NUMPY DATA MANGEMENT

By: Frederick Johnson Keller

GISC 3200K – Programming for Geo Sci and Tech, Dr. Huidae Cho
University of North Georgia – Oakwood, Gainesville

Frederick Keller
Fjkell6446@ung.edu

1 Abstract

This paper discusses the data management and mathematical computations of CSV files using Python and NumPy and compares that code to an R script. It will attempt to explain how to use the NumPy module to import in a CSV file read it as a 2D array and manage the data within. The CSV files used contain multiple records of GPS coordinates at different points and the software will manage the data to create a table of averages of the GPS coordinates and compare them to a “True” points CSV file. The paper also explains how some of the complicated tools can be used to simplify the problem and achieve the desired goal. It will talk about the pros and cons of using NumPy instead of just working the CSV as a basic Python array. This project idea stemmed from a project in another class and proved quite useful in understanding aspects of both classes.

2 Introduction

For the purpose of experiment replication by later colleagues I will briefly explain the experiment that stemmed the idea for this NumPY Data Management project. I was assigned a class project in a Global Navigation Satellite Systems (GNSS) class, and the project was to collect GPS coordinates at certain spots around campus. The coordinates were first collected on each student’s phone using the same app on iPhone (Coordinates) and another on Androids. The class was later given Garmin etrex 20x GPS devices and told collect data using certain types of setting. The settings were labeled as “a” for GPS only, “b” for GPS and WAAS, and “c” for GPS/GLONASS and WAAS. Together the class collected about 1,200 GPS coordinates for the 11 points and the professor started introducing us to R coding to crunch the data because an earlier homework assignment that required us to use Excel proved very time consuming and frustrating. The solution was to automate the data management and my professor showed us R and were later assigned to write R code as homework.

As you may have guessed, I thought this would be easy, like copy n paste kind of easy with minor changes and lot of research to explain the minor differences. I was horribly mistaken and quickly found out R is the preferred data management language because it is a free, open sourced language with lot of free to download packages that make specifying how to organize the data easy. It is just I am in a Python programming course and I already proposed the idea so I figured maybe I can translate the R code into a Python toolbox and use it in ArcGIS Pro. That was a bigger headache than I had anticipated also, but I do encourage anyone who reads this to attempt to further my script and work it into a Python toolbox.

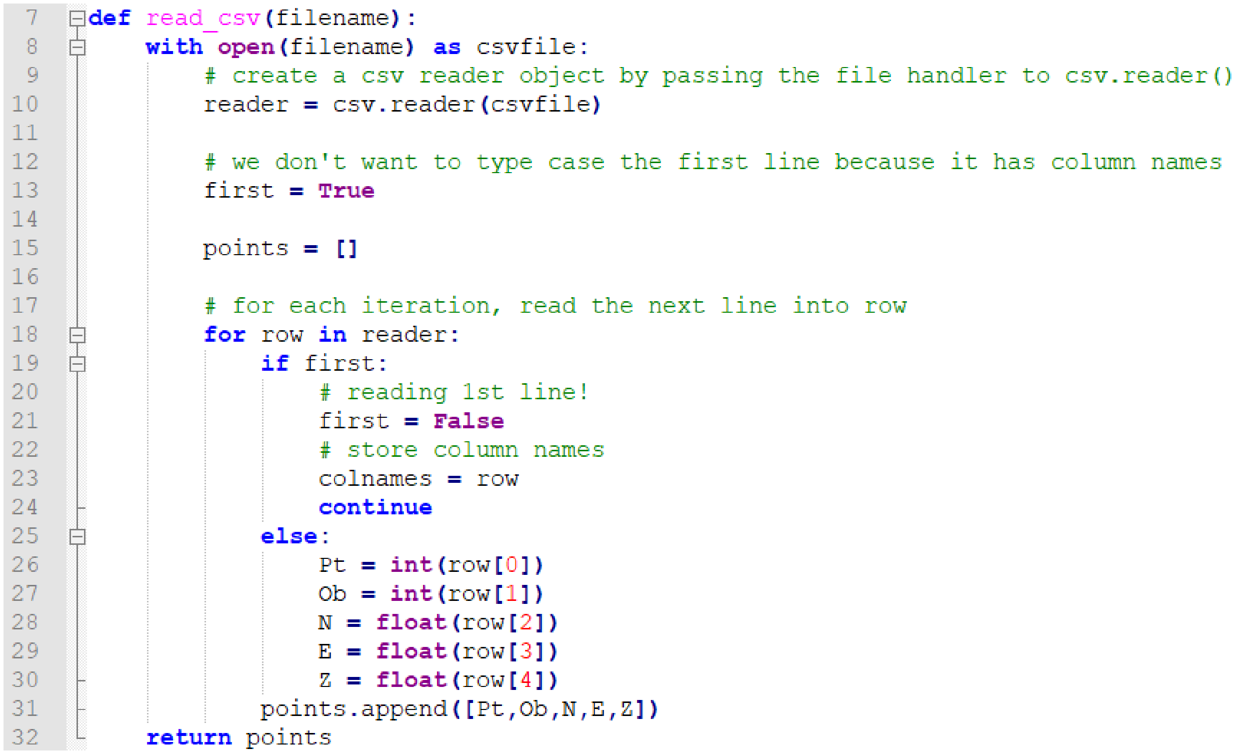
3 Materials and Methods

I will not be listing the materials used to do the experiment described in the introduction because this paper is about the R code being translated to Python not the difference in accuracy between GPS systems. The materials used to create the Python script are:

- Windows 10
- Python v 3.8
- NumPy v 20.2.1
- Notepad ++
- Python window found in ArcGIS Pro
- Microsoft Excel

The methods used had to be thought out, tried, and rethought out again. As previously stated in the latter half of the Introduction, I had originally proposed to translate the R code into a Python toolbox. I found a few different commands to import the CSVs and they worked in the toolbox and in a script.

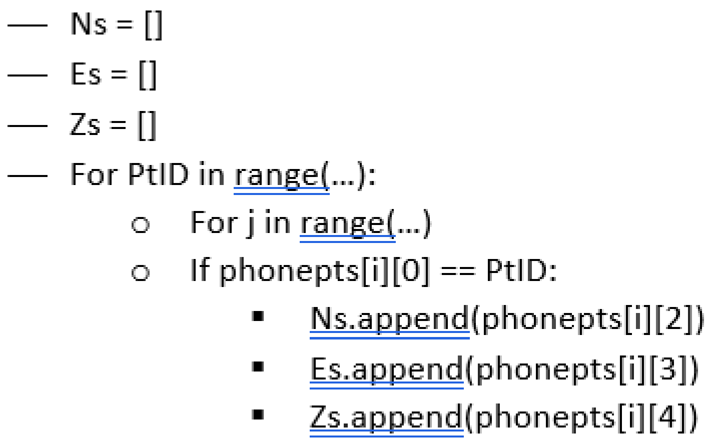
I had originally started by creating a 2D array and import the data into my array and it worked. I had managed to contain the data inside a self-made 2D array without completely understanding how basic 2D arrays work in Python. For those who do not know, Python auto cast all values in an array as a string, so you have to type cast each column with a for loop and an if statement. Mine looked like this:



```
def read_csv(filename):
    with open(filename) as csvfile:
        # create a csv reader object by passing the file handler to csv.reader()
        reader = csv.reader(csvfile)

        # we don't want to type case the first line because it has column names
        first = True

        points = []

        # for each iteration, read the next line into row
        for row in reader:
            if first:
                # reading 1st line!
                first = False
                # store column names
                colnames = row
                continue
            else:
                Pt = int(row[0])
                Ob = int(row[1])
                N = float(row[2])
                E = float(row[3])
                Z = float(row[4])
            points.append([Pt,Ob,N,E,Z])
    return points
```

Figure 1 Python code, imports csv to a 2D array

This worked and I was so happy, but it quickly started to show how difficult the rest of the project would be. I do understand indexing and slicing basics, and they are not too difficult concepts to comprehend. However, doing that on top of mathematical operations in a for loop can be a little mind numbing. I had attempted a “for” loop, but I found from both some research and my professor that I would need a nested for loop. My professor had proposed I run some code like this:



```
— Ns = []
— Es = []
— Zs = []
— For PtID in range(…):
    o For j in range(…)
    o If phonepts[i][0] == PtID:
        ▪ Ns.append(phonepts[i][2])
        ▪ Es.append(phonepts[i][3])
        ▪ Zs.append(phonepts[i][4])
```

Figure 2 2D array sample code

The Nested for loop above uses independent objects N, E, and Z to hold each individual aspect of the coordinate for every point recorded. I would now have 3 different single “for” loops to get averages for all three: N, E, and Z and append them together with a module command that allows appending along the axis = 1. Using axis = 1 appends them as columns, otherwise they may be appended as rows which will ruin everything.

Here is where a big difference between R and Python starts to show and it is how the two languages handle data in CSVs. Python requires the user to tell it what columns contain which data type like float, int, or str. R has the same requirements but you can tell the system which data type to use based of the columns header ID, where as in Python you would have to mark the column by its index. It does not sound more complicated but think about if you had not made the GPS records CSV. You may accidently miss index the column needed, in fact my project even had this problem.

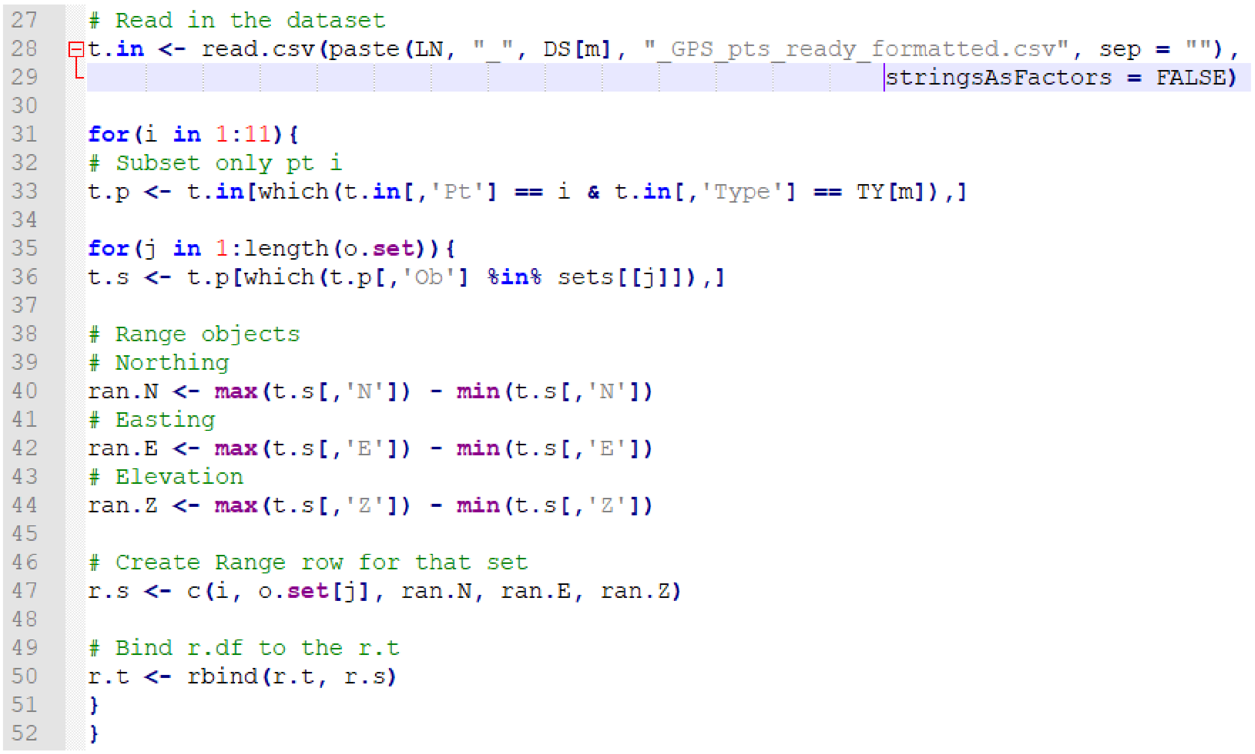
The big problem here is dataset formatting. For example, my CSV headers are formatted as such:

PhonePoints.csv: Pt(1-11), Ob(1-9), Type, Device, N, E, Z

TruePoints.csv Pt(1-11), Ob(zeros), N, E, Z

As you can plainly see I have more columns in my phonepts csv than my truepts csv which means matrix mathematics will not work. You can fix the number of columns by excluding “Type” and “Device” from the data selection like in figure 1. One thing you should notice now, figure 1 is coded incorrectly and pulls the data from columns “type” and “device” and puts them into columns N and E and tries to cast them as floats.

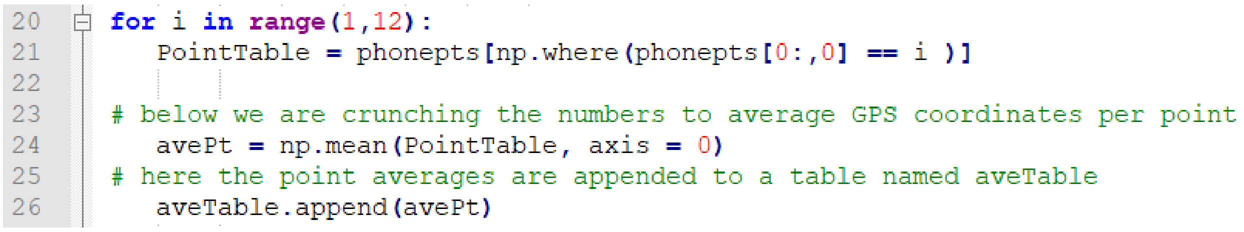
Take a look at some code to see the syntax differences between the two, figures 3 and 4.



```r
# Read in the dataset
t.in <- read.csv(paste(LN, "_", DS[m], "_GPS_pts_ready_formatted.csv", sep = ""),
                                                         stringsAsFactors = FALSE)

for(i in 1:11){
# Subset only pt i
t.p <- t.in[which(t.in[,'Pt'] == i & t.in[,'Type'] == TY[m]),]

for(j in 1:length(o.set)){
t.s <- t.p[which(t.p[,'Ob'] %in% sets[[j]]),]

# Range objects
# Northing
ran.N <- max(t.s[,'N']) - min(t.s[,'N'])
# Easting
ran.E <- max(t.s[,'E']) - min(t.s[,'E'])
# Elevation
ran.Z <- max(t.s[,'Z']) - min(t.s[,'Z'])

# Create Range row for that set
r.s <- c(i, o.set[j], ran.N, ran.E, ran.Z)

# Bind r.df to the r.t
r.t <- rbind(r.t, r.s)
}
}
```

Figure 3 Sample of R code

Above is a sample of R code that is reading in a CSV and pulling specified data in a nested for loop. If you look closely, you will notice that instead in indexing with numbers, the code is pulling data from columns by the header ID. I found this to be a much simpler and direct way to index columns for a for loop let alone a nested for loop. This also allows for extra columns in the original csv so you can process them as is. The next sample is from the Python code I used:



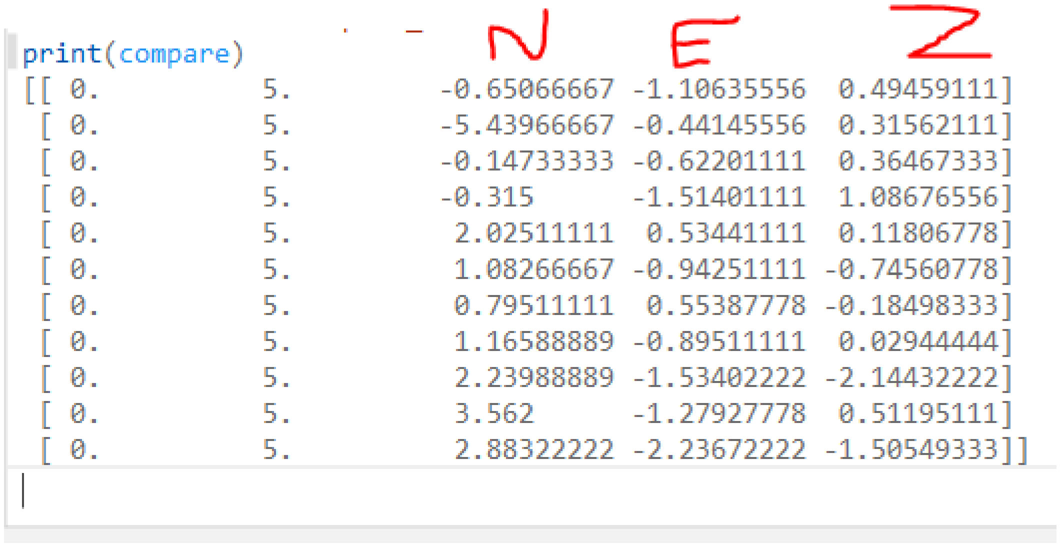
```python
20  for i in range(1,12):
21      PointTable = phonepts[np.where(phonepts[0:,0] == i )]
22
23  # below we are crunching the numbers to average GPS coordinates per point
24      avePt = np.mean(PointTable, axis = 0)
25  # here the point averages are appended to a table named aveTable
26      aveTable.append(avePt)
```

Figure 4 - A for loop from the Python script

You can tell this one is much simpler since it only has one for loop. What you are seeing here is the data being pulled based on the point ID. The goal is an average for each point and since each point was recorded 9 times, I need the software to check for all Pt 1s Obs 1-9. You may also notice the indexing at the end of line 21, it starts at 0 since Python is a zero-based counting system.

4 Results

The Python script to make a table of point averages and compare it to the TruePoints worked well. The results look a bit like this:



```
print(compare)
[[ 0.          5.          -0.65066667 -1.10635556  0.49459111]
 [ 0.          5.          -5.43966667 -0.44145556  0.31562111]
 [ 0.          5.          -0.14733333 -0.62201111  0.36467333]
 [ 0.          5.          -0.315      -1.51401111  1.08676556]
 [ 0.          5.           2.02511111  0.53441111  0.11806778]
 [ 0.          5.           1.08266667 -0.94251111 -0.74560778]
 [ 0.          5.           0.79511111  0.55387778 -0.18498333]
 [ 0.          5.           1.16588889 -0.89511111  0.02944444]
 [ 0.          5.           2.23988889 -1.53402222 -2.14432222]
 [ 0.          5.           3.562      -1.27927778  0.51195111]
 [ 0.          5.           2.88322222 -2.23672222 -1.50549333]]
```

Figure 5 - results when comparing phone points to the True Points

5 Discussion

Discussing results table:

The resulting table in the previous section is remarkably interesting. The negatives may seem strange or incorrect but since the coordinates are in NAD 1983 UTM Zone 17N format the table values can be read as meters away. The negatives in the N column simply mean the point was xx.xxx meters south while E column values can be read as xx.xxx meters west of the true point. I would also recommend updating the file in Microsoft Excel and format the N, E, Z columns as numbers with only 3 decimal places lime such:

Table 1 - Tab delimited results table

-0.651	-1.106	0.495
-5.440	-0.441	0.316
-0.147	-0.622	0.365
-0.315	-1.514	1.087
2.025	0.534	0.118
1.083	-0.943	-0.746
0.795	0.554	-0.185
1.166	-0.895	0.029
2.240	-1.534	-2.144
3.562	-1.279	0.512
2.883	-2.237	-1.505

Code Discussion:

The only problem in the Python code presented is it cannot do as much as the original R script. The R script was written to not only handle the PhonePoints.csv but also a GarminPoints.csv. The complication lies in the GarminPoints.csv since it has a “Type” column with more than one string value. The values differentiate the systems used to record the coordinates and the values are ‘a’ for GPS only, ‘b’ for GPS and WAAS, and ‘c’ for GPS, WAAS, and GLONASS systems. You can actually see this definition in the code in Figure 3’s line 33, “& t.in[,’Type’] == TY =[m]),]. The object TY was defined earlier in the script as a small array of values: [‘p’, ‘a’, ‘b’, ‘c’] so the values could be meticulously organized.

I tried fixing the problem myself but found it difficult to import all the data from the table and type cast that single ‘Type’ column as strings. I also could not find any information about whether or not the ‘&’ operator would work the same as it did in the R script. If it does then both CSV files can pass through the for loop but if not separate for loops would be needed for the two CSVs (phone and Garmin).

6 Conclusions

The R script was more fluid and capable than my basic 2D array and my NumPy array because it could differentiate the Type column string values. I am displeased I did not figure it all out before the semester ended, but I am proud to say the Python code does what it was intended to do. I also enjoy the idea that these two projects worked really well together to serve a common goal, education and understanding. If someone else was going to improve upon my work I would recommend they figure out more possibilities with NumPy to help sort through the Garmin file to pick out coordinates that are typed: ‘a’, ‘b’, ‘c’ then average them and follow the work flow presented.

Code Samples:

I want to insert the data of CSV file (network data such as: time, IP address, port number) into 2D list in Python.

Here is the code:

```
import csv
datafile = open('a.csv', 'r')
datareader = csv.reader(datafile, delimiter=';')
data = []
for row in datareader:
   data.append(row)
print (data[1:4])
```

pafpaf, screenname: et al. "Reading CSV File and Inserting It into 2d List in Python." *Stack Overflow*, StackOverFlow, 7 July 2014, stackoverflow.com/questions/24606650/reading-csv-file-and-inserting-it-into-2d-list-in-python.

Example: Adding Matrices

To add, the matrices will make use of a for-loop that will loop through both the matrices given.

```
M1 = [[8, 14, -6], [12,7,4], [-11,3,21]]

M2 = [[3, 16, -6], [9,7,-4], [-1,3,13]]

M3 = [[0,0,0], [0,0,0], [0,0,0]]

matrix_length = len(M1)

#To Add M1 and M2 matrices

for i in range(len(M1)):

        for k in range(len(M2)):

        M3[i][k] = M1[i][k] + M2[i][k]

#To Print the matrix

print("The sum of Matrix M1 and M2 = ", M3)
```

Output:

```
The sum of Matrix M1 and M2 =  [[11, 30, -12], [21, 14, 0], [-12, 6, 34]]
```

So here ^ I am trying to have M1 and M2 == CSVs and M3 be our average tables, the question becomes. Do I need the "len()" command since I know I want the average of Pts = 1:9?

- for i in range(len(M1[4],[5],[6])):
    - for k in 1:9 :

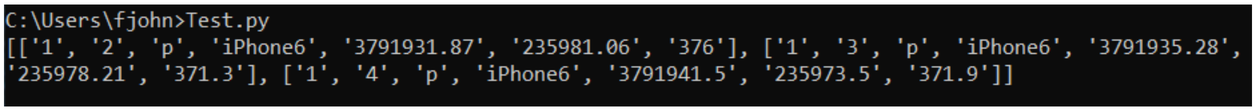
    - M3[i][k] = M1[i][k] + M2[i][k]
- Daidalos. “OPEN NOTEBOOKS.” *How to Perform Mathematical Operations on Array Elements in Python ?*, 2 Aug. 2019, moonbooks.org/Articles/How-to-perform-mathematical-operations-on-array-elements-in-python-/.

My Personal scripting, mid coding I took a snapshot.

I have scripted:

- import csv
- datafile = open('D:\Fall2020\Programming - GISC 3200K\Final Project\GNSS Data\PhonePoints.csv', 'r')
- datareader = csv.reader(datafile, delimiter=',')
- first = True
- data = []
- for row in datareader:
    - if first:
        - first = False
        - colnames = row
        - continue
    - else:
        - Pt = int(row[0])
        - Ob = int(row[1])
        - N = float(row[4])
        - E = float(row[5])
        - Z = float(row[6])
    - data.append(row)
- print (data[1:4])

^ Resulted: Should I be concerned about the ‘###’ ?? seems to signal a str not a float.


```
C:\Users\fjohn>Test.py
[['1', '2', 'p', 'iPhone6', '3791931.87', '235981.06', '376'], ['1', '3', 'p', 'iPhone6', '3791935.28',
'235978.21', '371.3'], ['1', '4', 'p', 'iPhone6', '3791941.5', '235973.5', '371.9']]
```

I later found out the reason I had the quotes was because I had forgotten to append my new objects together, so it was simply reading the CSV like normal which casts everything as a string.

References:

Na, The SciPy community. “Numpy.append¶.” *Numpy.append - NumPy v1.19 Manual*, The SciPy Community, 29 June 2020, numpy.org/doc/stable/reference/generated/numpy.append.html?highlight=numpy+append.

Intext: (The SciPy community *numpy.append¶*)

NA, The SciPy community. “Numpy.where¶.” *Numpy.where - NumPy v1.19 Manual*, The SciPy Community, 29 June 2020, numpy.org/doc/stable/reference/generated/numpy.where.html.

Intext: (The SciPy community *numpy.where¶*)

NA, The SciPy community. “Numpy.genfromtxt¶.” *Numpy.genfromtxt - NumPy v1.21.dev0 Manual*, The SciPy Community, 28 June 2008, numpy.org/devdocs/reference/generated/numpy.genfromtxt.html.

Intext: (The SciPy community *numpy.genfromtxt¶*)

Daidalos. “OPEN NOTEBOOKS.” *How to Perform Mathematical Operations on Array Elements in Python ?*, 2 Aug. 2019, moonbooks.org/Articles/How-to-perform-mathematical-operations-on-array-elements-in-python-/.