类似这样的技巧还有很多，上面提到的只是冰山一角。为了向读者展示数值计算内行的“门道”，我们列出了参考文献 [8]。特征值计算作为数值计算的另外一大问题，本书后文中也会着重加以介绍。

本章中只是从线性代数的立场出发讨论一些基本的矩阵算法。但是要上手处理实际问题，仅凭这点内容可能还是不够的。要想学习真正的数值计算，还是要参考专业书籍，本章仅仅是起到了敲门砖的作用。

### 3.1.2 关于本书中的程序

为了帮助读者理解，本书提供了用于矩阵的加减乘运算以及 LU 分解的程序。至于如何获取程序源代码，请参考前言 (e)。本节中不会给出完整的代码，只是精选了需要特别说明的部分。

我们给出的代码，只能算是“学习用”版本。关于代码，还请注意以下几点。

- 相比效率和容错性，我们优先采用简单、可读性强的代码，所以我们的程序并不一定适合用于实战场合
- 程序采用的是 Ruby 语言，但是我们回避了语言特有的方便的用法，而是采用了更一般化的、类似于传统程序设计语言的写法。这是因为本书的目的并不是讲解某种特定语言的用法。因此，使用其他语言的读者请不要担心，同样可以像读伪代码一样无障碍阅读（参考前言 (e)）。但是，一定请读者注意，不要误认为 Ruby 语言很弱哦！
- 在 Ruby 中，“#”之后表示注释

另外，当需要“实战用”的程序时，最好还是使用现成的软件包。实际上对于非数值分析专业的读者来讲，要想亲自实现这些算法压力也确实太大了。为此，这里推荐著名的线性代数的数值计算包 LAPACK。最后补充说明一点，在Ruby的标准库中也包含了专门处理矩阵计算问题的类库 —— matrix.rb。

## 3.2 热身：加减乘运算

虽说本章的主题是 LU 分解，但作为热身，我们先来试试加减乘的计算。另外，对于代码中用到的 1 维和 2 维数组，我们默认是已有定义的[①]。

向量和矩阵的运算，基本上全部采用 for 循环对每个分量（元素）进行操作。例如，向量求和的程序如下所示。

```
# 和（将向量 b 加到向量 a 上：a←a+b）
def vector_add(a, b)          #  定义函数（到 end 为止）
  a_dim = vector_size(a)      #  得到各向量的维数
```

① 因为根据语言的不同，声明和定义的方式大有不同，要是每种语言的情况都详加说明也没什么好处吧。