**User’s Guide**

Welcome to George and Sean’s Matrix Algebra Package!

This package comes with many features allowing your computer to perform operations on matrices for you. The operations range from simple addition or multiplication of matrices to more complex and normally time-consuming operations, like inversing and reducing a matrix.

Firstly, you will need to know how to use this package. You will have to know that this package is based around a Matrix class in Python 3.3. Some basic familiarity with the Python 3.3 shell will be assumed. First, open the “Matrix Class.py” file and press F5 to get into the shell.

To create a matrix, you will have to use the Matrix class. This can be done by initializing a matrix and storing it into a variable of your choosing. For the sake of simplicity, let’s use the variable M for the matrix . Now, to initialize this matrix, we will have to give Matrix() an input of [ [2,5], [1,3] ]. Indeed, the simplest way for Python to interpret a matrix is through a list of lists (a 2D list) where each element of the list is itself a list! Each sublist will represent a row of the matrix. The code then to initialize this matrix in the shell is the following:

>>> M = Matrix( [ [2,5], [1,3] ] )

Now, once we have a variable (here: M) assigned to a Matrix, there are many things we can do with this matrix with the methods, or functions, associated with the Matrix class:

***Pretty printing***

**self.prettyprint()**

The normal string output for a matrix class is just a simple list of lists, not that elegant to look at. If we want to see the matrix in a manner we’re more used to seeing, we can call the .prettyprint() method after the matrix. Let’s call this method on the matrix M we created:

>>> M.prettyprint()

2.00 5.00

1.00 3.00

***Adding***

**self + other (\_\_add\_\_)**

We can add matrices together. Let’s initialize a second matrix to the variable B and then add the M and B matrices:

>>> B = Matrix( [ [1,6], [2,4] ] )

>>> M + B

[[3,11], [3,7]]

We can store this addition in a variable ‘a’ and pretty print it:

>>> a = M + B

>>> a.prettyprint()

3.0 11.0

3.0 7.0

***Subtracting***

**self - other (\_\_sub\_\_)**

We can subtract matrices. Let’s initialize a second matrix to the variable B and then subtract the M and B matrices:

>>> B = Matrix( [ [1,6], [2,4] ] )

>>> M - B

[[1,-1], [-1,-1]]

We can store this addition in a variable ‘a’ and pretty print it:

>>> a = M - B

>>> a.prettyprint()

1.0 -1.0

-1.0 -1.0

***Multiplying***

**self \* other (\_\_mul\_\_)**

We can multiply matrices together. Let’s initialize a second matrix to the variable A and then subtract the M and B matrices:

>>> A = Matrix( [ [1,6,3], [2,1,4] ] )

>>> M \* B

[[12, 17, 26], [7, 9, 15]]

***Multiplying by a scalar***

**scalar \* self (\_\_rmul\_\_)**

We can multiply matrices by a scalar.

**Note:** In this package, the scalar must always be placed **BEFORE** the matrix:

>>> M = Matrix( [ [2,5], [1,3] ] )

>>> 5 \* M

[[10, 25], [5, 15]]

***Raising to a power***

**self \*\*power (\_\_pow\_\_)**

We can raise the matrix to a certain power, and this will give us another matrix. If we raise M to the power 3, we get another matrix. If we wish, we can assign another variable to this new matrix. Let’s use the variable ‘a’:

>>> a = M\*\*3

If we then call a, the new matrix will be displayed:

>>> a

[[43,120], [24,67]]

We can also pretty print the matrix to get a better view of it:

>>> a.prettyprint()

43.00 120.00

24.00 67.00

***Transposing***

**self.trans()**

We can take the transpose of our matrix as well by calling .trans() on it.

**Note:** This method will change the matrix **in place**, meaning that the original matrix will be modified:

>>> M.trans()

>>> M.prettyprint()

2.00 1.00

5.00 3.00

We can see here that the original matrix M has changed; it has been transposed.

**self.getTrans()**

We can also take the transpose of a matrix by making a copy, without modifying the original matrix by using the .getTrans() method. The result can then be stored in another variable (‘a’ here):

>>> M.getTrans()

[[2,1], [5,3]]

>>> a = M.getTrans()

>>> a

[[2,1], [5,3]]

***Finding the determinant***

**self.det**

We can also find the determinant of a matrix with the .det property.

**Note:** This is a property, and not a method, of a matrix. We therefore don’t have to use the brackets ‘()’ after calling ‘.det’. Let’s find the determinant of our original matrix M:

>>> M = Matrix( [ [2,5], [1,3] ] )

>>> M.det

1

The determinant of our matrix M is 1.

***Inversing***

**self.inverse()**

To invert a matrix, we can call the .inverse() method.

**Note:** This method will change the matrix **in place**, meaning that the original matrix will be modified:

>>> M.inverse()

>>> M.prettyprint()

3.00 -5.00

-1.00 2.00

**self.getInverse()**

We can also inverse a matrix by making a copy, without modifying the original matrix by using the .getInverse() method. The result can then be stored in another variable (‘a’ here):

>>> M.getInverse()

[[3,-5], [-1,2]]

>>> a = M.getTrans()

>>> a

[[3,-5], [-1,2]]

***Reducing***

**self.rowEchelon()**

To reduce a matrix to row echelon form, we can call the .rowEchelon() method.

**Note:** This method will change the matrix **in place**, meaning that the original matrix will be modified:

>>> M.rowEchelon()

>>> M.prettyprint()

2.0 5.0

0.0 0.5

**self.reduce()**

To reduce a matrix to **reduced** row echelon form, we can call the .reduce() method.

**Note:** This method will change the matrix **in place**, meaning that the original matrix will be modified:

>>> M.reduce()

>>> M.prettyprint()

1.0 0.0

0.0 1.0

***Obtaining the cofactor matrix***

**self.cofactors ()**

To obtain the cofactor matrix of a matrix, we can call the .cofactors() method.

>>> a = M.cofactors()

>>> a.prettyprint()

3.0 -1.0

-5.0 2.0

***Appending a matrix to another***

**self.append(other)**

We can append a matrix to another matrix, creating a new extended matrix.

**Note:** This method will change the original matrix **in place**, meaning that the original matrix will be modified:

>>> M = Matrix( [ [2,5], [1,3] ] )

>>> A = Matrix( [ [1,6,3], [2,1,4] ] )

>>> M.append(A)

>>> M.prettyprint()

2.0 5.0 1.0 6.0 3.0

1.0 3.0 2.0 1.0 4.0

***Creating the identity matrix***

**identity(n)**

A fast way of creating an identity matrix of size n:

>>> identity(3)

[[1,0,0], [0,1,0], [0,0,1]]

With all these methods, relatively complex chains of operations on matrices can be carried out and a result will be given quick as grease lightning!

>>> M = Matrix( [ [2,5], [1,3] ] )

>>> (M.det\*(((1/5)\*(M\*\*6)+M) \*M.getInverse())).prettyprint()

198.4 551.0

110.2 308.6

**MARKOV CHAINS**

An application to Markov Chains can be seen in the ‘Markov Chains Test Run.py’ file. Open the file for details.