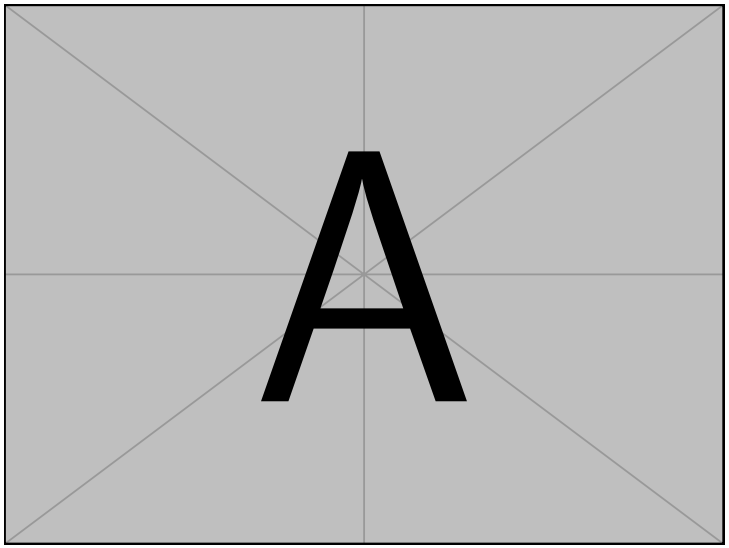
CEE384—Numerical Methods

Numerical Methods in Python

Expressing Mathematical Functions in Python



Arizona State University

Brian Chevalier

Updated: October 28, 2019

Contents

1 Introduction **1**

2 Sigma and Pi Notation **1**
2.1 Sum 1
2.2 Product 2

3 Creating a List of Values **2**
3.1 Range 2
3.2 List Comprehensions 3
3.3 Examples 3
3.3.1 The Sum of Squares 3
3.3.2 Dot Product 3

4 Building up a Matrix **4**
4.1 Tensor Product 4
4.2 Matrix Multiplication 4

References **5**

1 Introduction

This note set assumes you have a basic understanding of Python syntax including: `functions`, `lists`, and `for` loops. We will be looking at common code patterns for translating mathematical expressions into good Python code.

Using a *functional* style of programming will be a focus of this set of notes. Minimizing *state* and *mutation* of state. Variables only exist in the scope of the function.

2 Sigma and Pi Notation

You are likely familiar with Summation (also known as Sigma notation). It is a shorthand for describing the calculation of terms to be added together. Similarly the Greek capital Pi can be used to notate the product of a sequence of terms.

This section will look at how to express sums and products of terms elegantly in Python.

2.1 Sum

The simplest summation is the summation of all elements in a list. If a list of values is defined as $x = [x_0, x_1, \ldots, x_{n-1}]$ then the sum of all the elements of a list can be written as:

$$\sum_{i=0}^{i<n} x_i \tag{1}$$

where i is the iteration number, and n is the number of elements in the list.

In Python we can define our own summation function such as the one below:

```python
def sum(x):
    """
    A function that takes in a list of elements
        and returns the sum of all the elements
        in the list
    """
    total = 0          # initialize sum
    for xi in x:       # iterate over elements
        total += xi # add each item to the total
    return total
```

Creating this function allows us to encapsulate the boiler-plate of summing elements in a list. Python even has a built in function called `sum` that does just this. The following is an example of how either would be used:

```python
s = sum([0, 1, 2, 3]) # will be equal to 6
```

You should prefer using the built in `sum` function whenever computing summations. Built in functions will almost always be faster than your implementation, reduce the bugs that you write (code you don't write can't have bugs), and makes your code more expressive and clearer in purpose

2.2 Product

Product notation is very similar to summation notation. The notation has a lower and upper bound, and a term described in terms of the iteration counter. The only difference is the terms are multiplied together instead of added. The product of a list of elements can be written mathematically as:

$$\prod_{i=0}^{i<n} x_i \tag{2}$$

This can be implemented in a Python function very similar to the sum function. The product starts at 1, and each term is sequentially multiplied by the total.

```
def prod(x):
    """
    A function that takes in a list of elements
       and returns the product of all the
       elements in the list
    """
    total = 1          # initialize product
    for xi in x:       # iterate over elements
        total *= xi  # mult. by each item
    return total
```

Python does not have a way of taking the product of a list of elements, but is a feature of libraries in the standard library. `Numpy` offers the function `prod` which does this. This can be used as follows:

```
import numpy as np
p = np.prod([1, 2, 3, 4]) # will equal 24
```

You should use the `prod` function for the same reasons as using the built in `sum` function.

3 Creating a List of Values

We have looked at taking the sum and product of very simple expressions. What about more complicated sequences of terms?

3.1 Range

The easiest way to create a linearly spaced sequence of values is using the `range` function. Let's look at the case for a list of values beginning at 0, ending at 9, with a total of 10 values. This can be expressed mathematically in the following ways:

$$x = [0, 1, \ldots, 9] \tag{3}$$
$$(0 \leq x < 10) \tag{4}$$

which can be described in Python as:

```
x = range(10)
```

The `range` function will produce elements beginning at zero, up until just before 10. You can read the expression as "return a range of 10 values". Since a lower bound is omitted, it implicitly starts at zero.

Note: Python is *zero-indexed*, which means numbering starts at zero. The mathematical notation used here will match this and should make this a non-issue most of the time. There is much discourse about this topic that will not be covered here [1].

3.2 List Comprehensions

$$x_i = i^2 \tag{5}$$

With Python you can initialize an empty list, then loop over some range, and then append the required value to the list.

```
x = []
for i in range(n):
    x.append(i**2)
```

However, this is not best practice. There are many issues with this style. First: it must take at least three lines. Second: it requires *mutating* a variable, x, and adding items. This can be a very common source of bugs.

Instead we can use a very nice Python feature. With Python's built in 'List Comprehension' feature, a list such as this can also be created in one concise statement.

```
x = [ i**2 for i in range(n) ]
```

If we use list comprehensions, we can also very easily use Sigma and Pi notation within Python that will make our code look very similar to our math.

3.3 Examples

3.3.1 The Sum of Squares

$$\sum_{i=0}^{i<4} i^2 \tag{6}$$

```
x2sum = sum([i**2 for i in range(4)])
```

3.3.2 Dot Product

Combining the ideas of summation and the list comprehension, we can construct a function that takes the dot product of two lists. The dot product is defined as:

$$c = \sum_{i=0}^{i<n} a_i b_i \tag{7}$$

where a, and b are two arrays of length n, and c is the result of the dot product.

This can be implemented as a Python function:

```
def dot(a, b):
  n = len(a) # no. of elements in list
  return sum([a[i]*b[i] for i in range(n)])
```

Of course, there is a function vended by `Numpy` that you should prefer to use, demonstrated below.

```
import numpy as np
a = np.array([1, 2, 3])
b = np.array([1, 1, 1])
np.dot(a, b)
```

4 Building up a Matrix

We have seen how to take lists and produce a single value (sum and product), and how to compute a list with a list comprehension. What about generating a 2D array (a.k.a. matrix).

4.1 Tensor Product

The tensor product (also known as the outer product), takes two arrays and produces a matrix. The tensor product is defined as:

$$C_{i,j} = a_i b_j \quad (0 \leq j < m) \quad (0 \leq i < n) \tag{8}$$

where C is the resulting matrix with a shape of (n, m), a and b are input arrays with length n and m, respectively.

```
def tensor(a, b):
  n = len(a)
  m = len(b)
  return [
      [ a[i]*b[j] for j in range(m) ]
           for i in range(n)
  ]
```

4.2 Matrix Multiplication

Matrix multiplication is defined in summation notation as:

$$C_{i,j} = \sum_{k=0}^{k<n} A_{i,k} B_{k,j} \quad (0 \leq j < m) \quad (0 \leq i < n) \tag{9}$$

```python
def MatMult(A, B):
  n = len(A)    # no. rows in A
  Bn = len(B)    # no. rows in B & cols. in A
  m = len(B[0]) # no. cols. in B
  return [
  [sum([ A[i][k]*B[k][j] for k in range(Bn)])
            for j in range(m) ]
        for i in range(n) ]
```

Of course there is a more compact way to compute matrix multiplication in Python. Using `Numpy arrays` is the easiest way to do this.

```python
import numpy as np
A = np.array(A)
B = np.array(B)
C = A @ B
```

References

[1] E. W. Dijkstra. (1982, Aug) Why numbering should start at zero. [Online]. Available: http://www.cs.utexas.edu/users/EWD/transcriptions/EWD08xx/EWD831.html

[2] T. Oliphant, "NumPy: A guide to NumPy," USA: Trelgol Publishing, 2006–. [Online]. Available: http://www.numpy.org/

[3] Python Core Team, *Python: A dynamic, open source programming language*, Python Software Foundation, 2019, python version 3.7. [Online]. Available: https://www.python.org/