FDU 数值算法 0. 基础知识

本文根据邵老师授课内容整理而成，并参考了以下教材:

- Accuracy and Stability of Numerical Algorithms (2nd Edition, N. Higham) Chapter $1, 2, 3$ & Appendix C
- 数值线性代数 (第 2 版, 徐树方, 高立, 张平文) 绪论
- Computer Systems: A Programmer's Perspective (3rd Edition, R. Bryant & D. O'Hallaron) Chapter 2
- What every computer scientist should know about floating-point arithmetic (David Goldberg)

欢迎批评指正!

0.1 An Introduction

0.1.1 基本问题

数值线性代数主要研究以下四类基本问题:

- **求解线性方程组:**
  给定非奇异矩阵 $A \in \mathbb{C}^{n\times n}$ 和向量 $b \in \mathbb{C}^n$，计算 $Ax = b$ 的解 $x_\star \in \mathbb{C}^n$.
- **线性最小二乘问题:**
  给定矩阵 $A \in \mathbb{C}^{m\times n}$ 和向量 $b \in \mathbb{C}^m$，计算 $\min_{x\in\mathbb{C}^n} \|Ax - b\|_2$ 的解 $x_\star \in \mathbb{C}^n$.
- **矩阵特征值问题:**
  给定方阵 $A \in \mathbb{C}^{n\times n}$，计算满足 $Ax = x\lambda$ 的特征对 (λ, x).
- **矩阵奇异值问题:**
  给定矩阵 $A \in \mathbb{C}^{m\times n}$，计算满足 $Av = u\sigma$ 和 $A^{\mathrm{H}}u = v\sigma$ 的奇异值 σ 和奇异向量 $u \in \mathbb{C}^m, v \in \mathbb{C}^n$.

还有其他的问题，例如:

- 矩阵函数 $f(A)$ 的计算问题
- Sylvester 方程 $AX - XB = C$ 的求解问题
- 工程应用问题: 为什么带有奇数叶片的风扇往往比偶数叶片的更稳定?

0.1.2 误差

设 $\hat{x}$ 是实数 $x \in \mathbb{R}$ 的近似.
衡量 $\hat{x}$ 准确性的最有用的两个标准是**绝对误差** (absolute error) 和**相对误差** (relative error):

$$\begin{aligned}\mathrm{E}_{\mathrm{abs}}(\hat{x}) &:= |x - \hat{x}| \\ \mathrm{E}_{\mathrm{rel}}(\hat{x}) &:= \frac{\mathrm{E}_{\mathrm{abs}}(\hat{x})}{|x|} = \frac{|x - \hat{x}|}{|x|}.\end{aligned}$$

值得注意的是，当 $x = 0$ 时，相对误差 $\mathrm{E}_{\mathrm{rel}}(\hat{x})$ 是未定义的.

在数值计算中，误差主要来源于三个方面:
数据不确定性 (data uncertainty)、**舍入误差** (rounding error) 和**截断误差** (truncation error).

- 数据不确定性通常来源于数据的测量和存储，可以通过针对具体问题的扰动理论来分析.
- 有限精度算术运算不可避免地会产生舍入误差.
  对于同一问题，我们需要针对具体算法进行舍入误差分析.
- 很多数值方法 (例如数值积分的梯形公式) 都可以通过截断 Taylor 级数推导出来，
  而被截断的项便产生了截断误差.
  在某些情形下，截断误差的影响可能远大于舍入误差.

设 $f:\mathbb{R}^n\mapsto\mathbb{R}$ 是一阶连续可微的实值函数，给定 $x\in\mathbb{R}^n$.
设 $\hat{y}$ 是通过有限精度运算得到的 $y=f(x)$ 的近似值.
我们称 $\hat{y}$ 的绝对误差或相对误差为**前向误差** (forward error).
换一个角度，我们考虑对 x 做多大的扰动 Δx，
才能使扰动后的精确解 $f(x+\Delta x)$ 恰好等于当前计算得到的 $\hat{y}$?
通常来说，这样的 Δx 可能有很多个，我们可以取范数最小的扰动，即求解如下优化问题:

$$\begin{aligned}\min\quad & \|\Delta x\|\\ \text{s.t.}\quad & f(x+\Delta x)=\hat{y}.\end{aligned}$$

设其最优解为 $\Delta x_\star$，则我们称 $\|\Delta x_\star\|$ 或 $\|\Delta x_\star\|/\|x\|$ 为**后向误差** (backward error).

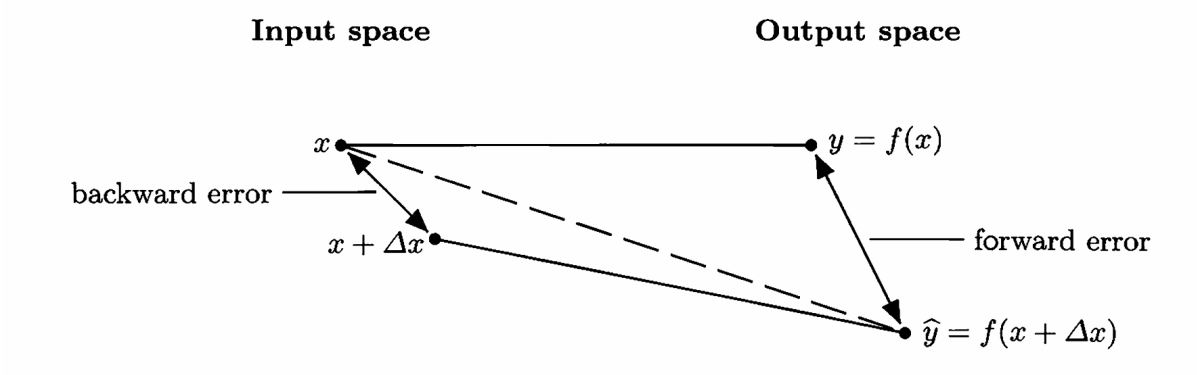

Figure 1.1. *Backward and forward errors for $y=f(x)$. Solid line = exact; dotted line = computed.*

根据 f 的一阶 Taylor 展开我们有

$$\begin{aligned}\hat{y}-y&=f(x+\Delta x)-f(x)\\&=\nabla f(x)^{\mathrm{T}}\Delta x+O(\|\Delta x\|^2),\end{aligned}$$

于是我们有

$$\begin{aligned}\frac{|\hat{y}-y|}{|y|}&=\left|\frac{\nabla f(x)^{\mathrm{T}}\Delta x}{y}\right|+O(\|\Delta x\|^2)\quad(\text{Cauchy–Schwarz inequality})\\&\lesssim\frac{\|\nabla f(x)\|\cdot\|\Delta x\|}{|f(x)|}\\&=\frac{\|\nabla f(x)\|\cdot\|x\|}{|f(x)|}\cdot\frac{\|\Delta x\|}{\|x\|}.\end{aligned}$$

我们称 $\kappa(x):=\|\nabla f(x)\|\cdot\|x\|/|f(x)|$ 为 f 的 (相对) **条件数** (condition number).
总之，我们有如下渐近不等关系:

$$\text{forward error}\lesssim\text{condition number}\times\text{backward error}.$$

换言之，计算解的后向误差在转化为前向误差时可能会被放大，放大因子可达到问题的条件数.
对于病态问题 (即条件数较大的问题)，即使计算解的后向误差很小，其前向误差也可能非常大.

对计算解的前向误差进行界定的过程称为**前向误差分析** (forward error analysis);
对计算解的后向误差进行界定的过程称为**后向误差分析** (backward error analysis).

- 一方面，我们可以将舍入误差解释为数据扰动.
  如果后向误差不大于数据本身的不确定性，那么计算得到的解便是可以被接受的.
- 另一方面，我们可以将界定前向误差的问题转化为扰动理论的问题.
  而对于许多问题，扰动理论已相当成熟，
  并且对于给定问题只需开发一次，不必针对每种数值方法重复分析.

某些情况下 (例如 f 在 x 附近具有严格单调性),
我们可能找不到一个 Δx 使得 $f(x+\Delta x)=\hat{y}$.
此时我们可以分析**混合前向-后向误差** (mixed forward-backward error):

$$f(x+\Delta x)=\hat{y}+\Delta y,\quad \|\Delta x\|\le\eta_1\|x\|,\quad |\Delta y|\le\eta_2|y|,$$

其中 $\eta_1,\eta_2>0$ 为充分小的正实数.
这意味着，计算得到的值 $\hat{y}$ 与由稍微扰动后的输入 $x+\Delta x$ 所得到的值 $f(x+\Delta x)$ 几乎没有区别.
换言之，$\hat{y}$ 对于近似正确的数据 $x+\Delta x$ 来说，近似就是正确的答案.

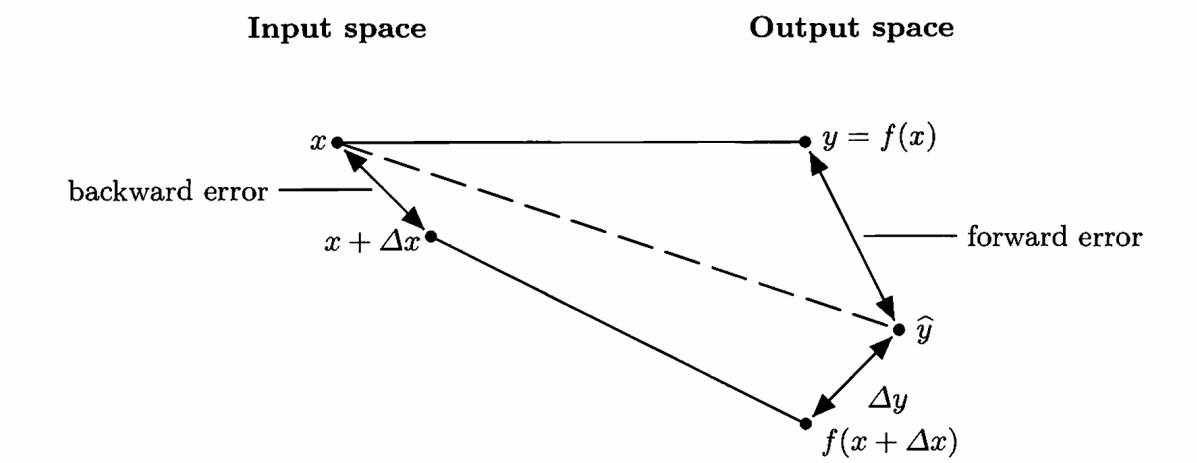

Figure 1.2. *Mixed forward–backward error for $y=f(x)$. Solid line = exact; dotted line = computed.*

0.1.3 舍入误差

在有限精度的浮点运算中，若用 $\mathrm{fl}(\cdot)$ 代表对一个表达式的求值，
则舍入误差的 **Wilkinson 模型**可描述为

$$\mathrm{fl}(x\ \mathrm{op}\ y)=(x\ \mathrm{op}\ y)(1+\delta),\quad |\delta|\le u,$$

其中 u 为对应的浮点数制的**单位舍入误差** (unit roundoff)，代表浮点舍入到最近可表示数时的**最大相对误差**,
而 op 代表 $+,-,*,/$ 四个运算中的一个.
通常来说，平方根运算在硬件层面已有直接支持，因此我们也可以认为其舍入误差满足上述模型:

$$\mathrm{fl}(\sqrt{x})=\sqrt{x}(1+\delta),\quad |\delta|\le u.$$

(Accuracy and Stability of Numerical Algorithms, Theorem 2.3)
在某些情况下，我们还会使用如下舍入误差模型:

$$\begin{cases}\mathrm{fl}(x\ \mathrm{op}\ y)=(x\ \mathrm{op}\ y)/(1+\delta), & \mathrm{op}=+,-,*,/\\ \mathrm{fl}(\sqrt{x})=\sqrt{x}/(1+\delta).\end{cases}\qquad(|\delta|\le u)$$

(累积相对误差的上界, 数值线性代数, 定理 2.3.3)
若 $|\delta_i|\le u\ (i=1,\dots,n)$ 且 $nu\le 1$，则我们有

$$\begin{aligned}\left|\prod_{i=1}^{n}(1+\delta_i)-1\right| &\le (1+u)^n-1 && \text{(Wilkinson's bound)}\\ &\le (1-u)^{-n}-1 && \text{(Shao Meiyue's bound)}\\ &\le \frac{nu}{1-nu} && \text{(Higham's bound)}\\ &= nu+O(u^2).\end{aligned}$$

> **(邵言邵语)**
> 这个界是 Wilkinson 提出来的 (停顿);
> 这个界是 Higham 提出来的 (停顿);
> 这个界是我提出来的 (哄堂大笑).

记 $\gamma_n := nu/(1-nu)$ 为 n 层浮点运算的累积相对误差的上界.

Proof:
前两个不等号是显然的，下面证明第三个不等号在 $nu \le 1$ 的条件下成立.
定义:

$$\begin{aligned} f(x) &:= (1-x)^{-n} - 1 - \frac{nx}{1-nx} \\ &= \frac{(1-x)^{-n}(1-nx) - 1}{1-nx} \end{aligned} \qquad (nx \le 1)$$

记其分子为 $g(x)$，则导数 $g'(x)$ 为

$$\begin{aligned} g'(x) &= -n \cdot (1-x)^{-(n+1)} \cdot (-1) \cdot (1-nx) + (1-x)^{-n} \cdot (-n) \\ &= -n(n-1)x(1-x)^{-n-1} \\ &\le 0. \end{aligned}$$

注意到 $g(0) = (1-0)^{-n}(1-0) - 1 = 0$,
故对于任意 $x \in (0, 1/n]$ 我们都有 $g(x) \le g(0) = 0$ 成立.
因此当 $nx \le 1$ 时，我们有 $f(x) = g(x)/(1-nx) \le 0$ 成立.
这说明第三个不等号在 $nu \le 1$ 的条件下成立.

(一个简单的例子)
考虑 $\beta = \sum_{i=1}^{n} \alpha_i$ 的舍入误差分析.
如果我们按标准的 "顺序加法" (left-to-right) 计算:

$$\mathrm{fl}(\beta) = \mathrm{fl}(\mathrm{fl}(\cdots \mathrm{fl}(\mathrm{fl}(\alpha_1 + \alpha_2) + \alpha_3) + \cdots + \alpha_{n-1}) + \alpha_n),$$

则会有 $n-1$ 层舍入误差的累积，相对误差上界为 γ_{n-1}:

$$\frac{|\mathrm{fl}(\beta) - \beta|}{\sum_{i=1}^{n} |\alpha_i|} \le \gamma_{n-1},$$

其中左侧代表问题的困难程度，我们是管不了的;
右侧的 γ_{n-1} 我们是管得了的，可通过算法改进.
例如我们可以将 $\alpha_1, \ldots, \alpha_n$ 两两结合着去算，这样只有 $\lceil \log_2(n) \rceil$ 层舍入误差的累积:

$$\frac{|\mathrm{fl}(\beta) - \beta|}{\sum_{i=1}^{n} |\alpha_i|} \le \gamma_{\lceil \log_2(n) \rceil},$$

但实际应用中我们很少会去这样算.

(数值线性代数, 例 2.3.1)
设 x, y 是 n 维浮点数向量，试估计内积运算 $\mathrm{fl}(x^{\mathrm{T}} y)$ 的相对误差上界.

Solution:
记 $S_k = \mathrm{fl}(\sum_{i=1}^{k} x_i y_i)$，我们有:

$$\begin{cases} S_1 = x_1 y_1 (1 + r_1) & (|r_1| \le u) \\ S_k = \mathrm{fl}(S_{k-1} + \mathrm{fl}(x_k y_k)) & \\ \quad = (S_{k-1} + x_k y_k (1 + r_k))(1 + \delta_k) & (|r_k|, |\delta_k| \le u,\ k \ge 2) \end{cases}$$

定义 $\delta_1 = 0$ 和 $S_0 = 0$，则有:

$$
\begin{aligned}
\mathrm{fl}(x^{\mathrm{T}}y) &= S_n \\
&= (S_{n-1} + x_n y_n(1+r_n))(1+\delta_n) \\
&= S_{n-1}(1+\delta_n) + x_n y_n(1+r_n)(1+\delta_n) \\
&= \cdots \\
&= S_0 \prod_{i=1}^{n}(1+\delta_i) + \sum_{k=1}^{n}\left(x_k y_k(1+r_k)\prod_{i=k}^{n}(1+\delta_i)\right) \quad (\text{note that } S_0 = 0) \\
&= 0 + \sum_{k=1}^{n}\left(x_k y_k(1+r_k)\prod_{i=k}^{n}(1+\delta_i)\right) \quad (\text{denote } \varepsilon_k := (1+r_k)\prod_{i=k}^{n}(1+\delta_i) - 1 \text{ for all } k \geq 1) \\
&= \sum_{k=1}^{n} x_k y_k(1+\varepsilon_k),
\end{aligned}
$$

其中 $1+\varepsilon_k = (1+r_k)\prod_{i=k}^{n}(1+\delta_i)$ $(k = 1, \ldots, n)$.
于是 $\mathrm{fl}(x^{\mathrm{T}}y)$ 的绝对误差上界为:

$$
\begin{aligned}
|\mathrm{fl}(x^{\mathrm{T}}y) - x^{\mathrm{T}}y| &\leq \sum_{k=1}^{n}|\varepsilon_k||x_k y_k| \\
&\leq \gamma_n \sum_{k=1}^{n}|x_k y_k| \\
&\leq \gamma_n |x|^{\mathrm{T}}|y|,
\end{aligned}
$$

其中 $\gamma_n = nu/(1-nu)$ 代表 n 层浮点运算的累积相对误差的上界，u 代表单位舍入误差.

值得注意的是，$\mathrm{fl}(x^{\mathrm{T}}y)$ 的相对误差上界为:

$$
\frac{|\mathrm{fl}(x^{\mathrm{T}}y) - x^{\mathrm{T}}y|}{|x^{\mathrm{T}}y|} \leq \gamma_n \frac{\sum_{k=1}^{n}|x_k y_k|}{|x^{\mathrm{T}}y|}.
$$

若 $|x^{\mathrm{T}}y| = |\sum_{k=1}^{n} x_k y_k| \ll \sum_{k=1}^{n}|x_k y_k|$ (即 x, y 近似正交)，则 $\mathrm{fl}(x^{\mathrm{T}}y)$ 的相对误差可能会很大.
此时，一种方法是采用更高精度的浮点数进行计算，最后将计算结果舍入到标准浮点数;
另一种方法是使用乘加运算和 Kahan 补偿求和提高精度，我们将具体算法放到后文.

最后我们分析一下矩阵基本运算的舍入误差.
对于矩阵 $A = [a_{ij}]$，我们定义 $|A| := [|a_{ij}|]$ (即逐元素取绝对值)，
并规定 $|A| \leq |B|$ 当且仅当对于任意 i, j 都有 $|a_{ij}| \leq |b_{ij}|$ 成立.

根据 Wilkinson 模型和前文对向量内积的舍入误差分析可知，
对于任意 n 阶浮点数矩阵 A, B 和浮点数 α，我们都有:

$$
\begin{cases}
\mathrm{fl}(\alpha A) = \alpha A + \Delta & (|\Delta| \leq u|\alpha A|) \\
\mathrm{fl}(A + B) = (A + B) + \Delta & (|\Delta| \leq u|A + B|) \\
\mathrm{fl}(AB) = AB + \Delta & (|\Delta| \leq \gamma_n |A||B|),
\end{cases}
$$

其中 $\gamma_n = nu/(1-nu)$ 代表 n 层浮点运算的累积相对误差的上界，u 代表单位舍入误差.

0.1.4 相消

两个几乎相等的数相减时所导致的精度损失现象称为**相消** (cancellation).
设实数 $\alpha, \beta \in \mathbb{R}$ 的浮点表示为:

$$
\begin{aligned}
\hat{\alpha} &= \mathrm{fl}(\alpha) = \alpha(1+\delta_1) \\
\hat{\beta} &= \mathrm{fl}(\beta) = \beta(1+\delta_2),
\end{aligned}
$$

其中 $|\delta_i| \leq u$ $(i = 1, 2)$，u 代表单位舍入误差.
现在考虑 α, β 进行浮点减法:

$$
\begin{aligned}
\mathrm{fl}(\alpha-\beta) &= \mathrm{fl}(\mathrm{fl}(\alpha)-\mathrm{fl}(\beta)) \\
&= \mathrm{fl}(\hat{\alpha}-\hat{\beta}) \\
&= (\hat{\alpha}-\hat{\beta})(1+\delta_3) \\
&= (\alpha(1+\delta_1)-\beta(1+\delta_2))(1+\delta_3) \\
&= ((\alpha-\beta)+(\alpha\delta_1-\beta\delta_2))(1+\delta_3),
\end{aligned}
$$

其中 $|\delta_3| \leq u$.
因此我们有:

$$
\begin{aligned}
\left|\frac{\mathrm{fl}(\alpha-\beta)-(\alpha-\beta)}{\alpha-\beta}\right| &= \left|\frac{(\alpha\delta_1-\beta\delta_2)(1+\delta_3)+(\alpha-\beta)\delta_3}{\alpha-\beta}\right| \\
&= \left|\frac{\alpha((1+\delta_1)(1+\delta_3)-1)-\beta((1+\delta_2)(1+\delta_3)-1)}{\alpha-\beta}\right| \\
&\leq \frac{|\alpha|\cdot|(1+\delta_1)(1+\delta_3)-1|+|\beta|\cdot|(1+\delta_2)(1+\delta_3)-1|}{|\alpha-\beta|} \\
&\leq \frac{|\alpha|+|\beta|}{|\alpha-\beta|}\gamma_2,
\end{aligned}
$$

其中 $\gamma_2 = 2u/(1-2u)$.
当 $\alpha \approx \beta$ 时，$|\alpha|+|\beta| \gg |\alpha-\beta|$，相对舍入误差上界会很大，很可能出事情.

值得注意的是，尽管相消在一般情况下是一种危险的操作，但它并非总是如此.

- 首先，如果 $\alpha, \beta \in \mathbb{R}$ 直接可用浮点数表示，即它们转换成浮点数时是无舍入误差的 (error-free):

  $$
  \begin{aligned}
  \delta_1 = \delta_2 &= 0 \\
  \hat{\alpha} = \mathrm{fl}(\alpha) &= \alpha(1+\delta_1) = \alpha \\
  \hat{\beta} = \mathrm{fl}(\beta) &= \beta(1+\delta_2) = \beta,
  \end{aligned}
  $$

  则我们有:

  $$
  \begin{aligned}
  \mathrm{fl}(\alpha-\beta) &= \mathrm{fl}(\mathrm{fl}(\alpha)-\mathrm{fl}(\beta)) \\
  &= \mathrm{fl}(\hat{\alpha}-\hat{\beta}) \\
  &= (\hat{\alpha}-\hat{\beta})(1+\delta_3) \\
  &= (\alpha-\beta)(1+\delta_3).
  \end{aligned}
  $$

  因此我们有:

  $$
  \left|\frac{\mathrm{fl}(\alpha-\beta)-(\alpha-\beta)}{\alpha-\beta}\right| = |\delta_3| \leq u.
  $$

  此时，即使 $\alpha \approx \beta$，浮点减法也是安全的.
- 其次，相消有时是问题固有的**病态性 (ill conditioning)** 的一种表现，因此可能是不可避免的.
- 最后，相消的影响还取决于其结果在后续计算中的作用.
  例如，当 $\gamma \gg \alpha \approx \beta > 0$ 时，尽管在计算 $\alpha-\beta$ 的过程中会发生相消，
  但对于后续 $\gamma+(\alpha-\beta)$ 的运算而言是相对无害的.

考虑一个具体的例子.
注意到当 $x \to 0$ 时，我们有:

$$
1-\cos(x) = 2\left(\sin\left(\frac{x}{2}\right)\right)^2 \to 2\cdot\left(\frac{x}{2}\right)^2 = \frac{1}{2}x^2
$$

尽管理论上 $1-\cos(x)$ 和 $2(\sin(x/2))^2$ 是等价的，
但在数值计算中，前者可能因相消而导致严重的精度损失.
MATLAB 代码验证:

```
% IEEE 754 单精度浮点数
x_vals = single(logspace(-3, -3.65, 10));

fprintf('%8s %18s %22s %22s %10s\n', ...
    'x', 'cos(x)', '(1-cos(x))/x^2', '0.5*(sin(x/2)/(x/2))^2', 'RelError');

for x = x_vals
    c = cos(x);

    % 直接计算形式 (1 - cos(x))/x^2 (可能相消)
    f1 = (single(1) - c) / (x.^2);

    % 数值稳定形式 0.5*(sin(x/2)/(x/2))^2
    f2 = single(0.5) * (sin(x/2)./(x/2)).^2;

    % 相对误差
    rel_err = abs(f1 - f2) ./ abs(f2);

    fprintf('%12.3e %15.8f\t %15.8f\t %15.8f %15.2e\n', ...
        double(x), double(c), double(f1), double(f2), double(rel_err));
end
```

运行结果:

x	cos(x)	(1-cos(x))/x^2	0.5*(sin(x/2)/(x/2))^2	RelError
1.000e-03	0.99999952	0.47683710	0.50000000	4.63e-02
8.468e-04	0.99999964	0.49874187	0.50000000	2.52e-03
7.171e-04	0.99999976	0.46369132	0.50000000	7.26e-02
6.072e-04	0.99999982	0.48499215	0.50000000	3.00e-02
5.142e-04	0.99999988	0.45090798	0.50000000	9.82e-02
4.354e-04	0.99999988	0.62882864	0.50000000	2.58e-01
3.687e-04	0.99999994	0.43847692	0.50000000	1.23e-01
3.122e-04	0.99999994	0.61149257	0.50000000	2.23e-01
2.644e-04	0.99999994	0.85277742	0.50000000	7.06e-01
2.239e-04	1.00000000	0.00000000	0.50000000	1.00e+00

我们可以看出，当 $x = 2.644 \times 10^{-4}$ 时，尽管 $\cos(x)$ 的精度达到了小数点后第 8 位，
但 $(1-\cos(x))/x^2$ 的结果为 0.85277742，而理论值应该接近 0.5，
这说明由于相消的影响，直接计算形式的结果甚至无法保证小数点后 1 位的正确性.

更为极端的情况出现在 $x = 2.239 \times 10^{-4}$ 时.
此时，$\cos(x)$ 与 1 的差异已经小于机器精度 $\text{eps} = 2^{-23} \approx 1.19 \times 10^{-7}$.
因此在浮点表示中，$\cos(x)$ 会被直接舍入为 1，从而使得 $1-\cos(x)$ 被计算为 0.
换言之，相消 "完全吃掉了" 有用的信息.

0.1.5 乘加运算

某些机器实现了**融合乘加** (fused multiply-add, FMA) 运算，简称**乘加运算**.
它允许一次浮点乘法紧接着一次加法或减法 (即 $x*y \pm z$) 作为一个单独的浮点运算来执行.
这样一来，原本需要经历两次舍入的计算过程被压缩为一次:

$$\text{fl}(x*y \pm z) = (x*y \pm z)(1+\delta), \quad |\delta| \le u.$$

此外，FMA 能在与单个乘法或加法相同的时钟周期内完成，
因此它在速度和精度方面都具有优势.

利用 FMA 可以在计算两个 n 维向量 x, y 的内积 $x^{\mathrm{T}}y$ 时，
仅产生 n 次舍入误差，而不是通常的 $2n-1$ 次.
(不过舍入误差仍然会累积 n 层，因此相对误差上界仍然是 $\gamma_n|x|^{\mathrm{T}}|y|/|x^{\mathrm{T}}y|$)
任何以内积作为主要运算的算法都能因此而受益.
事实上，在大多数科学计算中，占主导地位的基本运算模式都是一乘一加的组合.

特别地，考虑三个复数的一次乘加:

$$\begin{aligned}(a+b\mathrm{i})(c+d\mathrm{i})+(e+f\mathrm{i}) &= (ac-bd+e)+\mathrm{i}(ad+bc+f)\\ &= (ac-(bd+e))+\mathrm{i}(ad+(bc+f)).\end{aligned}$$

它一共需要四次乘法和四次加法，可以写成四次 FMA 的组合.

利用乘加指令，我们可以追踪乘法过程中因舍入而丢弃的低阶项:

$$\begin{aligned}p &= \mathrm{fl}(ab) = ab(1+\delta) \qquad (|\delta|\le u)\\ e &= \mathrm{fl}(ab-p) = -ab\cdot\delta \quad (\text{use FMA})\end{aligned}$$

其中 p 是乘积 ab 的舍入值，e 是因舍入而丢弃的低阶项，可以证明 $p+e$ 就是精确乘积 ab.
这是因为 ab 的有效数字至多有 $2(n_{\mathrm{f}}+1)$ 位 (其中 n_{f} 代表 frac 字段的位数)，
舍入得到的 p 的有效数字是前 $n_{\mathrm{f}}+1$ 位，而 e 的有效数字恰好是那些被丢弃的位.
据此定义如下辅助函数:

$$\begin{aligned}&\text{function } [p,\text{error}] = \mathrm{TwoProduct}(a,b)\\ &\quad p = \mathrm{fl}(ab)\\ &\quad \text{error} = \mathrm{fl}(ab-p) \quad (\text{use FMA})\\ &(\text{Note that } p+\text{error} = ab \text{ holds exactly})\end{aligned}$$

类似地，我们可以追踪加法过程中因舍入而丢弃的低阶项，定义如下辅助函数:

$$\begin{aligned}&\text{function } [s,\text{error}] = \mathrm{TwoSum}(a,b)\\ &\quad s = \mathrm{fl}(a+b)\\ &\quad a\text{-prime} = \mathrm{fl}(s-b)\\ &\quad a\text{-error} = \mathrm{fl}(a-a\text{-prime})\\ &\quad b\text{-prime} = \mathrm{fl}(s-a)\\ &\quad b\text{-error} = \mathrm{fl}(b-b\text{-prime})\\ &\quad \text{error} = \mathrm{fl}(a\text{-error}+b\text{-error})\\ &(\text{Note that } s+\text{error} = a+b \text{ holds exactly})\end{aligned}$$

基于上述辅助函数，我们可以给出通过误差补偿提高精度的内积计算算法:
(存疑: $\mathrm{err}2$ 需要合并到 c 中吗?)

$$\begin{aligned}&\text{function } s = \mathrm{CompensatedDot}(x,y)\\ &\quad s = 0\\ &\quad c = 0 \quad (\text{compensation accumulator})\\ &\quad n = \mathrm{length}(x)\\ &\quad \text{for } i = 1:n\\ &\qquad p,\mathrm{err}0 = \mathrm{TwoProduct}(x_i,y_i)\\ &\qquad s,\mathrm{err}1 = \mathrm{TwoSum}(s,p)\\ &\qquad c,\mathrm{err}2 = \mathrm{TwoSum}(c,\mathrm{err}1)\\ &\qquad c = \mathrm{fl}(c+\mathrm{err}0)\\ &\qquad s,\mathrm{err\_fold} = \mathrm{TwoSum}(s,c)\\ &\qquad c = \mathrm{err\_fold}\\ &\quad \text{end}\end{aligned}$$

值得注意的是，并非所有的科学计算的基本运算模式都是一乘一加的组合.
我们有时会遇到形如 $a*b\pm c*d$ 的两乘一加的组合，
例如求解二次方程 $ax^2-2bx+c=0$ 时，计算判别式 b^2-ac 就涉及两乘一加;
再例如计算 2 阶方阵的行列式:

$$M=\begin{bmatrix}a & b\\ c & d\end{bmatrix},\quad \det(M)=ad-bc.$$

在这种情况下，如果 FMA 使用不当，可能会导致对 $ad-bc$ 的符号判断出错.
例如，当 $\mathrm{fl}(ad)<bc<ad$ 时，使用一次 FMA 的计算过程 $\mathrm{fl}(\mathrm{fl}(ad)-bc)$ 得到的是负值，
但不使用 FMA 的计算过程 $\mathrm{fl}(\mathrm{fl}(ad)-\mathrm{fl}(bc))$ 得到的是非负值，这是由舍入的单调性保证的.
因此我们不能滥用 FMA——正如 Kahan 所警告的那样: FMA should not be used indiscriminately!

Kahan 给出了计算 $ad-bc$ 时 FMA 的正确用法 (参见 Jeannerod et al. 2013).

$$\begin{aligned}&\text{function } \det=\mathrm{Kahan}(a,b,c,d)\\ &\quad p,\mathrm{err}=\mathrm{TwoProduct}(b,c)\quad (p+\mathrm{err}=bc \text{ holds exactly})\\ &\quad \widehat{\det}=\mathrm{fl}(ad-p)\quad (\text{use FMA})\\ &\quad \det=\mathrm{fl}(\widehat{\det}-\mathrm{err})\end{aligned}$$

该算法包含一次乘法、两次 FMA 运算以及一次减法.
它将在解实系数二次方程的过程中被用到，参见 **Homework 01 Problem 06**.

0.2 IEEE 754 标准

0.2.1 编码准则

IEEE 754 标准是一种二进制浮点算术系统，将实数表示为 $V=(-1)^s\times\Sigma\times 2^E$ 的形式.

- **① 符号 (sign)** s
  由一个单独的符号位 s 编码.
  它决定这个数是正数 $(s=0)$ 还是负数 $(s=1)$.
  值得注意的是，数值 0 在该标准下存在两种表示，分别为 $+0$ 和 -0.
- **② 指数 (exponent)** E
  由 n_e 位阶码字段 $\exp=e_{n_\mathrm{e}-1}\cdots e_0$ 以移码形式编码，偏移量 $\mathrm{bias}=2^{n_\mathrm{e}-1}-1$.
  对于单精度浮点数来说，$n_\mathrm{e}=8$，偏移量 $\mathrm{bias}=2^7-1=127$.
  对于双精度浮点数来说，$n_\mathrm{e}=11$，偏移量 $\mathrm{bias}=2^{10}-1=1023$.
- **③ 尾数 (fraction/mantissa)** f
  由 n_f 位尾数字段 $\mathrm{frac}=f_{n_\mathrm{f}-1}\cdots f_0$ 编码.
  最后一位的单位 (unit in the last place) $\mathrm{eps}=2^{-n_\mathrm{f}}$,
  又称**机器精度** (machine precision)，即 1 与大于 1 的最小可表示数之间的差.
  (注意它与**单位舍入误差** (unit roundoff) $u=\mathrm{eps}/2=2^{-n_\mathrm{f}-1}$ 是不同的概念)

  因此对应的二进制小数 f 的编码范围为 $0\sim 1-\mathrm{eps}$.
  有效数字 (significand) Σ 的范围为 $1\sim 2-\mathrm{eps}$ (规格化数, $\Sigma=1+f$) 或 $0\sim 1-\mathrm{eps}$ (非规格化数, $\Sigma=f$).
  对于单精度浮点数来说，$n_\mathrm{f}=23$，机器精度 $\mathrm{eps}=2^{-23}\approx 1.19\times 10^{-7}$.
  对于双精度浮点数来说，$n_\mathrm{f}=52$，机器精度 $\mathrm{eps}=2^{-52}\approx 2.22\times 10^{-16}$.

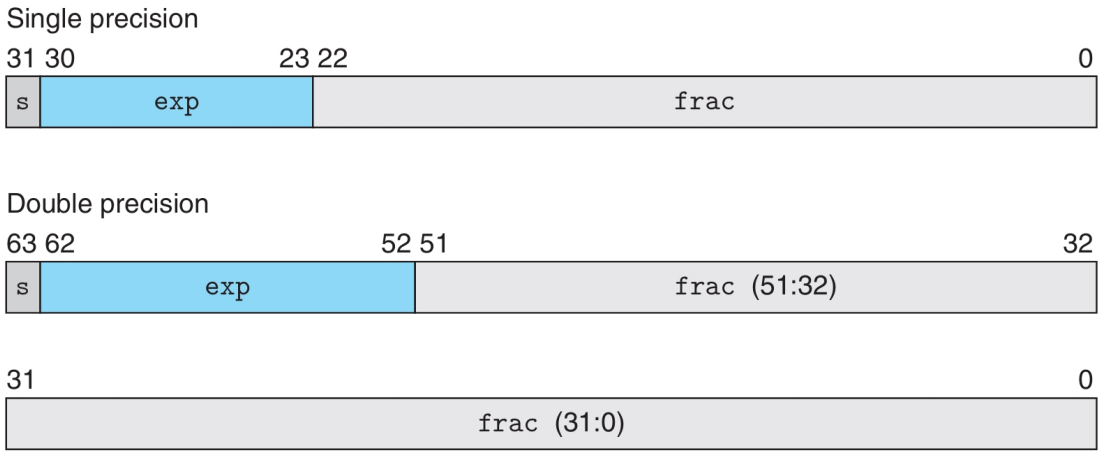

Figure 2.32 **Standard floating-point formats.** Floating-point numbers are represented by three fields. For the two most common formats, these are packed in 32-bit (single-precision) or 64-bit (double-precision) words.

根据 exp 字段的值，我们可将编码的数值分为三种情况:

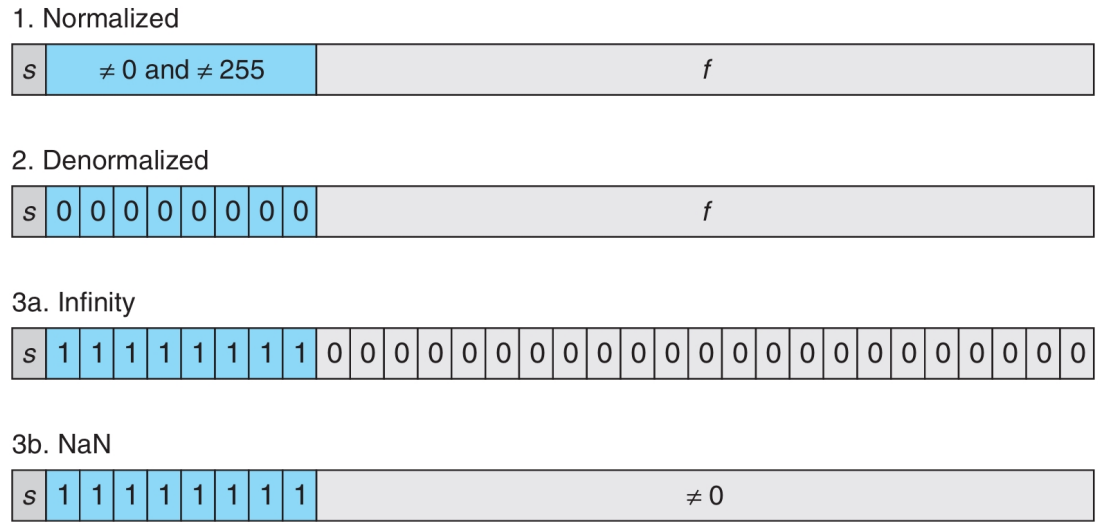

Figure 2.33 **Categories of single-precision floating-point values.** The value of the exponent determines whether the number is (1) normalized, (2) denormalized, or (3) a special value.

- **① 规格化数 (normalized value):**
  当 exp 字段的位模式既不全为 0，也不全为 1 时，该浮点数称为**规格化数**.
  编码范围为 $2-2^{n_e-1}\sim 2^{n_e-1}-1$，即 $1-\text{bias}\sim\text{bias}$.
  对于单精度浮点数来说，偏移量 $\text{bias}=127$，编码范围为 $-126\sim 127$.
  对于双精度浮点数来说，偏移量 $\text{bias}=1023$，编码范围为 $-1022\sim 1023$.

  frac 字段对应的二进制小数 f 的编码范围为 $0\sim 1-2^{-n_f}$，
  规格化数的有效数字 $\Sigma=1+f$ 的编码范围为 $1\sim 2-2^{-n_f}$.
  其中有效数字第一位的 1 没有出现在编码中，
  这称为**隐含的首一表示法** (implied leading 1 representation).

  对于单精度规格化浮点数来说，$n_f=23$，编码范围为:

  $$\pm[1\times 2^{-126},(2-2^{-23})\times 2^{+127}]\approx\pm[1.18\times 10^{-38},3.40\times 10^{+38}]$$

  对于双精度规格化浮点数来说，$n_f=52$，编码范围为:

  $$\pm[1\times 2^{-1022},(2-2^{-52})\times 2^{+1023}]\approx\pm[2.23\times 10^{-308},1.78\times 10^{+308}]$$

- **② 非规格化数 (subnormalized/denormalized value):**
  当 exp 字段的位模式全为 0 时，该浮点数称为**非规格化数**.
  编码的值为 $2-2^{n_e-1}=1-\text{bias}$ (以便平滑地过渡到规格化数).
  对于单精度浮点数来说，偏移量 $\text{bias}=127$，编码的值为 -126.
  对于双精度浮点数来说，偏移量 $\text{bias}=1023$，编码的值为 -1022.

  frac 字段对应的二进制小数 f 的编码范围为 $0\sim 1-2^{-n_f}$，
  非规格化数的有效数字 $\Sigma=f$ (没有隐含的首一) 的编码范围为 $0\sim 1-2^{-n_f}$.

  对于单精度非规格化浮点数来说，$n_f=23$，编码范围为:

  $$\pm[0\times 2^{-126},(1-2^{-23})\times 2^{-126}]\approx\pm[1.40\times 10^{-45},1.18\times 10^{-38}]$$

  对于双精度非规格化浮点数来说，$n_f=52$，编码范围为:

  $$\pm[0\times 2^{-1022},(1-2^{-52})\times 2^{-1022}]\approx\pm[4.94\times 10^{-324},2.23\times 10^{-308}]$$

  对于那些非常接近于 0 的数，非规格化数提供了一种称为**渐进下溢** (gradual underflow) 的机制，
  平滑地了填充从离 0 最近的规格化数到 0 之间的空隙，以避免直接向零下溢 (即直接舍入到零).

  此外，非规格化数还提供了表示数值 0 的方法，
  分别为 $+0$ (所有位全为 0) 和 -0 (除了符号位为 1，其他位全为 0).
  这两种数值 0 在进行加减乘除运算时是等价的，在比较运算中也相等 (即 $+0=-0$).
  值得说明的是，带符号的零为复数算术中的分支切割问题提供了一种优雅的处理方法，
  这也是 Kahan 将其纳入 IEEE 754 标准的核心动机，参见 Kahan 1987.

- **③ 特殊值 (special value):**
  当 exp 字段的位模式全为 1 时，该浮点数为特殊值.

  若 frac 字段为 0，则该浮点数表示无穷，
  符号位 $s=0$，则代表正无穷 $+\infty$; 符号位 $s=1$，则代表负无穷 $-\infty$.

  若 frac 字段为非零值，则该浮点数表示 NaN (非数, Not a Number)，
  意味着它既不是实数，也不是无穷，例如计算 $0/0, 0\times\infty, \sqrt{-1}, (+\infty)+(-\infty)$ 的结果便是 NaN.
  值得注意的是，NaN 的 frac 字段可以编码一些关于其来源的信息，用于后续的异常诊断.
  此外，NaN 还可以表示未初始化或缺失的数据.
  任何涉及 NaN 的算术运算都会返回 NaN 作为结果.
  NaN 在比较时被视为无序的，并且与包括其自身在内的所有值都不相等，
  因此我们通常需要使用 IEEE 推荐的 `isnan` 函数 (如果有提供的话) 来检测 NaN.

IEEE 754 浮点算术是一个封闭系统:
每个算术运算都会产生一个结果，无论该结果在数学上是否符合预期;
而异常运算会引发信号，要么设置一个标志 (flag) 并继续执行，要么发生中断，将控制权交给中断处理程序.
各种运算的默认结果如下表所示.

Table 2.2. *IEEE arithmetic exceptions and default results.*

Exception type	Example	Default result
Invalid operation	$0/0, 0\times\infty, \sqrt{-1}$	NaN (Not a Number)
Overflow		$\pm\infty$
Divide by zero	Finite nonzero/0	$\pm\infty$
Underflow		Subnormal numbers
Inexact	Whenever $fl(x\ \text{op}\ y)\neq x\ \text{op}\ y$	Correctly rounded result

IEEE 754 标准还对**单精度扩展** (single extended) 和**双精度扩展** (double extended) 格式规定了最低要求.
其中双精度扩展格式至少包含 79 位，其中尾数字段至少占 63 位，指数字段至少占 15 位，
因此其机器精度 $\text{eps}\leq 2^{-64}\approx 5.42\times 10^{-20}$，数值范围至少可达 $\pm[10^{-4932},10^{4932}]$.
引入扩展精度格式的目的是为了使双精度结果能够更可靠地计算 (避免中间结果的上溢和下溢)，

并且更准确地计算 (减少舍入误差的影响)，使得编写高精度的基本函数计算程序更加容易.

0.2.2 编码示例

为更直观地理解 IEEE 754 浮点数，我们考虑几个具体的位模式示例.

单精度浮点数: 1 位符号位，8 位阶码字段，23 位小数字段.

- 最小的正非规格化数的位模式为:

0	0000 0000	$\underbrace{000\ 0000\ 0000\ 0000\ 0000\ 0001}_{23}$

$$= (2^{-23}) \times 2^{-126} \approx 1.40 \times 10^{-45}$$

- 最大的正非规格化数的位模式为:

0	0000 0000	$\underbrace{111\ 1111\ 1111\ 1111\ 1111\ 1111}_{23}$

$$= (1 - 2^{-23}) \times 2^{-126} \approx 1.18 \times 10^{-38}$$

- 最小的正规格化数的位模式为:

0	0000 0001	$\underbrace{000\ 0000\ 0000\ 0000\ 0000\ 0000}_{23}$

$$= 1 \times 2^{-126} \approx 1.18 \times 10^{-38}$$

- 最大的正规格化数的位模式为:

0	1111 1110	$\underbrace{111\ 1111\ 1111\ 1111\ 1111\ 1111}_{23}$

$$= (2 - 2^{-23}) \times 2^{127} \approx 3.40 \times 10^{38}$$

双精度浮点数: 1 位符号位，11 位阶码字段，52 位小数字段.

- 最小的正非规格化数的位模式为:

0	0000 0000 000	$\underbrace{0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001}_{52}$

$$= (2^{-52}) \times 2^{-1022} \approx 4.94 \times 10^{-324}$$

- 最大的正非规格化数的位模式为:

0	0000 0000 000	$\underbrace{1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111}_{52}$

$$= (1 - 2^{-52}) \times 2^{-1022} \approx 2.23 \times 10^{-308}$$

- 最小的正规格化数的位模式为:

0	0000 0000 001	$\underbrace{0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000}_{52}$

$$= 1 \times 2^{-1022} \approx 2.23 \times 10^{-308}$$

- 最大的正规格化数的位模式为:

0	1111 1111 110	$\underbrace{1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111}_{52}$

$$= (2 - 2^{-52}) \times 2^{1023} \approx 1.80 \times 10^{308}$$

考虑将十进制整数 12345 表示为 $\mathrm{IEEE}\ 754$ 浮点数.

Solution:
首先写出 12345 的二进制表示:

$$12345_{10} = 11\_0000\_0011\_1001_2.$$

- 由于它是正数，故其浮点表示的符号位为 0.
- 由于它的最高位是第 14 位，故指数为 $14 - 1 = 13$.
  对于单精度浮点数，$\mathrm{bias} = 127$，故移码为 $13_{10} + 127_{10} = 140_{10} = 1000\_1100_2$.
  对于双精度浮点数，$\mathrm{bias} = 1023$，故移码为 $13_{10} + 1023_{10} = 1034_{10} = 100\_0000\_1100_2$.
- 由于它足够大，故无论用单精度还是双精度表示，均为规格化数.
  隐去尾数中的第一个 1，便得到小数部分 $1\_0000\_0011\_1001_2$，一共 13 位.
  对于具有 23 位小数字段的单精度浮点数来说，不发生截断，
  小数字段为 $100\_0000\_1110\_0100\_0000\_0000_2$.
  对于具有 23 位小数字段的单精度浮点数来说，不发生截断，
  小数字段为 $1000\_0001\_1100\_1000\_0000\_0000\_0000\_0000\_0000\_0000\_0000\_0000\_0000_2$.

单精度浮点数表示:

0	1000 1100	100 0000 1110 0100 0000 0000

$$= \mathrm{0x4640E400}$$

双精度浮点数表示:

0	100 0000 1100	1000 0001 1100 1000 0000 0000 0000 0000 0000 0000 0000 0000 0000

$$= \mathrm{0x40C81C8000000000}$$

Python 代码验证:

```python
import struct

x = 12345.0

# 单精度 (32-bit)
single_hex = f"0x{struct.unpack('>I', struct.pack('>f', x))[0]:08X}"
print("single precision hex:", single_hex)

# 双精度 (64-bit)
double_hex = f"0x{struct.unpack('>Q', struct.pack('>d', x))[0]:016X}"
print("double precision hex:", double_hex)
```

运行结果:

```
single precision hex: 0x4640E400
double precision hex: 0x40C81C8000000000
```

0.2.3 舍入准则

由于位宽限制，浮点数制只能表示某些离散的数值点，
因此任何实数运算都必须舍入到这些可表示点上，从而产生舍入误差.

为直观地理解各种舍入准则，考虑将以下二进制定点数按不同舍入准则舍入到 $1/4$ 位:

	10.00011_2 $(2\frac{3}{32})$	-10.00110_2 $(-2\frac{3}{16})$	10.11100_2 $(2\frac{7}{8})$	10.10100_2 $(2\frac{5}{8})$
向下舍入	10.00_2 (2)	-10.01_2 $(-2\frac{1}{4})$	10.11_2 $(2\frac{3}{4})$	10.10_2 $(2\frac{1}{2})$
向上舍入	10.01_2 $(2\frac{1}{4})$	-10.00_2 (-2)	11.00_2 (3)	10.11_2 $(2\frac{3}{4})$
向零舍入	10.00_2 (2)	-10.00_2 (-2)	10.11_2 $(2\frac{3}{4})$	10.10_2 $(2\frac{1}{2})$
向最接近的值舍入	10.00_2 (2)	-10.01_2 $(-2\frac{1}{4})$	–	–
向偶数舍入	10.00_2 (2)	-10.00_2 (-2)	11.00_2 (3)	10.10_2 $(2\frac{1}{2})$
银行家舍入	10.00_2 (2)	-10.01_2 $(-2\frac{1}{4})$	11.00_2 (3)	10.10_2 $(2\frac{1}{2})$

IEEE 754 标准默认采用**银行家舍入法** (banker's rounding)，即 "round-to-nearest, ties-to-even".
具体来说，给定真实值 $x \in \mathbb{R}$，
先尝试 **"向最接近的值舍入"** (round-to-nearest) 将 x 舍入到与之距离最近的可表示点上.
若左右两侧的可表示点与 x 的距离相同 (即出现 "平局")，
则 **"向偶数舍入"** (round-to-even)，即选择最低有效位为 0 的那个可表示点，以尽可能规避统计偏差.
一个经典的例子是将无限循环小数 $3/7$ 表示为 IEEE 754 浮点数，参见 **Homework 01 Problem 07**.

0.2.4 保护位

在过去，并非所有机器的浮点运算的舍入误差都符合 Wilkinson 模型.
最常见的原因是某些机器在执行浮点减法时没有使用**保护位** (guard digit).
保护位的作用可以通过一个简单的例子来说明:

$$
\begin{array}{ll}
2^1 \times 0.100 - & \\
2^0 \times 0.111 & \longrightarrow \quad 2^1 \times 0.100 - \\
& \qquad\quad 2^1 \times 0.0111 \\
\hline
& \qquad\quad 2^1 \times 0.0001 = 2^{-2} \times 0.100
\end{array}
$$

$$
\begin{array}{ll}
2^1 \times 0.100 - & \\
2^0 \times 0.111 & \longrightarrow \quad 2^1 \times 0.100 - \\
& \qquad\quad 2^1 \times 0.011 \quad \text{(last digit dropped)} \\
\hline
& \qquad\quad 2^1 \times 0.001 = 2^{-1} \times 0.100
\end{array}
$$

而 IEEE 754 标准引入了保护位 (同时采用银行家舍入法)，
因此其舍入误差 (在没有上溢或渐近下溢的情况下) 总是符合 Wilkinson 模型.

事实上，IEEE 754 标准还有很多比 Wilkinson 模型更强的性质.
例如，对于任意给定的浮点数 x，将其乘以 -1 或 2、与 0 做加减运算时，均不会产生舍入误差.
此外，由于保护位的存在，只要浮点数 x, y 足够接近，那么浮点减法 $x - y$ 不会产生舍入误差.
(Sterbenz 定理, Accuracy and Stability of Numerical Algorithms 定理 2.5)
在 IEEE 754 标准下，若浮点数 x, y 满足 $y/2 \le x \le 2y$，则 $\mathrm{fl}(x - y) = x - y$.

Sterbenz 定理在证明某些特殊算法的正确性时起着至关重要的作用.
一个很好的例子就是 **Heron 公式**:

$$S = \sqrt{m(m-a)(m-b)(m-c)}, \quad m = \frac{a+b+c}{2},$$

其中 a, b, c 为三角形的边长 (假设它们均是浮点数)，m 为半周长，S 为三角形的面积.
对于针状三角形，直接使用 Heron 公式计算时往往精度不高:
当 $a \approx b + c$ 时，半周长 $m \approx a$，因此计算 $m - a$ 时会发生严重的相消，对最终结果影响很大.

Kahan 提出的解决方法是，将 a, b, c 重新命名，使得 $a \geq b \geq c$，然后按以下形式计算面积:

$$S = \frac{1}{4}\sqrt{(a+(b+c))(c+(a-b))(c-(a-b))(a+(b-c))}.$$

- 由于 a 不会超过 b 的两倍，故根据 Sterbenz 定理可知 $a - b$ 是准确的 (没有舍入误差).
  注意到 c 和 $a - b$ 都是浮点数，因此 $c - (a - b)$ 的舍入误差符合 Wilkinson 模型，是安全的.
- 当 $b \approx c$ 时，计算 $b - c$ 时会发生相消，但对因式 $a + (b - c)$ 的影响较小.

事实上，改写后的公式的相对舍入误差满足:

$$\frac{|S_{\text{computed}} - S_{\text{exact}}|}{|S_{\text{exact}}|} \leq Cu,$$

其中 C 是一个不大的常数，u 是单位舍入误差，参见 Goldberg 1991 的 Theorem 3.

0.3 程序库

0.3.1 BLAS

基础线性代数子程序 (Basic Linear Algebra Subprograms, BLAS) 是一组用于矩阵和向量运算的 Fortran 原语.
它们分为三个层级，涵盖了线性代数中所有常见的运算 (参见):

- BLAS-1: 向量运算
  例如内积 $x^{\mathrm{T}}y$ (`xdot`)、数乘 $x \leftarrow \alpha x$ (`xSCAL`)、加法 $y \leftarrow \alpha x + y$ (`xAXPY`)、交换 $x \leftrightarrow y$ (`xSWAP`) 等.
- BLAS-2: 矩阵–向量运算
  例如 $y \leftarrow \alpha Ax + \beta y$ (`xGEMV`)、秩 1 更新 $A \leftarrow A + \alpha xy^{\mathrm{T}}$ (`xGER`)、三角方程求解 $x \leftarrow T^{-1}x$ (`xTRSV`) 等.
- BLAS-3: 矩阵运算
  例如 $C \leftarrow \alpha AB + \beta C$ (`xGEMM`)、三角方程求解 $A \leftarrow \alpha T^{-1}A$ (`xTRSM`) 等.

典型运算	运算量 f	数据传输次数 t	平均每次数据传输可做的运算量 f/t
$y \leftarrow \alpha x + y$	$2n$	$3n + 1$	$2/3$
$y \leftarrow \alpha Ax + \beta y$	$2n^2 + n$	$n^2 + 3n$	2
$C \leftarrow \alpha AB + \beta C$	$2n^3 + n^2$	$4n^2$	$n/2$

基于计算机组成与体系机构的知识，我们知道在编写程序时应当尽量减少内存和磁盘的访问次数.
显然 BLAS-3 的效率是最高的，取一次数据能进行更多的运算，因此编程时应尽可能多地使用 BLAS-3.

每一组 BLAS 仅定义了每个子程序的参数以及子程序必须完成的功能，但不规定具体的实现方式.
因此实现者在实现细节上具有一定的自由度.
然而，实现必须是数值稳定的，并且模型实现中提供了用于测试数值稳定性的代码.
BLAS 接口快速查询: BLAS Quick Reference Guide
BLAS 技术论坛网站为: BLAS Techical Forum.

0.3.2 LAPACK

LAPACK 于 1992 年正式发布.
它是一个可移植的 Fortran 77 子程序库，专门设计以充分发挥 BLAS-3 的计算优势，
用于解决数值线性代数中的四类基本问题: 线性方程组求解、线性最小二乘问题、特征值问题和奇异值问题.
在进行严肃的数值线性代数研究时，LAPACK 常被视为基础工具库.
关于其安装、编译与调用方式，可参见 **Homework 02 Problem 07**.

The End