Matrices as Regular Markov Chains

Bharath Variar
2019B5A70930H

Submission for Project 2
MATH F424: Applied Stochastic Processes
Department of Mathematics
BITS Pilani, Hyderabad Campus
November - December 2022

Contents

1 Calculating different powers of a square matrix of any given order

1.1 Code

The two functions defined below, `matrix_multiply()` and `matrix_power()` are used to multiply two matrices and raise a square matrix to an integer power, respectively. Note: Both function return -1 in case of errors in dimensions of inputs.

```python
import numpy as np

def matrix_multiply(matA, matB):
    """
    Returns the product of 2 matrics
    i/p: The two matrices to be multipled matA(n1 x n2), matB(n3 x n4)
    o/p: If n2 == n4: returns n1 x n4 product matrix
         else: returns -1
    """
    if matA.shape[1] != matB.shape[0]:
        print("Matrix multiplication invalid")
        return -1
    else:
        result_mat = np.zeros((matA.shape[0], matB.shape[1]))
        for i in range(matA.shape[0]):  # matA.shape[0] = number of rows
            row = matA[i]
            for j in range(matB.shape[1]):  # matB.shape[1] = number of columns
                col = matB[:, j]
                dot = 0
                for k in range(len(row)):
                    dot += row[k] * col[k]
                result_mat[i][j] = dot
        return result_mat

def matrix_power(matA, power):
    """
    Returns the input matrix raised to the power 'power'
    i/p: Matrix matA(n1 x n2), and integer power
    o/p: If n1 == n2 and power is an integer: Returns matA^power
         else: returns -1
    """
    if type(power) is not int:
        print("Power is not a valid integer")
        return -1
    result_mat = matA
```

```python
    for i in range(power - 1):
        result_mat = matrix_multiply(result_mat, matA)
        if type(result_mat) is int:
            return -1
    return result_mat
```

1.2 Implementation

```python
A = np.random.randint(10, size=(3, 3))
print(f"matA:\n {A}")
B = np.random.randint(10, size=(3, 5))
print(f"matB:\n {B}")
```

```
matA:
 [[0 3 4]
 [3 0 5]
 [1 0 1]]
matB:
 [[4 8 8 4 3]
 [6 5 0 5 8]
 [5 5 5 9 8]]
```

```python
print(f'matA x matB: \n{matrix_multiply(A, B)}')
```

```
matA x matB:
[[38. 35. 20. 51. 56.]
 [37. 49. 49. 57. 49.]
 [ 9. 13. 13. 13. 11.]]
```

```python
print(f"(matA)^3: \n{matrix_power(A, 3)}")
```

```
(matA)^3:
[[19. 39. 71.]
 [44. 15. 82.]
 [14.  3. 24.]]
```

Figure 1: Outputs of the multiplication and power functions

2 Regular Matrices and Markov Chains

2.1 Regular Matrices

Stochastic matrices have the following properties;

1. All elements belong in the range $[0, 1]$ since they represent probabilities.
2. The row sum of all elements should be 1 since a row represents all the possible transitions the process can make from any given state.

A stochastic matrix, $\mathbb{P}$ is said to be regular if:

$$\exists \; n > 1 \mid \mathbb{P}^n \text{ has only positive } (> 0) \text{ entries}$$

If the transition matrix of a Markov chain is regular, such a Markov chain is said to be a regular Markov chain has the following properties:

1. $\lim_{n \to \infty} \mathbb{P}^{n+1} = \mathbb{P}^n$
2. The Markov chain attains stable limiting transition probabilities.

2.2 Code

```
def create_stochastic_matrix(size):
    mat = np.zeros((size, size))
    for i in range(size):
        row_sum = 0
        for j in range(size - 1):
            mat[i][j] = np.random.uniform(0, (1 - row_sum))
            row_sum += mat[i][j]
        mat[i][-1] = 1 - row_sum
    return mat

def check_regular_matrix(matA, iterations= 1000):
    for i in range (1, iterations + 1):
        mat = matrix_power(matA, i)
        if (type(mat) == int): return -1 # matrix_power() returns -1 if error
        mat_size = len(mat) # Matrix has to be a square
        count = 0
        for j in range(mat_size):
            for k in range(mat_size):
                if (mat[j][k] == 0):
                    break
                else:
```

```
                count += 1
        if (count == (mat_size ** 2)):
            print(f"The matrix is regular, and, it raised to the power {i} is positive.")
            break
        else:
            print(f"The matrix is not regular upto its {i}th power", end = '\r')
    return
```

2.3 Implementation

```
# Debugging for validity of stochastic matrix
# Generate two random stochastic matrices
for i in range(2):
    b = create_stochastic_matrix(4)
    for i in range(4):
        for j in range(4):
            if (b[i][j] < 0 or b[i][j] > 1):
                print("Not Valid Stochastic Matrix")
    print(b)
    print()
```

```
[[0.1479514  0.69274649 0.09499326 0.06430885]
 [0.60291054 0.22313764 0.09703193 0.07691989]
 [0.38711216 0.37884478 0.13236358 0.10167948]
 [0.52474453 0.12183778 0.26965246 0.08376522]]

[[0.3915436  0.3904747  0.12859628 0.08938542]
 [0.04097717 0.5325034  0.16754051 0.25897893]
 [0.74265276 0.20133344 0.0304146  0.02559919]
 [0.52359309 0.36588025 0.10124252 0.00928413]]
```

Figure 2: Generating stochastic matrices

```python
check_regular_matrix(b)
D = np.array([[0.12351425, 0.50276079, 0.10840787, 0.26531709],
              [0.8681274, 0., 0.11212622, 0.01974638],
              [0.55888506, 0.14277434, 0.15657036, 0.14177024],
              [0.11016633, 0.07586725, 0.40404501, 0.40992142]])
print(f'MatD: \n{D}')
check_regular_matrix(D)
E = np.array([0, 0.5, 0.5, 9.5, 0, 0.5, 0, 1, 0]).reshape(3, -1)
print(f'MatE: \n{E}')
check_regular_matrix(E)
```

```
The matrix is regular, and when it is raised to power 1, it is positive.
MatD:
[[0.12351425 0.50276079 0.10840787 0.26531709]
 [0.8681274  0.         0.11212622 0.01974638]
 [0.55888506 0.14277434 0.15657036 0.14177024]
 [0.11016633 0.07586725 0.40404501 0.40992142]]
The matrix is regular, and when it is raised to power 2, it is positive.
MatE:
[[0.  0.5 0.5]
 [9.5 0.  0.5]
 [0.  1.  0. ]]
The matrix is regular, and when it is raised to power 4, it is positive.
```

```python
F = np.array([[0.7, 0, 0.3],
              [0, 1, 0],
              [0.2, 0, 0.8]]).reshape(3, -1)
print(f"MatF: \n{F}")
check_regular_matrix(F, 1000)
```

```
MatF:
[[0.7 0.  0.3]
 [0.  1.  0. ]
 [0.2 0.  0.8]]
The matrix is not regular upto its 1000th power
```

Figure 3: Checking the functions with random matrices

3 Null Space of a Matrix

3.1 Definition

The null space of a matrix A (r $\times$ n), also known as its kernel, is a subspace of the vector space spanned by A, which is spanned by vectors which are solutions to the homogeneous system of equations:

$$A.\overrightarrow{X} = \overrightarrow{\theta} \tag{1}$$

Where $\overrightarrow{\theta}$ is the 0 vector of dimension n, and is a trivial solution to equation (1).

3.2 Rank-Nullity Theorem

The Rank-Nullity Theorem states that the dimension of any vector space is equal to the rank of a matrix containing coefficients of equations in that vector space (image) and the number of vectors spanning its null space (kernel).
It can be observed from figure 5 that the theorem holds for the given test matrix.
It should be noted that the zero-vector ($\vec{0}$) is a trivial member of the null space, and it is not considered while counting the basis vectors of the kernel.

3.3 Code

```
def rref(matA):
    rref = matA.copy()
    pivot = 0
    rows = len(rref)
    cols = len(rref[0])
    rank = 0
    for r in range(rows):
        if (cols < pivot):
            break
        i = r
        while (rref[i, pivot] == 0):
            i += 1
            if (rows == 1):
                i = r
                pivot += 1
                if (cols == pivot):
                    break
        vec = rref[i]
        rref[i] = rref[r]
        rref[r] = vec
        if (rref[r][pivot] != 0):
            rref[r] = rref[r] / rref[r][pivot]
        for i in range(rows):
            if (i != r):
                rref[i] -= (rref[r]*rref[i][pivot])
        pivot += 1
    for row in range(rows):
        for col in range(cols):
            if (rref[row][col] != 0):
                rank += 1
                break

    return rref, rank
```

The above code finds out the rank and row-reduced echelon form of any given matrix.

3.4 Implementation

```
G = np.array([1, 2, -1, -4, 2, 3, -1, -11, -2, 0, -3, 22]).reshape(3, -1)
print(f'matG: \n{G}')
rref_g, r = rref(G)
print(f"rref(G): \n{rref_g}")
G = sp.Matrix(G)
print (f"sympy.G.rref(): \n{np.array(G.rref()[0])}")
```

```
matG:
[[  1   2  -1  -4]
 [  2   3  -1 -11]
 [ -2   0  -3  22]]
rref(G):
[[ 1  0  0 -8]
 [ 0  1  0  1]
 [ 0  0  1 -2]]
sympy.G.rref():
[[1 0 0 -8]
 [0 1 0 1]
 [0 0 1 -2]]
```

Figure 4: Checking the function that reduces given matrix to echelon form

```
H = np.array([[1, 0, 1, 3],
              [2, 3, 4, 7],
              [-1, -3, -3, -4]]).reshape(3, -1)
print(f"matH:\n{H}")
dimension = len(H[0])
print(f"Dimension(matH): {dimension}")
H = sp.Matrix(H)
rank_H = H.rank()
print(f"Rank(matH): {rank_H}")
null_space = [np.array(mat) for mat in H.nullspace()]
print("H.nullspace(): ")
[print(f'vector_{i+1}:\n{null_space[i]}') for i in range(len(null_space))]
print(f"Rank ({H.rank()}) + Nullity ({len(null_space)}) = Dimension ({dimension})")
```

```
matH:
[[ 1  0  1  3]
 [ 2  3  4  7]
 [-1 -3 -3 -4]]
Dimension(matH): 4
Rank(matH): 2
H.nullspace():
vector_1:
[[-1]
 [-2/3]
 [1]
 [0]]
vector_2:
[[-3]
 [-1/3]
 [0]
 [1]]
Rank (2) + Nullity (2) = Dimension (4)
```

Figure 5: Finding null space of matrix and verifying Rank-Nullity Theorem

References

[1] All the code in this report has been written by me and can be found in this GitHub Repository.

[2] https://en.wikipedia.org/wiki/Rank_nullity_theorem

[3] https://www.math.tamu.edu/~yvorobet/MATH304-504/Lect2-08web.pdf