Download this page as a jupyter notebook at Lab 9-TH

Laboratory 9: Matrices a Blue Pill Approach

Medrano, Giovanni

R11521018

ENGR 1330 Laboratory 9 - Homework

In [1]:
```
# Preamble script block to identify host, user, and kernel
import sys
! hostname
! whoami
print(sys.executable)
print(sys.version)
print(sys.version_info)
```

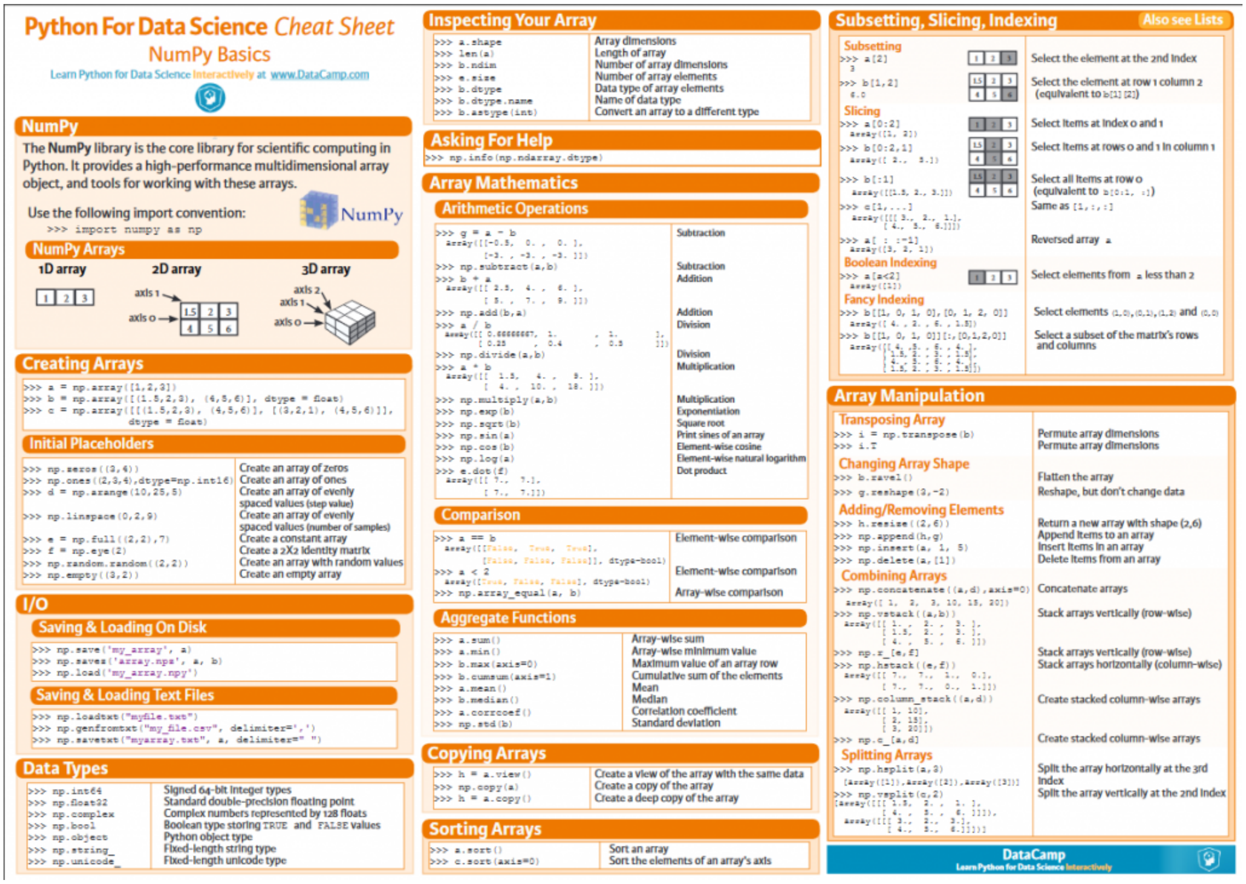
```
DESKTOP-6HAS1BN
desktop-6has1bn\medra
C:\Users\medra\anaconda3\python.exe
3.8.5 (default, Sep  3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)]
sys.version_info(major=3, minor=8, micro=5, releaselevel='final', serial=0)
```

Numpy Cheat Sheet(s)

Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays. The library's name is short for "Numeric Python" or "Numerical Python".

A pdf file of a summary sheet (you need to download):
https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Numpy_Python_Cheat_Sheet.pdf

or the graphic below (same information):



Python For Data Science *Cheat Sheet*

NumPy Basics

Learn Python for Data Science Interactively at www.DataCamp.com

NumPy

The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

```
>>> import numpy as np
```

NumPy

NumPy Arrays

1D array | 2D array | 3D array

axis 1, axis 0 | axis 2, axis 1, axis 0

Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

Initial Placeholders

`>>> np.zeros((3,4))`	Create an array of zeros
`>>> np.ones((2,3,4),dtype=np.int16)`	Create an array of ones
`>>> d = np.arange(10,25,5)`	Create an array of evenly spaced values (step value)
`>>> np.linspace(0,2,9)`	Create an array of evenly spaced values (number of samples)
`>>> e = np.full((2,2),7)`	Create a constant array
`>>> f = np.eye(2)`	Create a 2X2 identity matrix
`>>> np.random.random((2,2))`	Create an array with random values
`>>> np.empty((3,2))`	Create an empty array

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Data Types

`>>> np.int64`	Signed 64-bit integer types
`>>> np.float32`	Standard double-precision floating point
`>>> np.complex`	Complex numbers represented by 128 floats
`>>> np.bool`	Boolean type storing `TRUE` and `FALSE` values
`>>> np.object`	Python object type
`>>> np.string_`	Fixed-length string type
`>>> np.unicode_`	Fixed-length unicode type

Inspecting Your Array

`>>> a.shape`	Array dimensions
`>>> len(a)`	Length of array
`>>> b.ndim`	Number of array dimensions
`>>> e.size`	Number of array elements
`>>> b.dtype`	Data type of array elements
`>>> b.dtype.name`	Name of data type
`>>> b.astype(int)`	Convert an array to a different type

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

`>>> g = a - b`
`array([[-0.5, 0. , 0. ],`
`[-3. , -3. , -3. ]])`
	Subtraction

`>>> np.subtract(a,b)`	Subtraction

`>>> b + a`
`array([[ 2.5, 4. , 6. ],`
`[ 5. , 7. , 9. ]])`
	Addition

`>>> np.add(b,a)`	Addition

`>>> a / b`
`array([[ 0.66666667, 1. , 1. ],`
`[ 0.25 , 0.4 , 0.5 ]])`
	Division

`>>> np.divide(a,b)`	Division

`>>> a * b`
`array([[ 1.5, 4. , 9. ],`
`[ 4. , 10. , 18. ]])`
	Multiplication

`>>> np.multiply(a,b)`	Multiplication
`>>> np.exp(b)`	Exponentiation
`>>> np.sqrt(b)`	Square root
`>>> np.sin(a)`	Print sines of an array
`>>> np.cos(b)`	Element-wise cosine
`>>> np.log(a)`	Element-wise natural logarithm

`>>> e.dot(f)`
`array([[ 7., 7.],`
`[ 7., 7.]])`
	Dot product

Comparison

`>>> a == b`
`array([[False, True, True],`
`[False, False, False]], dtype=bool)`
	Element-wise comparison

`>>> a < 2` `array([True, False, False], dtype=bool)`	Element-wise comparison
`>>> np.array_equal(a, b)`	Array-wise comparison

Aggregate Functions

`>>> a.sum()`	Array-wise sum
`>>> a.min()`	Array-wise minimum value
`>>> b.max(axis=0)`	Maximum value of an array row
`>>> b.cumsum(axis=1)`	Cumulative sum of the elements
`>>> a.mean()`	Mean
`>>> b.median()`	Median
`>>> a.corrcoef()`	Correlation coefficient
`>>> np.std(b)`	Standard deviation

Copying Arrays

`>>> h = a.view()`	Create a view of the array with the same data
`>>> np.copy(a)`	Create a copy of the array
`>>> h = a.copy()`	Create a deep copy of the array

Sorting Arrays

`>>> a.sort()`	Sort an array
`>>> c.sort(axis=0)`	Sort the elements of an array's axis

Subsetting, Slicing, Indexing

Also see Lists

Subsetting	
`>>> a[2]` `3`	Select the element at the 2nd index
`>>> b[1,2]` `6.0`	Select the element at row 1 column 2 (equivalent to `b[1][2]`)
Slicing	
`>>> a[0:2]` `array([1, 2])`	Select items at index 0 and 1
`>>> b[0:2,1]` `array([ 2., 5.])`	Select items at rows 0 and 1 in column 1
`>>> b[:1]` `array([[1.5, 2., 3.]])`	Select all items at row 0 (equivalent to `b[0:1, :]`)

`>>> c[1,...]`
`array([[[ 3., 2., 1.],`
`[ 4., 5., 6.]]])`
	Same as `[1,:,:]`

`>>> a[ : :-1]` `array([3, 2, 1])`	Reversed array `a`
Boolean Indexing	
`>>> a[a<2]` `array([1])`	Select elements from `a` less than 2
Fancy Indexing	
`>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]` `array([ 4. , 2. , 6. , 1.5])`	Select elements (1,0),(0,1),(1,2) and (0,0)

`>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]`
`array([[ 4. ,5. , 6. , 4. ],`
`[ 1.5, 2. , 3. , 1.5],`
`[ 4. , 5. , 6. , 4. ],`
`[ 1.5, 2. , 3. , 1.5]])`
	Select a subset of the matrix's rows and columns

Array Manipulation

Transposing Array	
`>>> i = np.transpose(b)`	Permute array dimensions
`>>> i.T`	Permute array dimensions
Changing Array Shape	
`>>> b.ravel()`	Flatten the array
`>>> g.reshape(3,-2)`	Reshape, but don't change data
Adding/Removing Elements	
`>>> h.resize((2,6))`	Return a new array with shape (2,6)
`>>> np.append(h,g)`	Append items to an array
`>>> np.insert(a, 1, 5)`	Insert items in an array
`>>> np.delete(a,[1])`	Delete items from an array
Combining Arrays	
`>>> np.concatenate((a,d),axis=0)` `array([ 1, 2, 3, 10, 15, 20])`	Concatenate arrays

`>>> np.vstack((a,b))`
`array([[ 1. , 2. , 3. ],`
`[ 1.5, 2. , 3. ],`
`[ 4. , 5. , 6. ]])`
	Stack arrays vertically (row-wise)

`>>> np.r_[e,f]`	Stack arrays vertically (row-wise)

`>>> np.hstack((e,f))`
`array([[ 7., 7., 1., 0.],`
`[ 7., 7., 0., 1.]])`
	Stack arrays horizontally (column-wise)

`>>> np.column_stack((a,d))`
`array([[ 1, 10],`
`[ 2, 15],`
`[ 3, 20]])`
	Create stacked column-wise arrays

`>>> np.c_[a,d]`	Create stacked column-wise arrays
Splitting Arrays	
`>>> np.hsplit(a,3)` `[array([1]),array([2]),array([3])]`	Split the array horizontally at the 3rd index

`>>> np.vsplit(c,2)`
`[array([[[ 1.5, 2. , 1. ],`
`[ 4. , 5. , 6. ]]]),`
`array([[[ 3., 2., 3.],`
`[ 4., 5., 6.]]])]`
	Split the array vertically at the 2nd index

DataCamp

Learn Python for Data Science Interactively

Exercise 1: Going down to Southpark .... ?

Using the names below:

"Cartman, Kenny, Kyle, Stan, Butters, Wendy, Chef, Mr. Mackey, Randy, Sharon, Sheila, Towelie"

A) Create a 3x4 array and print it out

B) Sort the array alphabetically by column and print it out

C) From the sorted array, slice out a 2x2 array with ('Cartman, Randy, Sharon, Wendy'), and print it out

In [18]:
```
#import numpy
import numpy as np
# make an arry of names
names = np.array([['Cartman','Kenny','Kyle','Stan'],['Butters','Wendy','Chef','Mr.Macke
print(names)
# print array
print('---------------------')
namesArray = names.transpose()
print(namesArray)
print('---------------------')
print("Sorted by Column")
x = np.sort(namesArray)
namesArray = x.transpose()
print(namesArray)
print('---------------------')
print('sliced')
finals = np.ix_([1,2],[0,1])
final = namesArray[finals]
print(final)

# slice and print sliced array
```

```
[['Cartman' 'Kenny' 'Kyle' 'Stan']
 ['Butters' 'Wendy' 'Chef' 'Mr.Mackey']
 ['Randy' 'Sharon' 'Shiela' 'Towelie']]
---------------------
[['Cartman' 'Butters' 'Randy']
 ['Kenny' 'Wendy' 'Sharon']
 ['Kyle' 'Chef' 'Shiela']
 ['Stan' 'Mr.Mackey' 'Towelie']]
---------------------
Sorted by Column
[['Butters' 'Kenny' 'Chef' 'Mr.Mackey']
 ['Cartman' 'Sharon' 'Kyle' 'Stan']
 ['Randy' 'Wendy' 'Shiela' 'Towelie']]
---------------------
sliced
[['Cartman' 'Sharon']
 ['Randy' 'Wendy']]
```

Exercise 2: A Numpy Playground

- Step1: Create a 2x5 array with [0,1,2,3,4,5,6,7,8,9] and name it "Array1"
- Step2: Extract all the odd numbers of "Array1" and store them in a new array: "Array2"
- Step3: In "Array1" replace 0,5, and 9 with 100,500, and 900.
- Step4: Add three other odd numbers to "Array2"
- Step5: Calculate the average of "Array2"
- Step6: Take "Array2" to the power of 5 and print the result.

In [35]:
```
#import numpy
import numpy as np

# create and print array1
```

```
array = [0,1,2,3,4,5,6,7,8,9]
arrayReshaped = np.array(array).reshape(2,5)
print(arrayReshaped)
print('\n')
# create and print array2

array2 = [arrayReshaped[arrayReshaped % 2 == 1]]
print(array2)
print('\n')
# make replacements, print result
arrayReshaped[arrayReshaped == 0] = 100
arrayReshaped[arrayReshaped == 5] = 500
arrayReshaped[arrayReshaped == 9] = 900
print(arrayReshaped)
print('\n')
# Add three other odd numbers to "Array2"
array2 = np.append(array2, 11)
array2 = np.append(array2, 13)
array2 = np.append(array2, 15)
print(array2)
print('\n')
# Calculate the average of "Array2"
print(np.average(array2))
print('\n')
# Take "Array2" to the power of 5 and print the result.
print(np.power(array2, 5))
```

```
[[0 1 2 3 4]
 [5 6 7 8 9]]


[array([1, 3, 5, 7, 9])]


[[100   1   2   3   4]
 [500   6   7   8 900]]


[ 1  3  5  7  9 11 13 15]


8.0


[     1    243   3125  16807  59049 161051 371293 759375]
```

Exercise 3:

- Step1: Create a 1D Numpy array of all the numbers in your R-number: R######## | It should have eight numbers
- Step2: Create another 1D array by multiplying the previous array by 2
- Step3: Create another 1D array by multiplying the first array by 5

In [41]:

```
# import numpy
import numpy as np

arr = np.array([1,1,5,2,1,0,1,8])
print(arr)
```

```
print('\n')
# Create a 1D Numpy array of all the numbers in your R-number
arr2 = np.multiply(arr,2)
print(arr2)
print('\n')
# Create another 1D array by multiplying the previous array by 2
arr3 = np.multiply(arr,5)
print(arr3)
# Create another 1D array by multiplying the first array by 5
```

```
[1 1 5 2 1 0 1 8]


[ 2  2 10  4  2  0  2 16]


[ 5  5 25 10  5  0  5 40]
```

You will also find this link helpful for the next two exercises: https://www.codecademy.com/learn/learn-linear-algebra/modules/math-ds-linear-algebra/cheatsheet

Exercise 4

Consider the linear system given by

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{B}$$
where

\begin{gather} \mathbf{A}= \begin{pmatrix} 8.17\times10^6 & 0 &-0.5\times10^6\\ -1.5\times10^6 & 7.21\times10^6 & 0 \\ 0 & -1.5\times10^6 & 11.5\times10^6 \\ \end{pmatrix} ~\mathbf{b}= \begin{pmatrix} 2.0\times10^9 \\ 4.0\times10^9 \\ 1.0\times10^9 \\ \end{pmatrix} \end{gather}
Use **numpy** and **linalg** methods to find **x**

In [49]:
```
# import numpy
import numpy as np
# create A
A = np.array([[8.17*(10**6),0,-0.5*(10**6)],[-1.5*(10**6),7.21*(10**6),0],[0,-1.5*(10**
# create B
B = np.array([[2.0*(10**9)],[4.0*(10**9)],[1.0*(10**9)]])
print(A)
print('\n')
print(B)
print('\n')
# x = Ainv times B
inverse = np.linalg.inv(A)
print(inverse)
print('\n')
print('Now the value of x1, x2, and x3 is:')
x = np.dot(inverse, B)
print(x)
# print result
```

```
[[ 8170000.        0.  -500000.]
 [-1500000.  7210000.        0.]
 [       0. -1500000. 11500000.]]
```

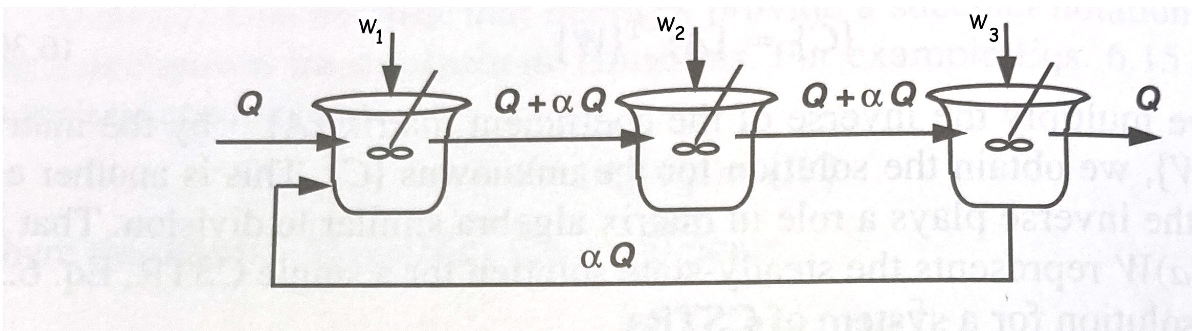
```
[[2.e+09]
 [4.e+09]
 [1.e+09]]


[[1.22602630e-07 1.10899080e-09 5.33054913e-09]
 [2.55067885e-08 1.38926975e-07 1.10899080e-09]
 [3.32697241e-09 1.81209097e-08 8.71011727e-08]]


Now the value of x1, x2, and x3 is:
[[254.97177212]
 [607.83046577]
 [166.2387564 ]]
```

Exercise 5.

Consider the steady-state reactor system with feedback as shown below:



	1	2	3
Volume, 10^6 m^3	2	4	3
Mean depth, m	3	7	3
Surface area, 10^6 m^2	0.667	0.571	1.000
Loading, kg yr^{-1}	2000	4000	1000

A mass balance of the system assuming complete mixing in each reactor (pond, lake, vat, ...) is

\begin{gather} \begin{matrix} W_1 = & (Q+\alpha Q + v A_1) \times C_1 & + 0 \times C_2 &- \alpha Q \times C_3\\ W_2 = & -(Q+\alpha Q) \times C_1 & + (Q+\alpha Q + v A_2) \times C_2 & 0 \times C_3 \\ W_3 = & 0 \times C_1 & - (Q+\alpha Q)\times C_2 & +(Q+\alpha Q + v A_3) \times C_3\\ \end{matrix} \end{gather}
where W_i is the loading in kilograms/year , A_i is the reactor surface area in 10^6 meters squared, v is the settling rate in each reactor in meters/year, α is the recycle fraction, and Q is the volumetric flow rate through the system. The unknown values are the constituient concentrations C_i in each reactor.

Decomposing the system of equations above into a Matrix-vector system $\mathbf{A} \cdot \mathbf{C} = \mathbf{W}$, where $\mathbf{A}$ is the coefficient matrix (which we will build using

Q,α, and $v A_i$), $\mathbf{C}$ is the vector of unknown concentrations of the constituients, and $\mathbf{W}$ is the loading vector yields:

\begin{gather} \begin{pmatrix} (Q+\alpha Q + v A_1) & + 0 &- \alpha Q \\ -(Q+\alpha Q) & + (Q+\alpha Q + v A_2) & 0 \\ 0 & - (Q+\alpha Q)& +(Q+\alpha Q + v A_3) \\ \end{pmatrix} \cdot \begin{pmatrix} C_1 \\ C_2 \\ C_3 \\ \end{pmatrix} ~=~ \begin{pmatrix} W_1 \\ W_2 \\ W_3 \\ \end{pmatrix} \end{gather}
Complete a script that constructs $\mathbf{A}$ given $Q=1\times10^6$ cubic meters/year, α = 0.5, and v = 10 meters/year. Estimate the reactor constituient concentrations in parts per million (milligrams).

Now repeat the computation with α = 0.0

In [56]:

```
# import numpy
import numpy as np
# create constants and properties arrays
v = 10
alpha = .5
Q = (1)*(10)**(6)
Q = (Q)*(10)**(-6)
v = (10)*(10)**(6)
# get units correct

l = np.array([2000,4000,1000])
l = (l)*(10)**(6)
print('Load is:')
print(l)
print('\n')

# Load = Load*1e6 #Load in milligrams
a = np.array([0.667,0.5871,1.000])
a = (a)*(10)**(6)
print('Area is:')
print(a)
print('\n')

# Area = Area*1e6 #Area in million squre meters

coff = [[(Q+alpha*Q+v*a[0]),(0),-(alpha*Q)],
        [-(Q+alpha*Q),(Q+alpha*Q+v*a[1]),(0)],
        [(0),-(Q+alpha*Q),(Q+alpha*Q+v*a[2])]]
print('The coefficient array is:')
print(coff)
print('\n')
# construct the coefficient array
inverse = np.linalg.inv(coff)
print('The inverse is:')
print(inverse)
print('\n')
# invert and solve for c
result = np.dot(inverse, l)
print('The final result of C1, C2, and C3 is:')
print(result)
# print result
```

```
Load is:
[2000000000 -294967296 1000000000]
```

```
Area is:
[ 667000.  587100. 1000000.]


The coefficient array is:
[[6670000000001.5, 0, -0.5], [-1.5, 5871000000001.5, 0], [0, -1.5, 10000000000001.5]]


The inverse is:
[[1.49925037e-13 1.91524064e-39 7.49625187e-27]
 [3.83048128e-26 1.70328734e-13 1.91524064e-39]
 [5.74572193e-39 2.55493102e-26 1.00000000e-13]]


The final result is:
[ 2.99850075e-04 -5.02414062e-05  1.00000000e-04]
```

In [59]:

```python
# import numpy
import numpy as np
# create constants and properties arrays - change to alpha=0
v = 10
alpha = 0.0
Q = (1)*(10)**(6)
Q = (Q)*(10)**(-6)
v = (10)*(10)**(6)
# get units correct
# Load = Load*1e6 #Load in milligrams
l = np.array([2000,4000,1000])
l = (l)*(10)**(6)
print('Load is:')
print(l)
print('\n')
# Area = Area*1e6 #Area in million squre meters
a = np.array([0.667,0.5871,1.000])
a = (a)*(10)**(6)
print('Area is:')
print(a)
print('\n')
# construct the coefficient array
coff = [[(Q+alpha*Q+v*a[0]),(0),-(alpha*Q)],
        [-(Q+alpha*Q),(Q+alpha*Q+v*a[1]),(0)],
        [(0),-(Q+alpha*Q),(Q+alpha*Q+v*a[2])]]
print('The coefficient array is:')
print(coff)
print('\n')
# invert and solve for c
inverse = np.linalg.inv(coff)
print('The inverse is:')
print(inverse)
print('\n')
# print result
result = np.dot(inverse, l)
print('The final result of C1, C2, and C3 is:')
print(result)
```

```
Load is:
[2000000000 -294967296 1000000000]


Area is:
```

```
[ 667000.  587100. 1000000.]


The coefficient array is:
[[6670000000001.0, 0, -0.0], [-1.0, 5871000000001.0, 0], [0, -1.0, 10000000000001.0]]


The inverse is:
[[1.49925037e-13 0.00000000e+00 0.00000000e+00]
 [2.55365419e-26 1.70328734e-13 0.00000000e+00]
 [2.55365419e-39 1.70328734e-26 1.00000000e-13]]


The final result of C1, C2, and C3 is:
[ 2.99850075e-04 -5.02414062e-05  1.00000000e-04]
```

In [ ]: