Research group
Discrete Geometries of Mathematics and Physics
www.mathischeap.com/dgmp

Discrete Geometries of Mathematics and Physics

线上学习与测验指导书

线性代数

Python 驱动的线性代数基础

Yi Zhang (张仪)
: www.mathischeap.com
@: zhangyi_aero@hotmail.com
: https://github.com/mathischeap

1 介绍

线性代数是一门极具实践意义的学科。它不同于理论数学中很大一部分学科主要依赖“纸笔”来完成推演，日常工作中的线性代数问题大多时候是借助计算机来完成的。因此，学会从课本走向计算机本该是线性代数这门课程的核心内容。事实上，我们使用的教材也是这样编排的：可以看到，它的最后一章是介绍面向应用的具体问题。但是由于各种原因，这一部分在实际教学中被忽略了。故此，我想借助线上学习模块来完成这个内容。

2 为什么是 Python?

当前流行的编程语言高达几百种，从中选出适合我们的、功能强大且短时间内不会被淘汰的编程语言并非易事。不过，用别人的选择作为参考永远都不会是一件坏事。图1是一个全球编程语言流行程度的排名。从图中我们可以看到，排名第一的 Python 的流行程度在大幅领先的基础上又取得了巨大的提升（7%），达到了惊人的 21.90%。这与 Python 的开源、易用性、功能多样性、跨平台应用能力有密不可分的关系。可以这么说，如果你在二十一世纪只能学习一门编程语言，那么 Python 一定是你的第一选择。因为，对于某个领域，可能存在一门比 Python 好的语言。但是，在其他很多领域，Python 一定会比它更好。或者说，你能够认为 Python 样样不精通，但它样样都行、是一个能力均衡的全能型选手。

3 线上自学内容

注意：你可以根据你的 感兴趣程度 以及 课后时间富余程度 自由决定学习的深度和难度。对 Python 更好的理解有助于（但不决定性影响）你完成之后的练习与测验。

Oct 2024	Oct 2023	Change	Programming Language	Ratings	Change
1	1		Python	21.90%	+7.08%
2	3	↑	C++	11.60%	+0.93%
3	4	↑	Java	10.51%	+1.59%
4	2	↓	C	8.38%	-3.70%
5	5		C#	5.62%	-2.09%
6	6		JavaScript	3.54%	+0.64%
7	7		Visual Basic	2.35%	+0.22%
8	11	↑	Go	2.02%	+0.65%
9	16	⇈	Fortran	1.80%	+0.78%
10	13	↑	Delphi/Object Pascal	1.68%	+0.38%
11	9	↓	SQL	1.64%	-0.15%
12	14	↑	MATLAB	1.48%	+0.22%

图 1: 全球编程语言流行程度实时排名。

3.1 学习内容一：Python 的经典网络教材

Learn Python The Hard Way 是一本 Python 学习的经典教材。很多人都是通过这本书直接完成从入门到精通的整个过程。这本书已经有中文在线版本，各位同学可以通过访问
`https://www.bookstack.cn/read/LearnPython3TheHardWay/spilt.1.learn-py3.md`
在线阅读。

3.2 学习内容二：Bilibili 上 Python 的公开课程

视频网站 bilibili 上拥有很多免费的优质学习内容。你可以通过搜索”Python” 轻松找到它们，如图 2。

图 2: 一些 bilibili 上免费的 Python 课程。

3.3 学习内容三：MOOC(慕课) 上的免费 Python 课程

MOOC（慕课）上拥有海量的免费课程。我们可以通过访问`https://www.mooc.org/`找到全球最优秀的大学、企业提供的 Python 课程，如图 3。

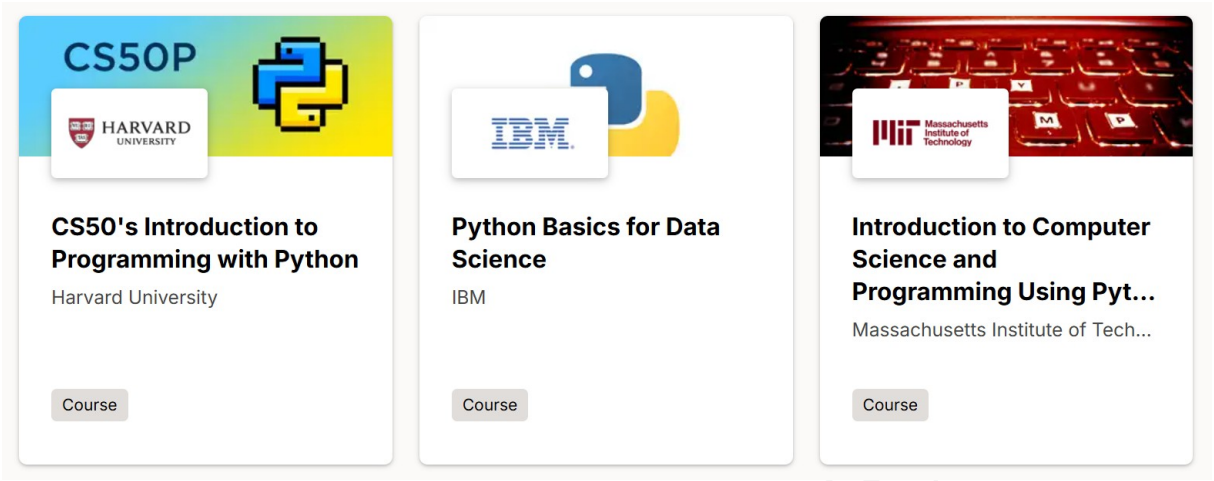

图 3: MOOC 上来自哈佛、IBM、麻省理工的 Python 课程。

3.4 学习内容四：专业书籍

使用 Python 处理线性代数问题本身就是一个重要的课题。当下已经有很多优秀的书籍介绍 Python 在线性代数领域的应用方法，如图 4.

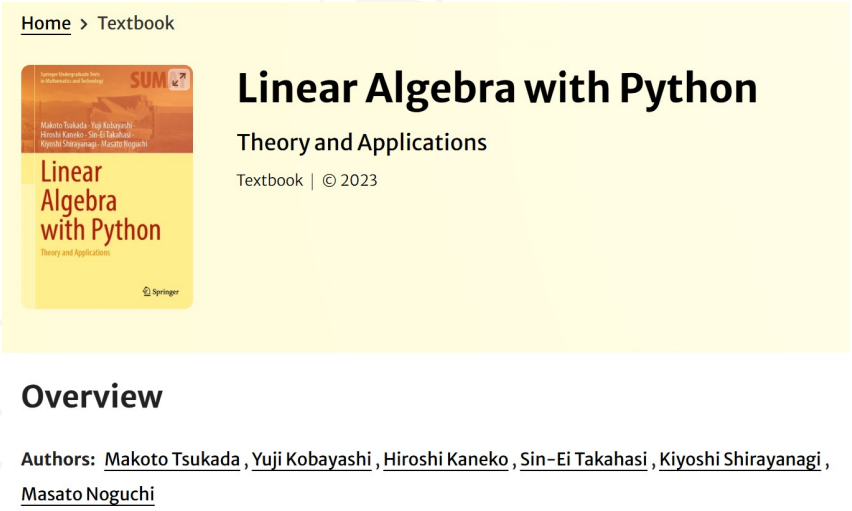

图 4: Springer Nature 上一本关于用 Python 处理线性代数问题的专业书籍`https://link.springer.com/book/10.1007/978-981-99-2951-1`。

另外，比如，伯克利大学提供的一本优秀的开源书籍Python Programming And Numerical Methods: A Guide For Engineers And Scientists对大学生而言也是入门 Python 的不二选择，见图 5。

4 练习

在这一节，我们将一起学习如何使用 Python 来完成各种线性代数计算。请跟随我们的教程一起完成所有的练习。

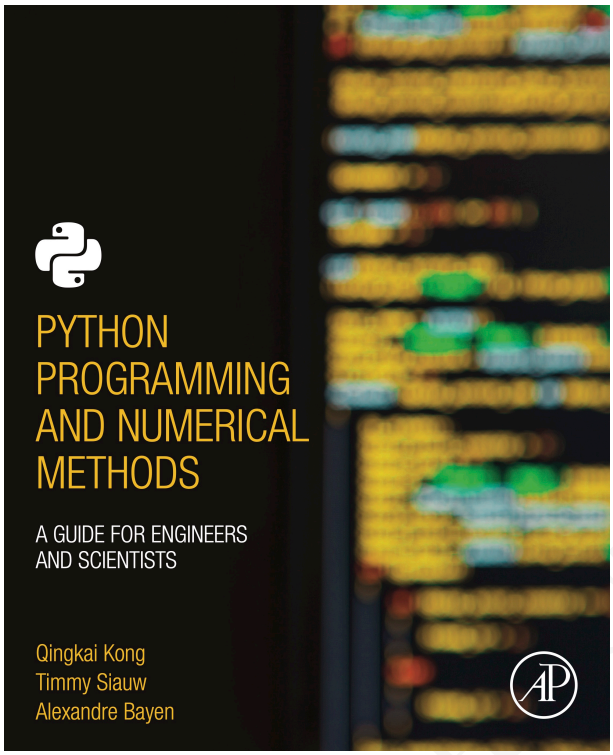

图 5: 伯克利大学的开源书籍Python Programming And Numerical Methods: A Guide For Engineers And Scientists。

4.1 准备好 Python 环境

如果你已经配置好了自己的 Python 编程环境（如 VSCode、PyCharm、Atom、Spyder 等），你可以跳过本节。否者，我推荐使用在线 Python 编译器：

`https://www.programiz.com/python-programming/online-compiler/`

点击上面的链接将打开一个网页，即 Programiz-Python 界面。界面上有左右两个窗口。左边的窗口用于输入：你可以输入 Python 代码。右端的窗口用于输出；显示代码运行信息、运行结果等。见图 6。本文档中剩余的内容都将基于 Programiz-Python 进行讲解。

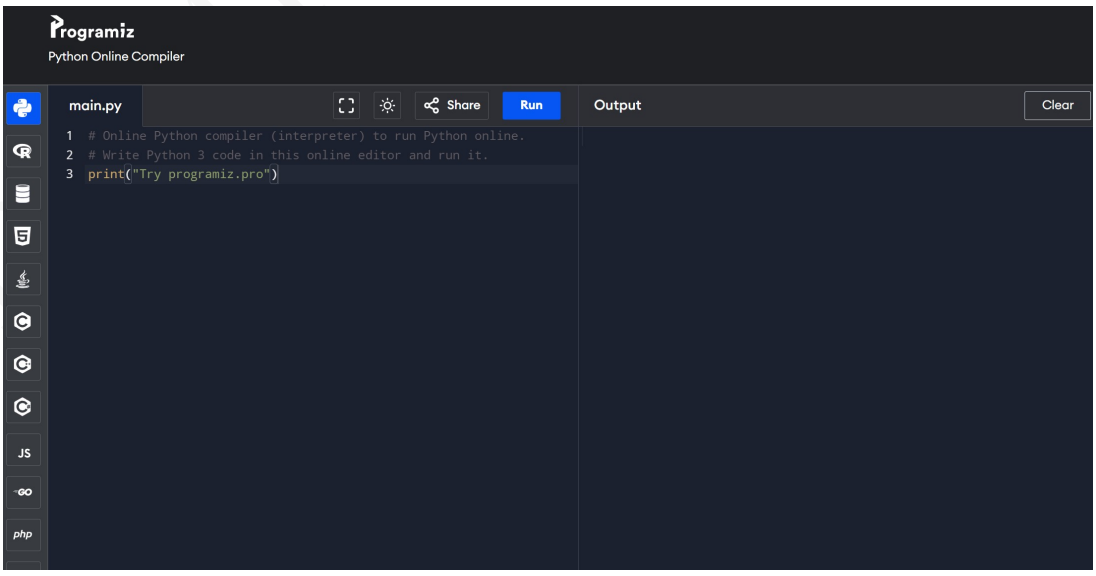

图 6: Programiz-Python 界面。左右两个窗口分为输入、输出窗口。

4.2 矩阵输入

我们将使用 numpy 库作为我们的运算工具。所以，在开始进行任何操作之前，我们需要将 numpy 库导入到我们的计算环境中来。所以，我们需要在左侧输入窗口输入以下内容：

```
import numpy as np
print(np)
```

我们可以通过**点击输入窗口右上角的蓝色 Run 按钮以运行代码**。运行后，我们可以在右侧输出窗口看到以下信息：

```
<module 'numpy' from '/usr/local/lib/python3.11/site-packages/numpy/__init__.py'>
```

这说明 numpy 库已经被正常导入。注意，我们使用了 print（即“打印”）命令来向输出窗口打印信息。我们之后将反复使用这个命令来观察我们的计算结果。

由于 numpy 库已经被我们成功导入，并且被重新命名成了 np。我们接下来可以通过使用 np 命令来调用 numpy 库中所有功能。比如，我们在输入窗口继续添加如下代码：

```
A = np.array([
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9],
  [-1, 0,-2]
])

E = np.array([
  [1, 0, 0],
  [0, 1, 0],
  [0, 0, 1]
])

B = np.array([
  [-1, 0, 3, 2],
  [0, 4, 5, -2]
])

C = np.array([
  [1, 2, -1, -3],
  [-2, 3, -6, -1]
])

print(A)
print(E)
print(B)
print(C)
```

点击 Run 按钮运行代码后我们可以看到我们成功定义了四个矩阵 A, E, B, C。它们分别是一个 4×3 的矩阵，3 阶单位矩阵和两个 2×4 的矩阵。

同理，我们可以定义行向量和列向量，比如:

```
a = np.array([
  [2],
  [0],
  [1]
])

b = np.array([
  [-2, -1, 0, 3]
])

print(a)
print(b)
```

同样，运行代码后我们可以观察到我们定义了一个 3 维的列向量和一个 4 维的行向量。

4.3 运算

在成功定义好矩阵的基础上，对矩阵的运算才是关键。这一节我们介绍如何使用 numpy 来进行基本的矩阵运算和求解。

矩阵的加减法 我们可以使用 $+-$ 算符执行矩阵的加减法，比如：

```
P = B + C
print(P)
```

```
Q = B - C
print(Q)
```

运行后，我们将看到我们成功计算了矩阵 B 和矩阵 C 的加减法。但是如果我们输入

```
P = B + A
print(P)
Q = B - A
print(Q)
```

运行后我们将发现输出窗口报错，我们无法得到结果。原因也很简单：矩阵的加减法要求两个矩阵为同型矩阵，但矩阵 A 为 4×3 的矩阵，矩阵 B 为 2×4 的矩阵，它们不同型故此无法进行加减运算。

同理，负矩阵可以通过 $-$ 算符来运算，如：

```
D = - B
```

矩阵的数乘 矩阵的数乘使用的算符为 $*$，比如

```
k = 0.5
F = k * A
print(F)
```

通过这段代码，我们首先定义了一个常数 $k=0.5$，然后我们计算了它与矩阵 A 的数乘。注意，在书面书写时，我们不需要书写数乘符号 $*$，但是在 Python 里，我们必须输入 $*$。这是因为如果不加上 $*$，程序无法得知你是在计算矩阵的数乘，还是在调用一个名为 kA 的变量。

当然，我们可以直接运算矩阵的数乘而不需定义常数变量，如：

```
F = 0.5 * A
print(F)
```

运行后我们可以发现我们得到相同的结果。

矩阵的乘法 我们通过使用 @ 算符来计算矩阵的乘法，比如：

```
G = A @ E
R = B @ A
print(G)
print(R)
```

由于 A 是 4×3 的矩阵，E 是 3 阶单位矩阵，B 是 2×4 的矩阵，上述运算满足矩阵的乘法的运算法则的要求，即前一个矩阵的列数必须等于后一个矩阵的行数，故此可以运行。运行程序可以看到计算结果。同矩阵的数乘类似，我们不能省略 @，否则，Python 将无从得知你是否在进行矩阵的乘法运算。

如果你尝试计算不满足矩阵乘法要求的情况，你将看到报错信息。

矩阵的转置 矩阵的转置可通过调用 T 属性运算。比如

```
AT = A.T
BT = B.T
print(AT)
print(BT)
```

我们看到，我们计算了矩阵 A，B 的转置，并将它们命名成 AT 和 BT。我们现在可以验证性质 $(\boldsymbol{BA})^{\mathsf{T}}=\boldsymbol{A}^{\mathsf{T}}\boldsymbol{B}^{\mathsf{T}}$：

```
Y = (B @ A).T
U = AT @ BT
print(Y)
print(U)
```

可以看到，上述命令输出了相同的 Y 和 U 矩阵。

矩阵的行列式 numpy 里计算矩阵行列式的函数是 np.linalg.det。回忆行列式的英文名称为 determinant。这里使用了它的前三个字母 det 为函数命名。如

```
d1 = np.linalg.det(E)
print(d1)
```

我们一定得到 1 因为单位矩阵的行列式一定等于 1。那么，如果我们进行如下运算

```
d2 = np.linalg.det(A)
```

程序将报错，因为矩阵 A 甚至都不是方阵，故此它没有行列式。但是如果计算

```
d3 = np.linalg.det(A @ A.T)
print(d3)
```

我们可以得到一个结果，因为 A@A.T 是一个 4 阶方阵，存在行列式。

逆矩阵 我们可以使用 np.linalg.inv 函数来求矩阵的逆矩阵。如

```
A1 = np.array([
  [1, 2, -1],
  [3, 3, 4],
  [-2, 5, 1]
])
invA1 = np.linalg.inv(A1)
print(invA1)
```

我们就能够得到矩阵 A1 的逆矩阵。

矩阵的秩 我们可以使用 np.linalg.matrix_rank 函数来计算矩阵的秩。比如：

```
B1 = np.array([
  [4, 1, 2, -1],
  [-1, 3, 3, 4],
  [-2, 5, 1, 0],
  [-3, 1, 1, 2]
])
r = np.linalg.matrix_rank(B1)
print(r)
```

我们能计算出该矩阵的秩为 4。

线性方程组求解 我们可以通过 np.linalg.solve 函数来求解线性方程组 Ax=b。比如

```
A = np.array([
  [4, 3, 1, 2, -1],
  [-1, 3, 0, 3, 4],
  [-2, 5, 1,  1, 0],
  [-3, 1, 1, 2, 2],
  [1, 2, 2, -1, -1]
])
b = np.array([
  [1],
  [2],
  [3],
  [-1],
  [0]
])
x = np.linalg.solve(A, b)
print(x)
```

这样，我们求解了一个五元一次方程组

$$\begin{cases} 4x_1 + 3x_2 + x_3 + 2x_4 - x_5 = 1 \\ -x_1 + 3x_2 + 3x_4 + 4x_5 = 2 \\ -2x_1 + 5x_2 + x_3 + x_4 = 3 \\ -3x_1 + x_2 + x_3 + 2x_4 + 2x_5 = -1 \\ x_1 + 2x_2 + 2x_3 - x_4 - x_5 = 0 \end{cases},$$

注意，np.linalg.solve 函数只能求解系数矩阵 $\boldsymbol{A}$ 为方阵的线性方程组。此时，要求矩阵 $\boldsymbol{A}$ 可逆，即它的秩等于它的阶数。

5 测验

测验的形式为线上测试，你将在规定的时间内使用 Python 计算特定的线性代数问题。测试的范围不会超出上一节中结果过的内容。请留意班级公告，在收到测试开始的通知后按时完成测试。