_k^{\mathrm{T}})=\begin{bmatrix}1&&&&&&\\&\ddots&&&&&\\&&\cos(\theta)&\cdots&\sin(\theta)&&\\&&\vdots&\ddots&\vdots&&\\&&-\sin(\theta)&\cdots&\cos(\theta)&&\\&&&&&\ddots&\\&&&&&&1\end{bmatrix}\begin{matrix}\\\\i\\\\k\\\\\\\end{matrix}$$

$$\begin{matrix}&&i&&k&&\end{matrix}$$

易证 $G(i,k,\theta)$ 总是一个正交阵.
从几何上看，$G(i,k,\theta)x$ 是在 (i,k) 坐标平面 $\mathrm{span}\{e_i,e_k\}$ 内将 x 按顺时针方向做了 θ 的旋转，
因此 Givens 变换又称为**平面旋转变换**.

设 $x\in\mathbb{R}^n$，记 $\begin{cases}y=G(i,k,\theta)x\\c=\cos(\theta)\\s=\sin(\theta)\end{cases}$，则有 $\begin{cases}y_i=cx_i+sx_k\\y_k=-sx_i+cx_k\\y_j=x_j\end{cases}\quad(\forall\, j\neq i,k)$

若要令 $y_k=0$，则只需取 $\begin{cases}c=\cos(\theta)=\frac{x_i}{\sqrt{x_i^2+x_k^2}}\\s=\sin(\theta)=\frac{x_k}{\sqrt{x_i^2+x_k^2}}\end{cases}$ 即有 $\begin{cases}y_i=\sqrt{x_i^2+x_k^2}\\y_k=0\end{cases}$

若用 $G(i,k,\theta)$ 变换左乘 (右乘) 矩阵 $A\in\mathbb{R}^{n\times n}$，
则它只改变 A 的第 i,k 行 (列) 的元素，其余元素保持不变.

直接使用 $\begin{cases}c=\cos(\theta)=\frac{x_i}{\sqrt{x_i^2+x_k^2}}\\s=\sin(\theta)=\frac{x_k}{\sqrt{x_i^2+x_k^2}}\end{cases}$ 计算 c 和 s，可能会发生下溢.

为简化记号，考虑 $\begin{cases}\begin{bmatrix}c&s\\-s&c\end{bmatrix}\begin{bmatrix}a\\b\end{bmatrix}=\begin{bmatrix}r\\0\end{bmatrix}\\r=\sqrt{a^2+b^2}\end{cases}$

为避免下溢，我们不直接使用 $\begin{cases}c=\frac{a}{r}\\s=\frac{b}{r}\end{cases}$，而是使用以下算法:

(计算 Givens 变换, 数值线性代数, 算法 3.2.2)

$$\begin{aligned}&\text{function: }[c,s]=\mathrm{Givens}(a,b)\\&\quad\text{if } b=0\\&\qquad c=1;\ s=0\\&\quad\text{else}\\&\qquad\text{if } |b|>|a|\\&\qquad\qquad t=\frac{a}{b};\ \ s=\frac{1}{\sqrt{1+t^2}};\ \ c=st\\&\qquad\text{else}\\&\qquad\qquad t=\frac{b}{a};\ \ c=\frac{1}{\sqrt{1+t^2}};\ \ s=ct\\&\qquad\text{end}\\&\quad\text{end}\\&\text{end}\end{aligned}$$

上述算法是数值稳定的.
假定计算结果是 $\tilde{c}$ 和 $\tilde{s}$，我们可以证明 $\begin{cases}\tilde{c}=c(1+\delta_c)&(\delta_c=O(\mathrm{eps}))\\\tilde{s}=s(1+\delta_s)&(\delta_s=O(\mathrm{eps}))\end{cases}$

若 a,b 为复数，记 $\begin{cases}a=a_0e^{i\theta_1}\\b=b_0e^{i\theta_2}\end{cases}$ 则我们可以利用一个对角的酉变换将 a,b 转为实数 a_0,b_0:

$$\begin{bmatrix}e^{-i\theta_1}&\\&e^{-i\theta_2}\end{bmatrix}\begin{bmatrix}a\\b\end{bmatrix}=\begin{bmatrix}e^{-i\theta_1}\cdot a_0e^{i\theta_1}\\e^{-i\theta_2}\cdot b_0e^{i\theta_2}\end{bmatrix}=\begin{bmatrix}a_0\\b_0\end{bmatrix}$$

然后对实向量 $\begin{bmatrix} a_0 \\ b_0 \end{bmatrix}$ 进行 Givens 变换:

$$
\begin{gathered}
[c, s] = \mathrm{Givens}(a_0, b_0) \\
\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} a_0 \\ b_0 \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}
\end{gathered}
$$

最终得到复向量 $\begin{bmatrix} a \\ b \end{bmatrix}$ 的 Givens 变换:

$$
\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} e^{-i\theta_1} & \\ & e^{-i\theta_2} \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}
$$

实际上复数的 Givens 变换可以有更快的实现 **(正确性存疑)**
参考 Iterative Methods for Sparse Linear Systems (Y. Saad) $6.5.9$ 节:
(假设 a, b, c, s 均为复数)

$$
\frac{\begin{cases} \begin{bmatrix} \bar{c} & \bar{s} \\ -s & c \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix} \\ r = \sqrt{|a|^2 + |b|^2} \end{cases} \quad \Rightarrow \quad \begin{cases} c = \frac{a}{r} \\ s = \frac{b}{r} \end{cases}}{}
$$

$$
\begin{aligned}
&\text{function: } [c, s] = \mathrm{Givens}(a, b) \\
&\qquad \text{if } b = 0 \\
&\qquad\qquad c = 1;\ s = 0 \\
&\qquad \text{else} \\
&\qquad\qquad \text{if } |b| > |a| \\
&\qquad\qquad\qquad t = \frac{a}{b};\ \ s = \frac{1}{\sqrt{1 + |t|^2}} \cdot \frac{b}{|b|};\ \ c = st \\
&\qquad\qquad \text{else} \\
&\qquad\qquad\qquad t = \frac{b}{a};\ \ c = \frac{1}{\sqrt{1 + |t|^2}} \cdot \frac{a}{|a|};\ \ s = ct \\
&\qquad\qquad \text{end} \\
&\qquad \text{end} \\
&\text{end}
\end{aligned}
$$

或者令 c 为实数，s 为复数:

$$
\begin{cases} \begin{bmatrix} c & \bar{s} \\ -s & c \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix} \\ r = \sqrt{|a|^2 + |b|^2} \end{cases} \quad \Rightarrow \quad \begin{cases} c = \frac{|a|}{r} \\ s = \frac{b}{r} \cdot \frac{|a|}{a} \end{cases}
$$

2.3 正交变换法

谱范数具有酉不变性，即对于任意酉矩阵 $U \in \mathbb{C}^n$ (满足 $U^{\mathrm{H}}U = UU^{\mathrm{H}} = I$) 都有 $\|UA\|_2 = \|A\|_2$

给定 $\begin{cases} A \in \mathbb{R}^{m \times n} \\ b \in \mathbb{R}^m \end{cases}$ 考虑线性最小二乘问题 $\min\limits_{x \in \mathbb{R}^n} \|Ax - b\|_2^2$
对于任意正交矩阵 $Q \in \mathbb{R}^n$ (满足 $Q^{\mathrm{T}}Q = QQ^{\mathrm{T}} = I$),
最小二乘问题 $\min\limits_{x \in \mathbb{R}^n} \|Q^{\mathrm{T}}(Ax - b)\|_2^2$ 都等价于原问题 $\min\limits_{x \in \mathbb{R}^n} \|Ax - b\|_2^2$
我们可以通过适当选取正交矩阵 Q 使得 $\min\limits_{x \in \mathbb{R}^n} \|Q^{\mathrm{T}}(Ax - b)\|_2^2$ 是较容易求解的形式.

这便是**正交变换法**的基本思想.

2.3.1 QR 分解

(QR 分解定理, 数值线性代数, 定理 $3.3.1$)

若 $A \in \mathbb{R}^{m \times n}$ $(m \geq n)$，则 A 具有 QR 分解 $A = Q \begin{bmatrix} R \\ 0_{(m-n) \times n} \end{bmatrix}$

其中 $Q \in \mathbb{R}^{m \times m}$ 是正交矩阵，$R \in \mathbb{R}^{n \times n}$ 是具有非负对角元的上三角阵.
特殊地，当 $m = n$ 且 A 非奇异时，上述分解是唯一的.

- **存在性证明:**
  对 n 使用数学归纳法.
  当 $n=1$ 时，命题自然成立.
  现假设命题对所有 $p\times(n-1)$ 矩阵成立 (其中 $p\geq n-1$)

  设 A 的第一列为 a_1，根据 Householder 变换的性质可知，
  存在正交矩阵 $Q_1\in\mathbb{R}^{m\times m}$ 使得 $Q_1^{\mathrm{T}}a_1=\|a_1\|_2\cdot e_1$
  我们记 $Q_1^{\mathrm{T}}A=\begin{bmatrix}\|a_1\|_2 & v^{\mathrm{T}}\\ 0_{m-1} & A_1\end{bmatrix}$

  对 $A_1\in\mathbb{R}^{(m-1)\times(n-1)}$ 应用归纳假设可知 A_1 具有 QR 分解 $A_1=Q_2\begin{bmatrix}R_2\\ 0_{(m-n)\times(n-1)}\end{bmatrix}$
  其中 $Q_2\in\mathbb{R}^{(m-1)\times(m-1)}$ 是正交矩阵，$R_2\in\mathbb{R}^{(n-1)\times(n-1)}$ 是具有非负对角元的上三角阵.
  则我们有:

  $$Q=Q_1\begin{bmatrix}1 & \\ & Q_2\end{bmatrix}\qquad R=\begin{bmatrix}\|a_1\|_2 & v^{\mathrm{T}}\\ 0_{n-1} & R_2\\ 0_{m-n} & 0_{(m-n)\times(n-1)}\end{bmatrix}\qquad A=Q\begin{bmatrix}R\\ 0_{(m-n)\times n}\end{bmatrix}$$

  因此 Q,R 满足命题的要求.
  根据数学归纳原理，存在性得证.

- **唯一性证明:**
  当 $m=n$ 且 A 非奇异时，假设 A 具有 QR 分解 $A=Q_1R_1=Q_2R_2$
  其中 $Q_1,Q_2\in\mathbb{R}^{m\times m}$ 是正交矩阵，$R_1,R_2\in\mathbb{R}^{n\times n}$ 是具有非负对角元的上三角阵.

  由于 A 非奇异 (即满秩)，故 R_1,R_2 的对角元均为正数 (即满秩)，因而可逆.
  我们有 $Q_2^{\mathrm{T}}Q_1=R_2R_1^{-1}$
  这表明正交阵 $Q_2^{\mathrm{T}}Q_1$ 等于一个对角元均为正数的上三角阵 $R_2R_1^{-1}$，
  因而只能是单位阵，即 $Q_2^{\mathrm{T}}Q_1=I$
  从而有 $\begin{cases}Q_1=Q_2\\ R_1=R_2\end{cases}$，即 A 具有唯一的 QR 分解.

利用 QR 分解，我们就可以实现**正交变换法**.

给定 $\begin{cases}A\in\mathbb{R}^{m\times n}\\ b\in\mathbb{R}^m\end{cases}$ 考虑线性最小二乘问题 $\min\limits_{x\in\mathbb{R}^n}\|Ax-b\|_2^2$ (假设 $\mathrm{rank}(A)=n\leq m$)
设 A 的 QR 分解是 $A=Q\begin{bmatrix}R\\ 0_{(m-n)\times n}\end{bmatrix}$，并分块为:

$$Q=[Q_1\quad Q_2]\qquad c=\begin{bmatrix}c_1\\ c_2\end{bmatrix}=\begin{bmatrix}Q_1^{\mathrm{T}}b\\ Q_2^{\mathrm{T}}b\end{bmatrix}=Q^{\mathrm{T}}b$$

则我们有:

$$\begin{aligned}\|Ax-b\|_2^2&=\|Q^{\mathrm{T}}(Ax-b)\|_2^2\\ &=\|Q^{\mathrm{T}}Ax-Q^{\mathrm{T}}b\|_2^2\\ &=\left\|\begin{bmatrix}R\\ 0_{m-n}\end{bmatrix}x-\begin{bmatrix}c_1\\ c_2\end{bmatrix}\right\|_2^2\\ &=\left\|\begin{bmatrix}Rx-c_1\\ c_2\end{bmatrix}\right\|_2^2\\ &=\|Rx-c_1\|_2^2+\|c_2\|_2^2\end{aligned}$$

因此 x 是 $\min\limits_{x\in\mathbb{R}^n}\|Ax-b\|_2^2=\min\limits_{x\in\mathbb{R}^n}\|Q^{\mathrm{T}}(Ax-b)\|_2^2$ 当且仅当 x 是 $Rx=c_1$ 的解.
综上所述，正交变换法的基本步骤为:

- 计算 A 的 QR 分解 $A=Q\begin{bmatrix}R\\ 0_{(m-n)\times n}\end{bmatrix}$
- 计算 $c_1=Q_1^{\mathrm{T}}b$
- 使用回代法求解上三角方程组 $Rx=c_1$

由此可知，实现正交变换法的关键是实现矩阵 A 的 QR 分解.

2.3.2 Householder 方法

下面我们使用 **Householder 方法**计算矩阵 A 的 QR 分解.
它与 Gauss 消去法计算 LU 分解 (利用 Gauss 变换逐步将 A 转换为上三角阵 U) 类似，
就是利用 Householder 变换逐步将 A 转换为上三角阵 R (具体来说是 $\begin{bmatrix} R \\ 0_{(m-n)\times n} \end{bmatrix}$)
以 $\begin{cases} m=6 \\ n=5 \end{cases}$ 的情况为例:

$$A = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix}$$

$$H_1A = \begin{bmatrix} \times & \times & \times & \times & \times \\ & + & \times & \times & \times \\ & + & \times & \times & \times \\ & + & \times & \times & \times \\ & + & \times & \times & \times \\ & + & \times & \times & \times \end{bmatrix}$$

$$H_2H_1A = \begin{bmatrix} \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & + & \times & \times \\ & & + & \times & \times \\ & & + & \times & \times \\ & & + & \times & \times \end{bmatrix}$$

$$H_3H_2H_1A = \begin{bmatrix} \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & + & \times \\ & & & + & \times \\ & & & + & \times \end{bmatrix}$$

$$H_4H_3H_2H_1A = \begin{bmatrix} \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \\ & & & & + \\ & & & & + \end{bmatrix}$$

$$H_5H_4H_3H_2H_1A = \begin{bmatrix} \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \\ & & & & \times \\ & & & & 0 \end{bmatrix}$$

给定矩阵 $A \in \mathbb{R}^{m\times n}\ (m \geq n)$
假定我们已进行了 $k-1$ 步，得到 Householder 变换 $H_1, \ldots, H_{k-1}$，使得:

$$A_k = H_{k-1}\cdots H_1A = \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ & A_{22}^{(k)} \end{bmatrix}$$

其中 $A_{11}^{(k)} \in \mathbb{R}^{(k-1)\times(k-1)}$ 是具有非负对角元的上三角阵，记 $A_{22}^{(k)} = [u_k, \ldots, u_n] \in \mathbb{R}^{(m-k+1)\times(n-k+1)}$

第 k 步是:

- 确定 Householder 变换 $\begin{cases} r_{kk} = \|u_k\|_2 \\ v_k = u_k - r_{kk} e_1 \\ \beta_k = \frac{2}{v_k^{\mathrm{T}} v_k} \\ \tilde{H}_k = I_{m-k+1} - \beta_k v_k v_k^{\mathrm{T}} \in \mathbb{R}^{(m-k+1)\times(m-k+1)} \end{cases}$ 使得 $\tilde{H}_k u_k = r_{kk} e_1$

  在实际计算中，我们还会将 v_k 的第一个分量调整为 1 (以方便存储)，并相应地调整 β_k
- 计算 $\tilde{H}_k A_{22}^{(k)} = (I_{m-k+1} - \beta_k v_k v_k^{\mathrm{T}}) A_{22}^{(k)} = A_{22}^{(k)} - v_k \cdot (\beta_k (A_{22}^{(k)})^{\mathrm{T}} v_k)^{\mathrm{T}} = \begin{bmatrix} r_{kk} & w_k^{\mathrm{T}} \\ & A_{22}^{(k+1)} \end{bmatrix}$

  令 $H_k = \begin{bmatrix} I_{k-1} & \\ & \tilde{H}_k \end{bmatrix}$ 并记 $A_{12}^{(k)} = \begin{bmatrix} z_k & \tilde{A}_{12}^{(k)} \end{bmatrix}$ 则我们有:

$$
\begin{aligned}
A_{k+1} &= H_k A_k \\
&= \begin{bmatrix} I_{k-1} & \\ & \tilde{H}_k \end{bmatrix} \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ & A_{22}^{(k)} \end{bmatrix} \\
&= \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ & \tilde{H}_k A_{22}^{(k)} \end{bmatrix} \\
&= \left[\begin{array}{c|cc} A_{11}^{(k)} & z_k & \tilde{A}_{12}^{(k)} \\ \hline & r_{kk} & w_k^{\mathrm{T}} \\ & & A_{22}^{(k+1)} \end{array}\right] \\
&= \left[\begin{array}{cc|c} A_{11}^{(k)} & z_k & \tilde{A}_{12}^{(k)} \\ & r_{kk} & w_k^{\mathrm{T}} \\ \hline & & A_{22}^{(k+1)} \end{array}\right] \\
&= \begin{bmatrix} A_{11}^{(k+1)} & A_{12}^{(k+1)} \\ & A_{22}^{(k+1)} \end{bmatrix}
\end{aligned}
$$

  其中我们记 $\begin{cases} A_{11}^{(k+1)} = \begin{bmatrix} A_{11}^{(k)} & z_k \\ & r_{kk} \end{bmatrix} \in \mathbb{R}^{k\times k} \\ A_{12}^{(k+1)} = \begin{bmatrix} \tilde{A}_{12}^{(k)} \\ w_k^{\mathrm{T}} \end{bmatrix} \in \mathbb{R}^{k\times(n-k)} \end{cases}$

这样，从 $k = 1$ 开始，对 A 依次进行 n 次 Householder 变换，
我们就可将 A 转化为上三角阵 $A^{(n)} = \begin{bmatrix} A_{11}^{(n)} \\ 0_{(m-n)\times n} \end{bmatrix} = H_n \cdots H_1 A.$
现在记 $\begin{cases} R = A_{11}^{(n)} \\ Q = H_1 \cdots H_n \end{cases}$ 即有 $A = Q \begin{bmatrix} R \\ 0_{(m-n)\times n} \end{bmatrix}$ (显然 R 的对角元均非负)

下面考虑使用 Householder 方法计算 A 的 QR 分解的存储问题.
当分解完成后，A 通常不再被需要，可用它来存储 Q 和 R.

此外，我们通常不用将 Q 显式存储，
而只需存放构成它的 n 个 m 阶 Householder 变换 H_k $(k = 1, \ldots, n)$ 的 v_k 和 β_k 即可.

在实际计算中，我们通常将 v_k 的第 k 个分量调整为 1 (以方便存储)，并相应地调整 β_k
于是 v_k 形如 $v_k = (0, \ldots, 0, 1, v_{k+1}^{(k)}, \ldots, v_n^{(k)})^{\mathrm{T}}$
因此我们可将 v_k 的第 $k+1$ 个分量至第 m 个分量存储在 A 的第 k 个对角元以下的位置.

以 $\begin{cases} m = 4 \\ n = 3 \end{cases}$ 的情况为例:

$$H_3H_2H_1A=\begin{bmatrix}R\\0_{1\times 3}\end{bmatrix}=\left[\begin{array}{ccc}r_{11}&r_{12}&r_{13}\\&r_{22}&r_{23}\\&&r_{33}\\\hline 0&0&0\end{array}\right]$$

$$H_1=I_4-\beta_1v_1v_1^{\mathrm{T}}\qquad v_1=(1,v_2^{(1)},v_3^{(1)},v_4^{(1)})^{\mathrm{T}}$$

$$H_2=I_4-\beta_2v_2v_2^{\mathrm{T}}\qquad v_2=(0,1,v_3^{(2)},v_4^{(2)})^{\mathrm{T}}$$

$$H_3=I_4-\beta_3v_3v_3^{\mathrm{T}}\qquad v_3=(0,0,1,v_4^{(3)})^{\mathrm{T}}$$

$$\text{Storage:}\begin{cases}d:=(\beta_1,\beta_2,\beta_3)\\A:=\begin{bmatrix}r_{11}&r_{12}&r_{13}\\v_2^{(1)}&r_{22}&r_{23}\\v_3^{(1)}&v_3^{(2)}&r_{33}\\v_4^{(1)}&v_4^{(2)}&v_4^{(3)}\end{bmatrix}\end{cases}$$

综合上述讨论，可得如下算法:
(Householder 方法计算 QR 分解, 数值线性代数, 算法 $3.3.1$)

$\text{function: } [Q,R]=\text{Householder\_QR}(A)$
    $[m,n]=\text{size}(A)$
    $Q=I_m$
    $\text{for } k=1:\min(m-1,n)$
        $[v,\beta]=\text{Householder}(A(k:m,k))$
        $A(k:m,k:n)=(I_{m-k+1}-\beta vv^{\mathrm{T}})A(k:m,k:n)=A(k:m,k:n)-(\beta v)(v^{\mathrm{T}}A(k:m,k:n))$
        $Q(1:m,k:m)=Q(1:m,k:m)(I_{m-k+1}-\beta vv^{\mathrm{T}})=Q(1:m,k:m)-(Q(1:m,k:m)v)(\beta v)^{\mathrm{T}}$
    end
end

该算法是数值稳定的.
如果不计算正交矩阵 Q 的话，总浮点运算量为 $2n^2m-\frac{1}{3}n^3$.
考虑求解最小二乘问题 $\min_{x\in\mathbb{R}^n}\|Ax-b\|_2^2$ (假设 $\mathrm{rank}(A)=n\leq m$),
与正则化方法相比，利用 Householder 方法计算 QR 分解的正交变换法代价更大，但得到的计算解更加精确.

上述算法拓展到复数域上的结果参见 Homework 4 的 Problem 3 (2)
上述算法的分块形式参见 Homework 4 的 Problem 6
上述算法可以改进以规避 Q 的生成 (如果 m 很大的话 $Q\in\mathbb{C}^{n\times n}$ 可能存储不下)，参考 Homework 4 的 Problem 7

Householder 方法并不是实现 QR 分解的唯一方法.
我们也可以利用 Givens 变换或 Gram-Schmidt 正交化来实现.

- 一般来说，利用 Givens 变换来实现 A 的 QR 分解所需的运算量大约是 Householder 方法的二倍.
  但如果 A 有较多的零元素 (即较为稀疏)，则灵活地使用 Givens 变换往往会使运算量大为减少.
  例如 Homework 4 的 Problem 4 和 Problem 8
- Cholesky 实现 A 的精简 QR 分解:

$$L=\text{Cholesky}(A^{\mathrm{H}}A)$$
$$R=L^{\mathrm{H}}$$
$$\text{solve } QR=A \text{ by forward sweep}$$

  其本质是直接求解法方程 $A^{\mathrm{H}}Ax=A^{\mathrm{H}}b$
  这种方法速度较快，但不太稳定.
  长方矩阵的条件数定义为:

$$\kappa_2(A)=\|A\|_2\|A^{\dagger}\|_2$$

$$A=U\begin{bmatrix}\Sigma_r&0_{r\times(n-r)}\\0_{(m-r)\times r}&0_{(m-r)\times(n-r)}\end{bmatrix}V^{\mathrm{H}}$$

$$A^{\dagger}=V\begin{bmatrix}\Sigma_r^{-1}&0_{r\times(m-r)}\\0_{(n-r)\times r}&0_{(n-r)\times(m-r)}\end{bmatrix}U^{\mathrm{H}}$$

则我们有 $\kappa_2(A^{\mathrm{H}}A) = (\kappa_2(A))^2$
因此 Cholesky QR 会加剧问题的病态程度.
不过现代算法为这种算法提供了一些补丁，使得它在高效的同时，精度得到补救.
(做两遍，第一遍加位移，尽管不准但已经良态很多了，第二遍就能算准了)

复数域上 Cholesky QR 算法的实现参见 Homework 4 的 Problem 3 (1)

- Gram-Schmidt 正交化来实现 A 的 QR 分解数值不稳定.
  在后面我们将对 Gram-Schmidt 正交化进行改进，以提高数值稳定性.

此外，QR 分解不仅可用来求解最小二乘问题，
它也是数值线性代数诸多重要算法的基础,，例如的求解特征值问题的 QR 方法.

我们亦可利用 QR 分解求解线性方程组 $Ax = b$ (其中 $A \in \mathbb{R}^{n\times n}$):

- 计算 A 的 QR 分解 $A = QR$，于是 $Ax = b \Leftrightarrow QRx = b \Leftrightarrow Rx = Q^{\mathrm{T}}b$
- 使用回代法求解上三角方程组 $Rx = Q^{\mathrm{T}}b$ 得到 x

对于某些病态的线性方程组，使用 QR 分解的计算结果往往比 LU 分解的要好得多 (当然计算量也大得多)

2.3.3 Gram-Schmidt 方法

(1) CGS & MGS

投影到一组标准正交的向量张成的空间上 $\mathrm{span}\{q_1, \dots, q_n\}$ 的 (正交) 投影算子 $P := \sum_{i=1}^{n} q_i q_i^{\mathrm{H}}$

考虑 $A \in \mathbb{C}^{n\times n}$ 的 QR 分解:

$$A = [a_1, a_2, \dots, a_n] = [q_1, q_2, \dots, q_n] \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ & r_{22} & \cdots & r_{2n} \\ & & \ddots & \vdots \\ & & & r_{nn} \end{bmatrix} = QR$$

我们有:

$$a_1 = q_1 r_{11}$$
$$\Rightarrow$$
$$\begin{cases} r_{11} = \|a_1\|_2 \\ q_1 = \frac{a_1}{\|a_1\|_2} = \frac{a_1}{r_{11}} \end{cases}$$
$$a_k = \sum_{i=1}^{k} q_i r_{ik} = \sum_{i=1}^{k} q_i \langle q_i, a_k\rangle = q_k r_{kk} + \sum_{i=1}^{k-1} q_i \langle q_i, a_k\rangle \ (k = 2, \dots, n)$$
$$\Rightarrow$$
$$\text{for } k = 2, \dots, n \begin{cases} r_{ik} = \langle q_i, a_k\rangle \ (i = 1, \dots, k-1) \\ r_{kk} = \|a_k - \sum_{i=1}^{k-1} q_i r_{ik}\|_2 \\ q_k = \frac{1}{r_{kk}}(a_k - \sum_{i=1}^{k-1} q_i r_{ik}) \end{cases}$$

这就是**经典 Gram-Schmidt 正交化 (CGS)** 算法的数学原理.
其第 i 步作用的投影算子为:

$$P_i^{\perp} := I_n - Q_{i-1}Q_{i-1}^{\mathrm{H}} = I_n - (q_1 q_1^{\mathrm{H}} + \cdots + q_{i-1} q_{i-1}^{\mathrm{H}})$$

注意到 CGS 中，我们是把 $a_k - \sum_{i=1}^{k-1} q_i r_{ik}$ 放在一起计算的，计算 $r_{ik} = \langle q_i, a_k\rangle$ 时一直用的是原始向量 a_k
理论上我们可以将其拆分，使得每步计算 r_{ik} 都能用到新的计算结果:

$$
\begin{gathered}
a_1 = q_1 r_{11} \\
\Rightarrow \\
\begin{cases} r_{11} = \|a_1\|_2 \\ q_1 = \frac{a_1}{\|a_1\|_2} = \frac{a_1}{r_{11}} \end{cases} \\
a_k = \sum_{i=1}^{k} q_i r_{ik} = \sum_{i=1}^{k} q_i \langle q_i, a_k \rangle = q_k r_{kk} + \sum_{i=1}^{k-1} q_i \langle q_i, a_k \rangle \ (k = 2, \ldots, n) \\
\Rightarrow \\
\text{for } k = 2, \ldots, n \begin{cases} r_{1k} = \langle q_1, a_k \rangle \\ r_{ik} = \langle q_i, a_k - \sum_{p=1}^{i-1} q_p r_{p_k} \rangle \ (i = 2, \ldots, k-1) \\ r_{kk} = \| a_k - \sum_{p=1}^{k-1} q_p r_{pk} \|_2 \\ q_k = \frac{1}{r_{kk}} (a_k - \sum_{p=1}^{k-1} q_p r_{pk}) \end{cases}
\end{gathered}
$$

这就是**修正的 Gram-Schmidt 正交化 (MGS)** 算法的数学原理.
其第 i 步作用的投影算子为:

$$
P_i^{\perp} := (I_n - q_{i-1} q_{i-1}^{\mathrm{H}}) \cdots (I_n - q_1 q_1^{\mathrm{H}})
$$

在数学上 MGS 和 CGS 是等价的，但在数值计算中 MGS 会比 CGS 速度更慢而数值稳定性更高.
这是因为计算中产生的误差项也参与了正交化，因此不会被传递下去.
正交性损失会从 $O((\kappa_2(A))^2)$ 级别降到 $O(\kappa_2(A))$ 级别，但依然不够稳定.
正交性损失可能会造成精度瓶颈，即一个理论上收敛的问题迭代到一定精度会出现振荡:
正交性损失迫使其偏离理论结果，而理论结果又迫使其收敛，从而在一定精度范围内振荡.

(Applied Numerical Linear Algebra 算法 3.1)
对于任意给定的列满秩矩阵 $A = [a_1, \ldots, a_n] \in \mathbb{C}^{m \times n}$ (其中 $m \geq n$)
根据**经典 Gram-Schmidt 正交化 (CGS)** 和**修正的 Gram-Schmidt 正交化 (MGS)** 可得到 A 的精简 QR 分解:

$$
\begin{array}{l}
\text{function: } [Q, R] = \mathrm{Gram\_Schmidt\_QR}(A, \text{modified}, \text{tolerance}) \\
\quad [m, n] = \mathrm{size}(A) \\
\quad r = n \quad (r \text{ stands for rank of matrix } A, \text{ whose initial value is } n) \\
\quad Q = \mathrm{zeros}(m, m) \\
\quad R = \mathrm{zeros}(m, n) \\
\hline
\quad \text{for } k = 1 : n \quad (\text{Calculate the } k\text{-th column of } Q \text{ and } R) \\
\qquad Q(1:m, k) = A(1:m, k) \\
\qquad \text{for } i = 1 : k-1 \quad (\text{subtract component in } q_i \text{ from } a_k \text{ in two ways}) \\
\qquad\quad R(i,k) = \begin{cases} Q(1:m,i)^{\mathrm{H}} A(1:m,k) & \text{if modified } = \mathrm{FALSE\ (CGS)} \\ Q(1:m,i)^{\mathrm{H}} Q(1:m,k) & \text{if modified } = \mathrm{TRUE\ (MGS)} \end{cases} \\
\qquad\quad Q(1:m,k) = Q(1:m,k) - R(i,k) Q(1:m,i) \\
\qquad \text{end} \\
\qquad R(k,k) = \|Q(1:m,k)\|_2 \\
\qquad \text{if } R(k,k) < \text{tolerance} \quad (a_k \text{ is linearly dependent on } a_1, \ldots, a_{k-1}) \\
\qquad\quad r = k-1 \quad (\text{rank of matrix } A) \\
\qquad\quad \text{break} \\
\qquad \text{end} \\
\qquad Q(1:m,k) = \dfrac{1}{R(k,k)} Q(1:m,k) \\
\quad \text{end} \\
\hline
\quad Q = Q(1:m, 1:r) \\
\quad R = R(1:r, 1:n) \\
\text{end}
\end{array}
$$

注意其中对于 $R(i,i)$ 是否等于零的检查只是为了确保程序在 A 列不满秩时不会出现零除错误.
最终我们得到列满秩矩阵 $A \in \mathbb{R}^{m \times n}$ 的精简 QR 分解 (其中 $m \geq n$)
其中 $\mathrm{rank}(A) = n$，$Q \in \mathbb{R}^{m \times n}$ 列标准正交，$R \in \mathbb{R}^{n \times n}$ 为上三角阵，满足 $A = QR$
若将 Q 补全为正交阵 $\tilde{Q} = [Q, Q_{\perp}] \in \mathbb{R}^{m \times m}$，则我们有:

$$
A = [Q \quad Q_{\perp}] \begin{bmatrix} R \\ 0_{(m-n) \times n} \end{bmatrix}
$$

(2) 优化

不过上述算法依然是很粗糙的:

- ① 可能会出现把秩 r 算低了的错误 (这会导致算法提前终止)
- ② 某些操作可以向量化以优化运行速度
- ③ 没有考虑 $m < n$ 的情况

优化后的算法如下:

$$
\begin{array}{l}
\text{function: } [Q, R] = \text{Gram\_Schmidt\_QR}(A, \text{modified}, \text{tolerance}) \\
\quad [m, n] = \text{size}(A) \\
\quad r = 0 \quad (r \text{ stands for rank of matrix } A) \\
\quad Q = \text{zeros}(m, m) \\
\quad R = \text{zeros}(m, n) \\
\hline
\quad \text{for } k = 1 : \min(m, n) \\
\qquad Q(1:m, r+1) = A(1:m, k) \\
\qquad \text{if modified} == \text{TRUE (MGS: Modified Gram-Schmidt)} \\
\qquad\quad \text{for } i = 1 : r \\
\qquad\qquad R(i, k) = Q(1:m, i)^{\mathrm{H}} Q(1:m, r+1) \\
\qquad\qquad Q(1:m, r+1) = Q(1:m, r+1) - R(i, k) Q(1:m, i) \\
\qquad\quad \text{end} \\
\qquad \text{else (CGS: Classic Gram-Schmidt)} \\
\qquad\quad R(1:r, k) = Q(1:m, 1:r)^{\mathrm{H}} Q(1:m, r+1) \\
\qquad\quad Q(1:m, r+1) = Q(1:m, r+1) - Q(1:m, 1:r) R(1:r, k) \\
\qquad \text{end} \\
\qquad R(r+1, k) = \|Q(1:m, r+1)\|_2 \\
\qquad \text{if } R(r+1, k) < \text{tolerance} \quad (\text{indicates linear dependence}) \\
\qquad\quad R(r+1, k) = 0 \\
\qquad \text{else} \\
\qquad\quad Q(1:m, r+1) = \dfrac{1}{R(r+1, k)} Q(1:m, r+1) \\
\qquad\quad r = r + 1 \quad (\text{increment rank}) \\
\qquad \text{end} \\
\quad \text{end} \\
\hline
\quad \text{if } n > m \quad (\text{fill remaining R}) \\
\qquad \text{for } k = m+1 : n \\
\qquad\quad R(1:r, k) = Q(1:m, 1:r)^{\mathrm{H}} A(1:m, k) \\
\qquad \text{end} \\
\quad \text{end} \\
\hline
\quad Q = Q(1:m, 1:r) \\
\quad R = R(1:r, 1:n) \\
\text{end}
\end{array}
$$

其 Matlab 代码为:

```
function [Q, R] = Gram_Schmidt_QR(A, modified, tolerance)
    [m, n] = size(A);
    r = 0; % Initial rank of matrix A
    Q = zeros(m, m);
    R = zeros(m, n);

    for k = 1:min(m,n) % Calculate the i-th column of Q and R
        Q(1:m, r+1) = A(1:m, k);

        if modified
            for i = 1:r % Subtract component in q_j from a_i
                R(i, k) = Q(1:m, i)' * Q(1:m, r+1); % MGS
                Q(1:m, r+1) = Q(1:m, r+1) - R(i, k) * Q(1:m, i);
            end
        else
            R(1:r, k) = Q(1:m, 1:r)' * Q(1:m,r+1); % CGS
```

```
            Q(1:m, r+1) = Q(1:m, r+1) - Q(1:m, 1:r) * R(1:r, k);
        end

        R(r+1, k) = norm(Q(1:m, r+1), 2); % 2-norm

        % Check if the norm is smaller than the tolerance, indicating linear dependence
        if R(r+1, k) < tolerance
            R(r+1, k) = 0;  % Set R entry to zero if linearly dependent
        else
            Q(1:m, r+1) = Q(1:m, r+1) / R(r+1, k);  % Normalize the vector
            r = r + 1;  % Increment the rank
        end

    end

    % Additional step: if the number of columns n is greater than m,
    % compute the remaining upper triangular part of R using the orthonormal Q matrix
    if n > m
        for k = m+1:n
            % Compute the projections of columns of A onto the previously computed orthonormal
columns of Q
            R(1:r, k) = Q(1:m, 1:r)' * A(1:m, k);
        end
    end

    Q = Q(1:m, 1:r);
    R = R(1:r, 1:n);
end
```

(3) 重正交化

重正交化是正交化步骤走两遍，不是整个 Gram-Schmidt 走两遍.
(不过后者的效果也不差，第一遍的正交化结果虽然精度不高，但已经良态很多了)
无论 CGS 还是 MGS，重正交化的结果都是好的.
(如果矩阵过分病态的话，例如条件数为 10^{30} 的时候，我们可能需要多次重正交化，但一般来说一次重正交化就够了)

function: $[Q, R] = \text{Gram\_Schmidt\_QR}(A, \text{tolerance}, \text{modified}, \text{reorthogonalized})$

$[m, n] = \text{size}(A)$

$r = 0$ (r stands for the rank of matrix A)

$Q = \text{zeros}(m, m)$

$R = \text{zeros}(m, n)$

$\delta = \text{zeros}(m, 1)$

---

if reorthogonalized == TRUE

max_iter = 2

else

max_iter = 1

end

---

for $k = 1 : \min(m, n)$

$Q(1 : m, r + 1) = A(1 : m, k)$

if modified == TRUE (MGS: Modified Gram-schmidt)

for iter = 1:max_iter

for $i = 1 : r$

$\delta(i) = Q(1 : m, i)^{\mathrm{T}} Q(1 : m, r + 1)$

$R(i, k) = R(i, k) + \delta(i)$

$Q(1 : m, r + 1) = Q(1 : m, r + 1) - \delta(i) Q(1 : m, i)$

end

end

else (CGS: Classic Gram-Schmidt)

for iter = 1:max_iter

$\delta(1 : r) = Q(1 : m, 1 : r)^{\mathrm{T}} Q(1 : m, r + 1)$

$R(1 : r, k) = R(1 : r, k) + \delta(1 : r)$

$Q(1 : m, r + 1) = Q(1 : m, r + 1) - Q(1 : m, 1 : r) \delta(1 : r)$

end

end

$R(r + 1, k) = \|Q(1 : m, r + 1)\|_2$

if $R(r + 1, k) <$ tolerance (indicates linear dependence)

$R(r + 1, k) = 0$

else

$$Q(1 : m, r + 1) = \frac{1}{R(r + 1, k)} Q(1 : m, r + 1)$$

$r = r + 1$ (increment rank)

end

end

---

if $n > m$ (fill remaining R)

for $k = m + 1 : n$

$R(1 : r, k) = Q(1 : m, 1 : r)^{\mathrm{H}} A(1 : m, k)$

end

end

$Q = Q(1 : m, 1 : r)$

$R = R(1 : r, 1 : n)$

end

其 Matlab 代码为:

```
function [Q, R] = Gram_Schmidt_QR(A, tolerance, modified, reorthogonalized)
    % This function performs the Gram-Schmidt QR factorization of a matrix A
    % It supports both classical and modified versions of the GS algorithm,
    % and it allows for reorthogonalization to improve numerical stability.
    %
    % Inputs:
    %   - A: The m x n matrix to be factorized
    %   - tolerance: The threshold below which a vector is considered linearly dependent
    %   - modified: Boolean flag to choose between Classical Gram-Schmidt (CGS)
    %               or Modified Gram-Schmidt (MGS)
    %   - reorthogonalized: Boolean flag to perform reorthogonalization (improves numerical
stability)
```

```
    %
    % Outputs:
    %   - Q: An m x r orthonormal matrix (r is the rank of A, or the number of orthogonal
vectors)
    %   - R: An r x n upper triangular matrix

    [m, n] = size(A);  % Get the size of matrix A (m rows, n columns)
    r = 0;  % Initialize rank of A
    Q = zeros(m, m);  % Preallocate Q as an m x m zero matrix
    R = zeros(m, n);  % Preallocate R as an m x n zero matrix
    delta = zeros(m, 1);  % Temporary vector for storing projection coefficients

    % Set the number of orthogonalization iterations based on the reorthogonalized flag
    if reorthogonalized
        max_iter = 2;  % If reorthogonalization is enabled, perform two passes
    else
        max_iter = 1;  % Otherwise, perform only one pass
    end

    % Main loop over each column of matrix A (for each column k)
    for k = 1:min(m,n)
        % Initialize the k-th column of Q as the k-th column of A
        Q(1:m, r+1) = A(1:m, k);

        % If modified Gram-Schmidt (MGS) is selected
        if modified
            for iter = 1:max_iter  % Repeat orthogonalization based on max_iter
                for i = 1:r  % Loop over previously computed columns of Q
                    delta(i) = Q(1:m, i)' * Q(1:m, r+1);  % Compute projection of Q_k on Q_i
                    R(i, k) = R(i, k) + delta(i);  % Update the corresponding entry in R
                    Q(1:m, r+1) = Q(1:m, r+1) - delta(i) * Q(1:m, i);  % Subtract projection
from Q_k
                end
            end
        else  % Classical Gram-Schmidt (CGS)
            for iter = 1:max_iter
                delta(1:r) = Q(1:m, 1:r)' * Q(1:m,r+1);  % Compute projections in one step
                R(1:r, k) = R(1:r, k) + delta(1:r);  % Update R
                Q(1:m, r+1) = Q(1:m, r+1) - Q(1:m, 1:r) * delta(1:r);  % Subtract the projection
from Q_k
            end
        end

        % Compute the 2-norm of the current column of Q (for normalization)
        R(r+1, k) = norm(Q(1:m, r+1), 2);

        % Check if the norm is smaller than the tolerance, indicating linear dependence
        if R(r+1, k) < tolerance
            R(r+1, k) = 0;  % Set R entry to zero if linearly dependent
        else
            Q(1:m, r+1) = Q(1:m, r+1) / R(r+1, k);  % Normalize the vector
            r = r + 1;  % Increment the rank
        end

    end

    % Additional step: if the number of columns n is greater than m,
    % compute the remaining upper triangular part of R using the orthonormal Q matrix
    if n > m
        for k = m+1:n
            % Compute the projections of columns of A onto the previously computed orthonormal
columns of Q
            R(1:r, k) = Q(1:m, 1:r)' * A(1:m, k);
        end
    end

    % Reduce the size of Q and R to the actual rank r of A
```

```
    Q = Q(1:m, 1:r);  % Return the first r columns of Q
    R = R(1:r, 1:n);  % Return the first r rows of R
end
```

2.3.4 亏秩最小二乘问题

(1) 列选主元的 QR 分解

亏秩最小二乘问题可通过列选主元的 QR 分解求解:

$$Q^{\mathrm{T}}AP = \begin{bmatrix} R_{11} & R_{12} \\ 0_{(m-r)\times r} & 0_{(m-r)\times(n-r)} \end{bmatrix}$$

其中 $A \in \mathbb{R}^{m\times n}$ 满足 $r = \mathrm{rank}(A) < n \leq m$
$Q \in \mathbb{R}^{m\times m}$ 为实正交阵，$P \in \mathbb{R}^{n\times n}$ 为置换矩阵，$R_{11} \in \mathbb{R}^{r\times r}$ 为上三角阵.
设我们已经计算了 Householder 矩阵 $H_1, \ldots, H_{k-1}$ 和置换矩阵 $P_1, \ldots, P_{k-1}$ 使得:

$$(H_{k-1}\cdots H_1)A(P_1\cdots P_{k-1}) = R^{(k-1)} = \begin{bmatrix} R_{11}^{(k-1)} & R_{12}^{(k-1)} \\ 0_{(m-k+1)\times(k-1)} & R_{22}^{(k-1)} \end{bmatrix}$$

其中 $R_{11}^{(k-1)} \in \mathbb{R}^{(k-1)\times(k-1)}$ 是非奇异上三角阵.
设 $R_{22}^{(k-1)}$ 的列向量为 $z_k^{(k-1)}, \ldots, z_n^{(k-1)} \in \mathbb{R}^{m-k+1}$
取 $p := \arg\max_{k\leq j\leq n} \|z_j^{(k-1)}\|_2$ (如果有多个最大值，则取其中最小的下标)

- 若 $\mathrm{rank}(A) = k-1$，则理论上 $\max_{k\leq j\leq n} \|z_j^{(k-1)}\|_2 = 0$，算法终止.
- 若 $\mathrm{rank}(A) > k-1$，则 $\max_{k\leq j\leq n} \|z_j^{(k-1)}\|_2 > 0$
  我们取 P_k 将 $R_{22}^{(k-1)}$ 的最大列 (第 p 列) 移到最前面 (第 k 列)
  然后计算 H_k 将 $R_{22}^{(k-1)}P_k$ 的第 k 列 (现在是最大列) 的非对角元消成零.

利用 $\|\cdot\|_2$ 的酉不变性，我们可以通过修正旧的列范数来得到新的列范数，而不必每一步都重新计算列范数:

$$\|z_j^{(k)}\|_2^2 = \|z_j^{(k-1)}\|_2^2 - r_{kj}^2 \quad (j = k+1, \ldots, n)$$

这可使列选主元的工作量由 $O(mn^2)$ 减少到 $O(mn)$
综上所述，我们得到如下算法:
(列选主元的 Householder QR 分解, Matrix Computation, 算法 $5.4.1$)

Given $A \in \mathbb{R}^{m\times n}$ $(m \geq n)$

---

for $j = 1 : n$
    $c(j) = \|A(:,j)\|_2^2$
end
$r = 0$
$p = \arg\max\limits_{1\leq j\leq n} c(j)$
while $c(p) > 0$
    $r = r + 1$
    $\text{pivot}(r) = p$
    $A(:,r) \leftrightarrow A(:,p)$
    $c(r) \leftrightarrow c(k)$
    $[v,\beta] = \text{Householder}(A(r:m,r))$
    $A(r:m,r:n) = (I_{m-r+1} - \beta vv^{\mathrm{T}})A(r:m,r:n) = A(r:m,r:n) - \beta v(v^{\mathrm{T}}A(r:m,r:n))$
    $A(r+1:m,r) = v(2:m-r+1)$
    for $j = r+1 : n$
        $c(j) = c(j) - A(r,j)^2$
    end
    if $r < n$
        $p = \arg\max\limits_{r+1\leq j\leq n} c(j)$
    else
        break
    end
end

上述算法得到了 $r = \text{rank}(A)$ 和列选主元的 QR 分解:

$$Q^{\mathrm{T}}AP = \begin{bmatrix} R_{11} & R_{12} \\ 0_{(m-r)\times r} & 0_{(m-r)\times(n-r)} \end{bmatrix}$$

其中 $Q = H_1 \cdots H_r$ 的第 j 个 Householder 向量的 $j+1:m$ 分量存储在 $A(j+1:m,j)$ 中.
置换矩阵 P 由整数向量 pivot 标记 (它可通过将单位矩阵 I_n 的第 j 行与第 $\text{pivot}(j)$ 行互换得到)
总计算量为 $4mnr - 2r^2(m+n) + \frac{4}{3}r^3$

(2) 完全正交分解

亏秩最小二乘问题的 RQ 解法，参见 Homework 06 Problem 05
我们也可以对列选主元的 QR 分解的结果右乘一系列 Householder 矩阵将其完全正交分解:

$$Q^{\mathrm{T}}AP = \begin{bmatrix} R_{11} & R_{12} \\ 0_{(m-r)\times r} & 0_{(m-r)\times(n-r)} \end{bmatrix}$$

$$\widetilde{H}_r \cdots \widetilde{H}_1 \begin{bmatrix} R_{11}^{\mathrm{T}} \\ R_{12}^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} T_{11}^{\mathrm{T}} \\ 0_{(n-r)\times r} \end{bmatrix}$$

$$\Rightarrow$$

$$Q^{\mathrm{T}}AU = T = \begin{bmatrix} T_{11} & 0_{r\times(n-r)} \\ 0_{(m-r)\times r} & 0_{(m-r)\times(n-r)} \end{bmatrix}$$

其中 $T_{11} \in \mathbb{R}^{r\times r}$ 为下三角阵，$U = P\widetilde{H}_1 \cdots \widetilde{H}_r \in \mathbb{R}^{n\times n}$ 为实正交阵.

2.4 补充

2.4.1 最小二乘问题的变体

最小二乘问题本质上是求解线性方程组的问题:

$$\min_x \|b-Ax\|_2^2$$

$$\Leftrightarrow$$

$$\text{Solve normal equation: } A^{\mathrm{H}}Ax=A^{\mathrm{H}}b$$

$$\Leftrightarrow$$

$$\begin{cases} r=b-Ax \\ A^{\mathrm{H}}r=0_n \end{cases}$$

$$\Leftrightarrow$$

$$\text{Solve augmented equation: } \begin{bmatrix} I_m & A \\ A^{\mathrm{H}} & 0_{n\times n} \end{bmatrix} \begin{bmatrix} r \\ x \end{bmatrix} = \begin{bmatrix} b \\ 0_n \end{bmatrix}$$

带 l_2 惩罚项的最小二乘问题本质上是标准的最小二乘问题:

$$\min_x \|b-Ax\|_2^2+\mu\|x\|_2^2$$

$$\Leftrightarrow$$

$$\min_x \left\| \begin{bmatrix} A \\ \sqrt{\mu}I_n \end{bmatrix} x - \begin{bmatrix} b \\ 0_n \end{bmatrix} \right\|_2^2$$

因此其法方程为:

$$\begin{bmatrix} A \\ \sqrt{\mu}I_n \end{bmatrix}^{\mathrm{H}} \begin{bmatrix} A \\ \sqrt{\mu}I_n \end{bmatrix} x = \begin{bmatrix} A \\ \sqrt{\mu}I_n \end{bmatrix}^{\mathrm{H}} \begin{bmatrix} b \\ 0_n \end{bmatrix}$$

$$\Leftrightarrow$$

$$(A^{\mathrm{H}}A+\mu I_n)x=A^{\mathrm{H}}b$$

一般我们取 $\mu>0$ 为一个较小的正实数，这样 $A^{\mathrm{H}}A$ 会变得更加良态
大特征值几乎不变，而小特征值在相对程度上增大很多倍.

线性等式约束的最小二乘问题:

$$\min_{Cx=d} \|Ax-b\|_2^2 \text{ where } \begin{cases} A\in\mathbb{C}^{m\times n} \text{ and } \operatorname{rank}(A)=n\le m \\ b\in\mathbb{C}^m \\ C\in\mathbb{C}^{p\times n} \text{ and } \operatorname{rank}(C)=p<n \\ d\in\mathbb{C}^p \end{cases}$$

Lagrange 乘子法自然是一般解法，可参考 Homework 6 Problem 2.
下面我们介绍**消元法**:
设 $C\in\mathbb{C}^{p\times n}$ 的 RQ 分解为:

$$\begin{bmatrix} R & 0_{p\times(n-p)} \end{bmatrix} \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} = C \text{ where } \begin{cases} R\in\mathbb{C}^{p\times p} \\ Q_1\in\mathbb{C}^{p\times n} \\ Q_2\in\mathbb{C}^{(n-p)\times n} \end{cases}$$

其中 Q_1 是行标准正交的.
我们记:

$$y:=Qx=\begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix}x=\begin{bmatrix} Q_1x \\ Q_2x \end{bmatrix}=\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

$$x=\begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix}^{\mathrm{H}}y=\begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix}^{\mathrm{H}}\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}=\begin{bmatrix} Q_1^{\mathrm{H}}y_1 \\ Q_2^{\mathrm{H}}y_2 \end{bmatrix}$$

$$\tilde{A}:=AQ^{\mathrm{H}}=A\begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix}^{\mathrm{H}}=\begin{bmatrix} AQ_1^{\mathrm{H}} & AQ_2^{\mathrm{H}} \end{bmatrix}=\begin{bmatrix} \tilde{A}_1 & \tilde{A}_2 \end{bmatrix}\in\mathbb{C}^{m\times n}$$

$$\tilde{b}:=b-\tilde{A}_1y_1$$

当 R 固定时，Q_1 也是固定的，而 Q_2 是自由变动的，(甚至) 不用保证 $Q=\begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix}$ 为酉矩阵.
此时 $\begin{cases} y_1:=Q_1x \\ \tilde{A}_1:=AQ_1^{\mathrm{H}} \\ \tilde{b}:=b-\tilde{A}_1y_1 \end{cases}$ 是固定的，而 $\begin{cases} y_2:=Q_2x \\ \tilde{A}_2:=AQ_2^{\mathrm{H}} \end{cases}$ 是依赖于 Q_2 的，可以自由变动.

则我们有:

$$
\begin{aligned}
\|b-Ax\|_2^2 &= \|b-AQ^{\mathrm{H}}y\|_2^2 \\
&= \|b-\tilde{A}y\|_2^2 \\
&= \left\| b-\begin{bmatrix}\tilde{A}_1 & \tilde{A}_2\end{bmatrix}\begin{bmatrix}y_1\\y_2\end{bmatrix}\right\|_2^2 \\
&= \|(b-\tilde{A}_1y_1)-\tilde{A}_2y_2\|_2^2 \\
&= \|\tilde{b}-\tilde{A}_2y_2\|
\end{aligned}
$$

$$
\min_x \|b-Ax\|_2^2 \;\Leftrightarrow\; \min_{y_2}\|\tilde{b}-\tilde{A}_2y_2\|
$$

2.4.2 GTH 算法

参见 Homework 06 Problem 07

2.4.3 Arnoldi 过程

给定 $A\in\mathbb{C}^{n\times n}$ 和 $b\in\mathbb{C}^n$，我们便可以计算序列:

$$
\begin{aligned}
y_1 &= b \\
y_2 &= Ay_1 = Ab \\
y_3 &= Ay_2 = A^2b \\
&\cdots \\
y_n &= Ay_{n-1} = A^{n-1}b
\end{aligned}
$$

我们只需计算至 $y_n=A^{n-1}b$ 即可
这是因为 Cayley-Hamilton 定理保证了 $A^k\ (k\geq n)$ 可以表示为 $I_n,A,\dots,A^{n-1}$ 的线性组合.

> **(Cayley-Hamilton 定理, Matrix Analysis 定理 2.4.3.2)**
> 设 $p_A(t):=\det(tI_n-A)$ 是 $A\in\mathbb{C}^{n\times n}$ 的特征多项式，则我们有 $p_A(A)=0_{n\times n}$ 成立.
> 换言之，任意复方阵都满足其特征方程.

我们记:

$$
\begin{aligned}
Y &:= [y_1,y_2,\dots,y_{n-1},y_n] \\
c &:= -Y^{-1}A^ny_1 \\
C &:= [e_2,e_3,\dots,e_n,-c] = \begin{bmatrix}
0 & & & & -c_1 \\
1 & 0 & & & -c_2 \\
 & 1 & \ddots & & \vdots \\
 & & \ddots & 0 & -c_{n-1} \\
 & & & 1 & -c_n
\end{bmatrix}
\end{aligned}
$$

其中我们暂且假定 $Y\in\mathbb{C}^{n\times n}$ 非奇异
则我们有:

$$
\begin{aligned}
AY &= [Ay_1,Ay_2,\dots,Ay_{n-1},Ay_n] \\
&= [y_2,y_3,\dots,y_n,A^ny_1] \\
&= [Ye_2,Ye_3,\dots,Ye_n,Y\cdot(-Y^{-1}A^ny_1)] \quad (\text{note that } c:=Y^{-1}A^ny_1) \\
&= Y[e_2,e_3,\dots,e_n,-c] \\
&= YC
\end{aligned}
$$

于是我们有 $Y^{-1}AY=C$
注意到 C 的特征多项式 $p_C(t)=t^n+\sum_{i=1}^n c_it^{i-1}$ 非常容易得到，原则上可通过求解 $p_C(t)$ 来得到 A 的特征值.
整个过程我们只需使用矩阵-向量乘法就能把 A 化为一个非常简单的形式.
此时求解线性方程组 $Ax=b$ 的问题便等价于求解 $C(Y^{-1}x)=Y^{-1}b$ (其中 $C=Y^{-1}AY$)
但上述形式在实际应用中并不有效:

- ① 即使 A 是稀疏矩阵，Y 也可能是稠密的，故没有利用期望变换后的问题比原问题求解起来更容易

- ② Y 可能是一个非常病态的矩阵，导致 $c = Y^{-1}(A^n b)$ 算不准.
  可以预见 (参见幂法)，序列 $\{y_i = A^{i-1}b\}$ 收敛到对应于 A 的模最大特征值的一个特征向量
  因而 Y 的列 $y_1, y_2, \dots, y_n$ 可能会趋于线性相关，信息丢失会非常严重.

为克服上述困难，我们使用酉矩阵 Q 代替 Y.
记 Y 的前 $k = 1, \dots, n$ 列张成的空间为 **Krylov 子空间** $\mathcal{K}(A, b, k) := \operatorname{span}\{b, Ab, \dots, A^{k-1}b\}$
我们要求 $Q \in \mathbb{C}^{n\times n}$ 的前 $k = 1, \dots, n$ 列也张成 Krylov 子空间 $\mathcal{K}(A, b, k)$
这样做有以下好处:

- ① 酉矩阵 Q 一定是良态的 $(\|Q\|_2 = 1)$，且容易求逆 $(Q^{-1} = Q^{\mathrm{H}})$
- ② 为得到 $Ax = b$ 的精确解，我们仅仅需要计算 Q 的前几列，远少于矩阵维数 n

记 Y 的 QR 分解为 $Y = QR$
暂且假定 $Y \in \mathbb{C}^{n\times n}$ 非奇异，则我们有:

$$Y^{-1}AY = (R^{-1}Q^{\mathrm{H}})A(QR) = C$$
$$\Leftrightarrow$$
$$H := Q^{\mathrm{H}}AQ = RCR^{-1}$$

考虑到 R, R^{-1} 均为上三角阵，且 C 是上 Hessenberg 矩阵
可以证明 $H := Q^{\mathrm{H}}AQ = RCR^{-1}$ 也是上 Hessenberg 矩阵.
特殊地，若 A 是 Hermite 阵，则 $T := Q^{\mathrm{H}}AQ = RCR^{-1}$ 是一个 Hermite 三对角阵.

实际计算中 Y 可能不是非奇异阵，故我们只需计算 Q 的部分列.
记 $Q := [q_1, \dots, q_r]$ (其中 $1 \le r \le \operatorname{rank}(Y)$)
根据 $AQ = QH$ (其中 $H \in \mathbb{C}^{r\times r}$ 为上 Hessenberg 阵)，因而有 $Aq_j = \sum_{i=1}^{j+1} h_{i,j}q_i$
由于 Q 列标准正交，故我们有 $q_k^{\mathrm{H}}Aq_j = \sum_{i=1}^{j+1} h_{i,j}q_k^{\mathrm{H}}q_i = 0 + h_{k,j}\cdot 1 = h_{k,j}\ (1 \le k \le j)$
最后有 $h_{j+1,j}q_{j+1} = Aq_j - \sum_{i=1}^{j} h_{i,j}q_i$
得到 $AQ = QH + \text{rank-one}$
其中 rank-one 代表遗留的秩一矩阵，仅在最后一列有非零元素.
它的存在是因为 Aq_r 不一定能完全表示为 $q_1, \dots, q_r$ 的线性组合.

于是我们得到**基于 Gram-Schmidt 正交化的 Arnoldi 过程**:
(Matlab 代码参考 Homework 6 Problem 4 Algorithm 1)
(**基于 Householder 变换的 Arnoldi 过程**参见 Homework 6 Problem 4 Algorithm 2)

Given square matrix $A \in \mathbb{C}^{n\times n}$, vector $b \in \mathbb{C}^n$ and integer $1 \le r \le n$

function: $[Q, H] = \text{Gram\_Schmidt\_Arnoldi}(A, b, \text{tolerance}, \text{modified}, \text{reorthogonalized})$

$n = \dim(A)$

$Q = \text{zeros}(n, n)$

$Q(1:n, 1) = \dfrac{b}{\|b\|_2}$

$H = \text{zeros}(n, n)$

$\delta = \text{zeros}(n, 1)$

---

if reorthogonalized == TRUE

max_iter = 2

else

max_iter = 1

end

---

for $k = 1 : r-1$

$Q(1:n, k+1) = AQ(1:n, k)$

if modified == TRUE (MGS: Modified Gram-schmidt)

for iter = 1:max_iter

for $i = 1 : k$

$\delta(i) = Q(1:n, i)^{\mathrm{H}} Q(1:n, k+1)$

$H(i, k) = H(i, k) + \delta(i)$

$Q(1:n, k+1) = Q(1:n, k+1) - \delta(i) Q(1:n, i)$

end

end

else (CGS: Classic Gram-Schmidt)

for iter = 1:max_iter

$\delta(1:k) = Q(1:n, 1:k)^{\mathrm{H}} Q(1:n, k+1)$

$H(1:k, k) = H(1:k, k) + \delta(1:k)$

$Q(1:n, k+1) = Q(1:n, k+1) - Q(1:n, 1:k)\delta(1:k)$

end

end

$H(k+1, k) = \|Q(1:n, k+1)\|_2$

if $H(k+1, k) <$ tolerance (indicates linear dependence)

$r = k$

break

else

$Q(1:n, k+1) = \dfrac{1}{H(k+1, k)} Q(1:n, k+1)$

end

end

$H(1:r, r) = Q(1:n, 1:r)^{\mathrm{H}} (AQ(1:n, r))$ (fill the last column)

$Q = Q(1:n, 1:r)$

$H = H(1:r, 1:r)$

end

2.4.4 Lanzcos 过程

当 A 是对称阵时，Arnoldi 过程得到的 $T := Q^{\mathrm{T}} AQ$ 是一个对称三对角阵.

理论上，我们可将 Arnoldi 过程简化为 **Lanczos 过程** (考虑到 T 的很多上三角元为零)

给定对称阵 $A \in \mathbb{R}^{n\times n}$ 和单位向量 $q_1 \in \mathbb{R}^n$

我们记:

$$\widetilde{T}_k = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \beta_2 & \alpha_3 & \ddots & \\ & & \ddots & \ddots & \beta_{k-1} \\ & & & \beta_{k-1} & \alpha_k \\ & & & & \beta_k \end{bmatrix}$$

记 $Q_k := [q_1, \ldots, q_k]$ (其中 $1 \le k \le \operatorname{rank}(\mathcal{K}(A, q_1, n))$)
(其中 Krylov 子空间 $\mathcal{K}(A, q_1, n) = \operatorname{span}\{q_1, Aq_1, \ldots, A^{n-1}q_1\}$)
根据 $AQ_k = Q_{k+1}\widetilde{T}_k$ 可知 $Aq_k = \beta_{k-1}q_{k-1} + \alpha_k q_k + \beta_k q_{k+1}$
由于 $q_1, \ldots, q_{k+1}$ 标准正交，故我们有 $q_k^{\mathrm{T}} A q_k = \alpha_k$
最后有 $\beta_k q_{k+1} = Aq_k - \alpha_k q_k - \beta_{k-1}q_{k-1}$

(Lanczos 过程, Matrix Computation $9.1.2$ 节)

Given symmetric matrix $A \in \mathbb{R}^{n\times n}$ and $q_1 \in \mathbb{R}^n$ such that $\|q_1\|_2 = 1$

$k = 0;\ \beta_0 = 1;\ q_0 = 0_n;\ r_0 = q_1$
while $\beta_k \neq 0$
$\quad q_{k+1} = \dfrac{r_k}{\beta_k}$
$\quad k = k + 1$
$\quad \alpha_k = q_k^{\mathrm{T}} A q_k$
$\quad r_k = Aq_k - \alpha_k q_k - \beta_{k-1}q_{k-1}$
$\quad \beta_k = \|r_k\|_2$
end

Matlab 代码如下: (Homework 6 Problem 6)

```
function [Q, T] = Lanczos(A, b, r, tolerance)
    % Lanczos Algorithm to compute the tridiagonal matrix T and orthonormal basis Q
    %
    % Input:
    %   A   - A symmetric matrix (n x n)
    %   b   - A vector (n x 1)
    %   r   - Number of Lanczos iterations
    %
    % Output:
    %   Q   - Orthogonal basis vectors (n x m)
    %   T   - Tridiagonal matrix (m x m)

    % Initialize variables
    n = size(A, 1);  % Get the size of matrix A
    Q = zeros(n, r);  % Initialize orthogonal basis matrix Q with zeros (n x r)
    T = zeros(r, r);  % Initialize tridiagonal matrix T with zeros (r x r)

    % Normalize the initial vector b to create the first orthogonal basis vector
    Q(:, 1) = b / norm(b);

    % Start the Lanczos iterations
    for j = 1:r
        % Compute the matrix-vector product of A and the j-th basis vector
        z = A * Q(:, j);

        % Compute the diagonal entry of T (the Rayleigh quotient)
        T(j, j) = Q(:, j)' * z;

        if j == 1
            % For the first iteration, adjust z by subtracting the first term
            z = z - T(j, j) * Q(:, j);
        else
            % For subsequent iterations, subtract the contributions from the last two basis
vectors
            z = z - T(j, j) * Q(:, j) - T(j-1, j) * Q(:, j-1);
```

```
        end

        % Compute the off-diagonal entry of T and check for convergence
        T(j, j+1) = norm(z, 2);  % Norm of the vector z is the off-diagonal element
        T(j+1,j) = T(j, j+1);    % Since T is symmetric

        if norm(z, 2) < tolerance
            % If the norm of z is below the tolerance, reduce the number of iterations
            r = j-1;
            break;  % Exit the loop if convergence is achieved
        else
            % If not converged, normalize z to create the next basis vector
            Q(:, j+1) = z / T(j, j+1);
        end
    end

    % Return results by truncating Q and T to the size of the computed basis
    Q = Q(:, 1:r);  % Truncate Q to include only the computed basis vectors
    T = T(1:r, 1:r);  % Truncate T to the size of the computed matrix
end
```

The End