Discrete Geometries of Mathematics and Physics

Online Learning and Testing

Linear Algebra

Linear Algebra with Python

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

1 Introduction

Linear algebra is a very practical subject. Unlike most mathematical subjects that require pen and paper for derivations, linear algebra is usually performed with computers. Thus, knowing how to bridge it to computers from books should be a key output of a fine linear algebra lecture. However, it is a reality that nowadays universities are unable to design practical content for that many students in the course of linear algebra. Meanwhile, many lecturers, unfortunately, have forgotten that linear algebra is, in fact, not about achieving a good grade in the final written examination. In order to make a small change, I made this online learning and testing to let the young college students in my courses have a chance to fill a little of the gap.

2 Why Python?

There are hundreds of programming languages used nowadays. It is not easy to pick a proper one. Is it powerful enough? Will it be obsolete soon? And there are many other factors to be considered. Before making the choice, it is never a bad choice to have a look at others' selections. In Fig. 1, you can see a recent global programming language ranking. It shows that Python is taking the lead and is even growing it. This is simply because of its accessibility and compatibility. Let's say, if you can only learn one programming language in the twenty-first century, it must be Python. Python may not be the most powerful language in a particular technical aspect. But it does okay in all aspects. In other words, you can probably find a better language in each field. But you will always have Python there as an okay alternative.

Therefore, we use Python as the tool for this online learning and testing. But please do not worry. It is not difficult; you will not need to learn Python in a systematic manner. All aspects needed for this online learning and testing will be demonstrated in Section 4.

Let's put it another way. If you just wish to accomplish this task, you can do it in a very short time by studying this document solely. But if you find it (learning Python) interesting, it will open a new gate for you, and you could spend a lot of time on Python for the rest of your life, like me. Of course, a better understanding of Python will help you with this online learning and testing. And, furthermore, it will benefit you a lot in your career.

Oct 2025	Oct 2024	Change	Programming Language	Ratings	Change
1	1		Python	24.45%	+2.55%
2	4	↑	C	9.29%	+0.91%
3	2	↓	C++	8.84%	-2.77%
4	3	↓	Java	8.35%	-2.15%
5	5		C#	6.94%	+1.32%
6	6		JavaScript	3.41%	-0.13%
7	7		Visual Basic	3.22%	+0.87%
8	8		Go	1.92%	-0.10%
9	10	↑	Delphi/Object Pascal	1.86%	+0.19%

Fig. 1: A global ranking of programming languages.

3 Online learning

As aforementioned, you can decide for yourself how much time should be spent on this section. If you are in a hurry, you can jump to Section 4 after you have obtained an understanding of the fundamental framework of Python.

3.1 Learning content #1: A classic textbook on Python

Learn Python the Hard Way is a classic textbook for Python. Lots of people, including me, got their first experience of Python through this book. It was an open-source book, but now become a commercial one (this, of course, is a pity), see `https://learnpythonthehardway.org/`. I think you will not regret it if you pay for it, although there are open-source alternatives anyway.

3.2 Learning content #2: Free courses on MOOC

There are numerous free courses for Python on MOOC (`https://www.mooc.org/`) from the best universities and enterprises. For example, see Fig. 2。

3.3 Learning content #3: More

There are books particular on using Python to take care of linear algebraic, for example, see Fig. 3. In addition, Berkeley has a good open-source book which guides young engineers to Python, see Python Programming And Numerical Methods: A Guide For Engineers And Scientists and Fig. 4.

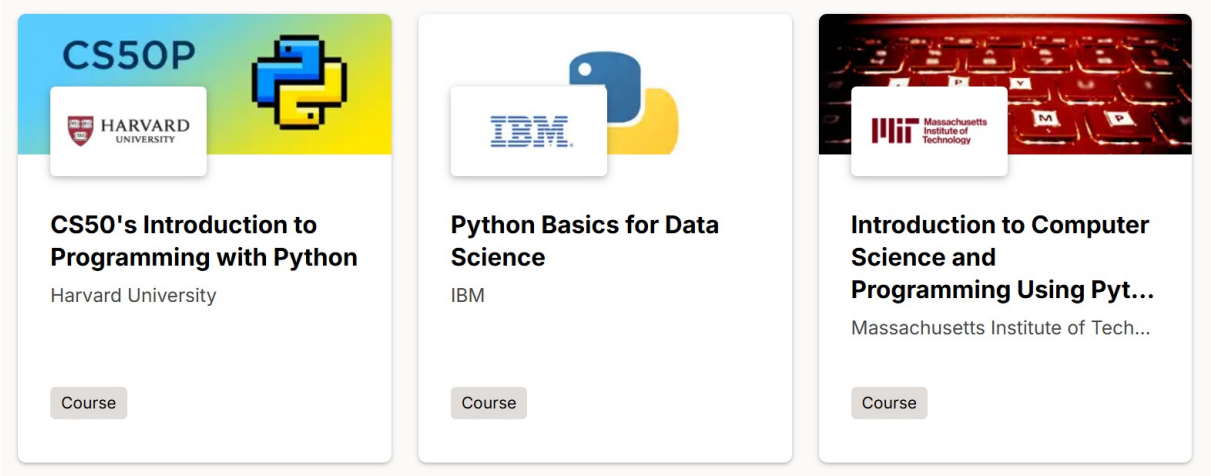

Fig. 2: Some free courses on MOOC(`https://www.mooc.org/`) from Harvard, IBM and MIT.

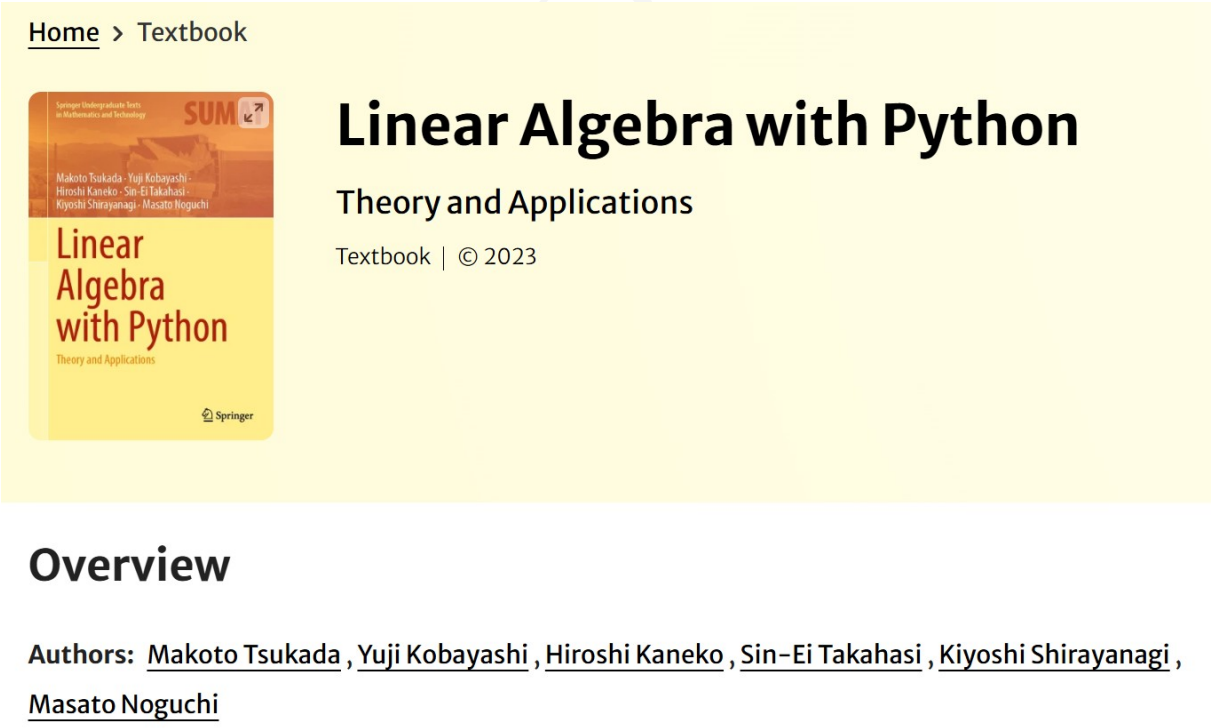

Fig. 3: A book about how to use Python for linear algebraic problems on Springer Nature, `https://link.springer.com/book/10.1007/978-981-99-2951-1`.

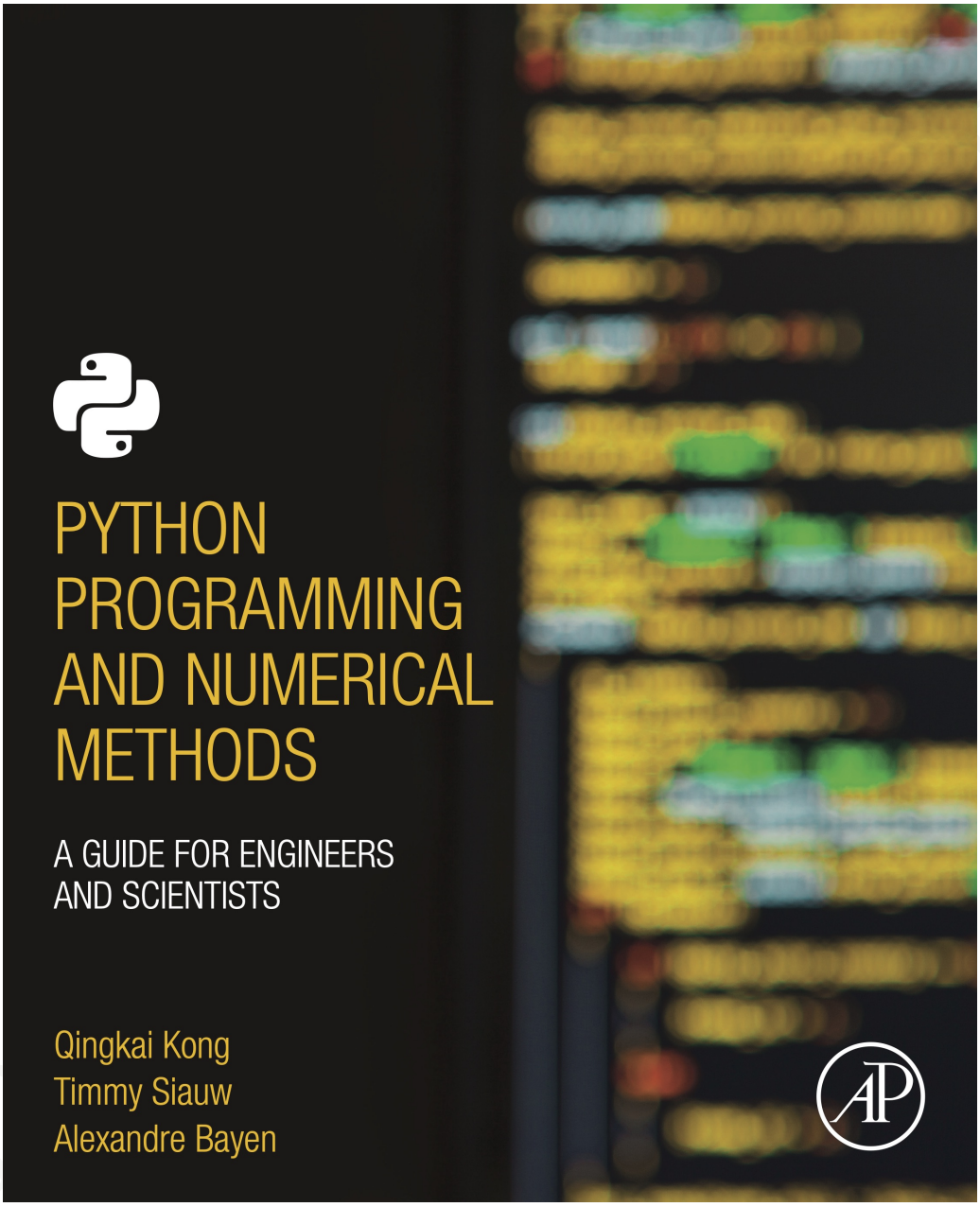

Fig. 4: An open-source book from Berkeley, Python Programming And Numerical Methods: A Guide For Engineers And Scientists, on Python.

4 Practices

In this section, we demonstrate how to use Python to accomplish linear algebraic computations, including mainly matrix computations.

4.1 Get your Python environment ready

If you have prepared your Python environment, including the kernel and an editor (like VSCode, PyCharm, Atom, Spyder, etc.), you can skip this paragraph. Otherwise, I suggest you click on the link `https://www.programiz.com/python-programming/online-compiler/` which will lead you to the Programiz-Python page. Note that it will ask you whether you agree with their policies and regulations at your first visit. You will see an interface of two windows. The left window is where you input your code, and the right window shows the output. See Fig. 5. The rest of this document will be based on Programiz-Python.

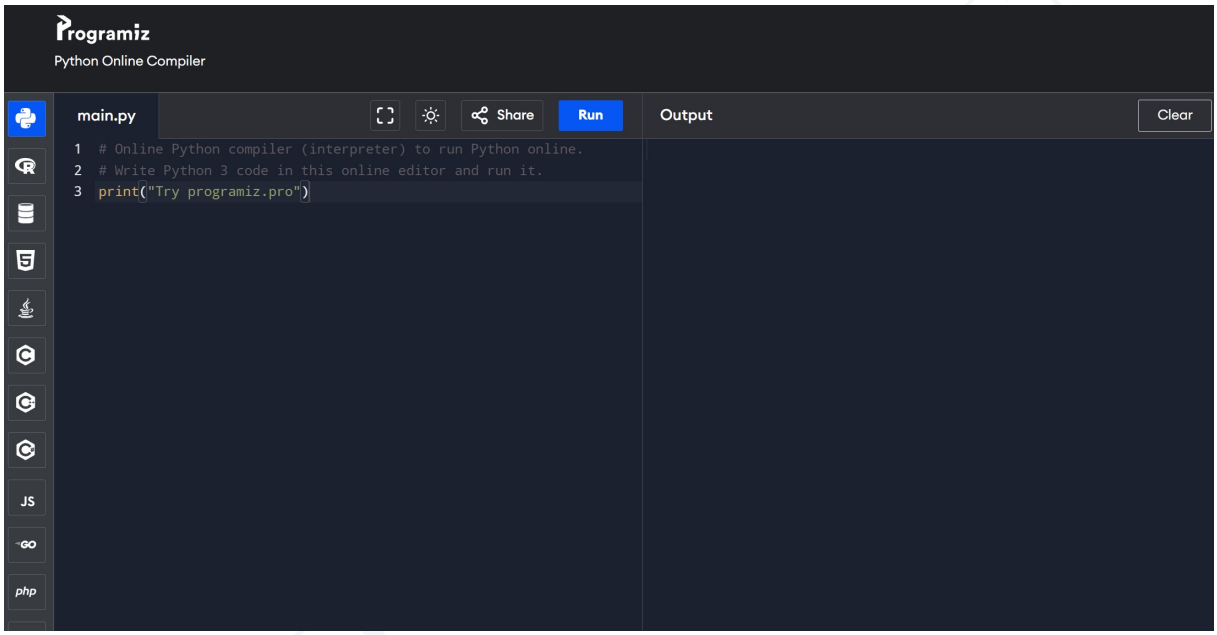

Fig. 5: The interface of Programiz-Python. The left window is for inputing and the right one is for outputting.

4.2 Inputing matrices

We will use the `numpy` package as the tool. Thus, before any computations, we should import `numpy`. To do this, we type the following code in the left window,

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

Then we click on the "Run" icon, which will execute the code we have just input. After a short time, we should see the following message (or a similar message) in the right window,

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

This shows we have correctly imported the `numpy` package to the current computational environment and have renamed it as a shortcut, `np`. Also, notice that we used another function `print`. Yep, it does what its name says; it prints the information of something in the right window. Here, we used

it to print the information of the `np` package. We will repeatedly use the `print` function to show our results.

Since we have imported `numpy` and have renamed it as `np`, we can call any function of `numpy` by calling it from `np`. For example, the `array` function of `numpy` is usually used for defining a matrix. So we can call it now through `np.array`. For example, we can define matrices as follows.

```
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)
```

If we run it by clicking on the "Run" icon, we can see that we have successfully defined four matrices, `A, E, B, and C`. They are a 4×3 matrix, a 3×3 unit matrix, and two 4×2 matrices, respectively.

Similarly, we can define column and row vectors as follows,

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

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

print(a)
print(b)
```

where we can see in the output window that `a` is a 3-dimensional column vector and `b` is a 4-dimensional row vector.

4.3 Operations

Upon the defined matrices, we can perform operations on them. This actually is the key of this document. Now, we study how to do basic operations of matrices using `numpy`.

Addition and subtraction We can use $+-$ operators to do the addition and subtraction between matrices. For example,

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

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

If we run the code, we can see that we have successfully performed the addition and subtraction between matrices `B` and `C`. Alternatively, if we input

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

We will see an error raised in the output window. The reason is simple: the addition or subtraction requires that the matrices must be of the same shape. But `A` is 4×3 and is of a different shape from that of `B`.

Likely, the negative matrix can be obtained using the $-$ operator, for example,

```
D = - B
print(D)
```

Scalar multiplication The operator for the scalar multiplication between a number and a matrix is $*$. For example,

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

Here, we first defined a scalar $k = 0.5$. And then we compute the multiplication between k and the matrix `A`. Note that, in a written format, we do not need to explicitly write an operator for the scalar multiplication. So the multiplication between k and the matrix `A` is just

$$k\boldsymbol{A}.$$

But, in Python, we cannot skip the operator $*$. This is because, if we do not type an $*$ inbetween, the program does not know we are computing the multiplication or are just referring to a variable named `kA`.

Of course, you can directly do the multiplication without defining a scalar, like

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

We can see the same result if you hit the run button.

Multiplication between matrices We can use the @ operator to do the multiplication between matrices. For example

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

Note that `A` is 4×3, `E` is 3×3, and `B` is 2×4. Therefore, these multiplications are legal; we will see that Python gives the correct results. Same as the scalar multiplication, where we cannot omit the $*$ operator, here we cannot omit the @ operator. The reason is the same as well.

Now, please try

```
F = A @ B
```

What does Python give? Could you explain why?

Transpose The transpose of a matrix can be obtained by calling the `T` property of a matrix, i.e.,

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

We can see from the results that we correctly obtain the transposes of matrices `A` and `B`. And we have named them `AT` and `BT`. We can now check the proposition $(\boldsymbol{BA})^\mathsf{T} = \boldsymbol{A}^\mathsf{T}\boldsymbol{B}^\mathsf{T}$ by doing

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

We see `Y` and `U` are the same.

Inner product between vectors We have introduced how to input vectors. Assume that we have two column vectors:

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

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

Now, we can use the transpose and the matrix multiplication to compute the inner product between them, like

```
ab = a.T @ b
print(ab)
```

It gives a 1×1 matrix, [[7]], which, as we explained in the lecture, can be regarded as a number, 7. It is the result of the inner product between the two column vectors.

If you find it inconvenient to type a column vector, you can use the transpose operator to transpose a row vector. For example,

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

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

print(a == c.T)
```

We can see `a` is the same as `c.T` since each component is equal (`True` as the result shows).

Determinant of a matrix In `numpy`, the function used to compute the determinant of a matrix is `np.linalg.det` (it takes the first three letters of the word determinant),

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

We can see that the result is 1, as the determinant of a unit matrix must be 1. But how about doing

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

The code will give an error since the matrix `A` is not a square matrix; it has no determinant. Alternatively, if we do

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

we can obtain a result since `A@A.T` is a 4×4 matrix; a squared matrix has its determinant.

Inverse matrix We can use the function `np.linalg.inv` to invert a matrix like

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

Here we computed the inverse matrix of `A1` and named it `invA1`. Recall how difficult it is to compute the inverse matrix, as we explained in the lectures. With Python, it becomes trivial. But keep in mind that it is always computationally expensive to invert a matrix. If the matrix is big, even Python will feel the stress. It is normal that Python may need hours, days, or even weeks to invert a matrix.

Solving a set of linear equations We can use the function `np.linalg.solve` to solve a set of linear equations. For instance,

```
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)
```

Here, we in fact solved a set of linear equations as

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

Note that `np.linalg.solve` can only solve a set of linear equations whose coefficient matrix `A` is square. In other words, it needs that the number of equations is the same as the number of unknowns. And, it requests that `A` must be invertible.

5 Testing

The online testing will be announced when about 60% of lectures have been given. You will need to compute a series of linear algebraic problems using Python in a limited time. The problems will not go beyond those practices of Section 4.

Please keep an eye on the class announcement, and accomplish it in time.

Happy coding.