Sarthak Agarwal

Professor Mehrdad Khosravi

Subject Math 2B

20 June 2019

Efficiency of Determinant Calculation Methods

Abstract: For a square matrix, one can capture relevant information by using just a single number, called the determinant. In mathematics, the concept of determinant comes handy for various purposes like solving linear equations, capturing how linear transformation change the geometry of the system. Studying square matrices is fascinating because of their beautiful and organized symmetry having an equal number of rows and columns. This symmetry enables us to use different methods to manipulate the matrix elements and calculate many essential values, including its determinant. Especially with the extensive use of Linear Algebra and square matrices in the field of computer science, it is imperative to know which of the methods to calculate the determinant is the most efficient way.

Introduction: For the completion of this project, I have used python to compute the determinant of the n*n matrix using three different methods:

1. Checking for a Special Matrix.
    a. Upper Triangular Matrix
    b. Lower Triangular Matrix
    c. Diagonal Matrix
2. Gaussian Elimination

3. Calculating the minor of a matrix using recursion.

After computing the determinant of the matrix, I have calculated the time taken it takes for the script to run each algorithm and then we can compare the time taken for the different processes to come to a conclusion on which method is the fastest and the most efficient of all that we have calculated.

Definition of Determinant: Determinant of (n x n) a matrix and is a real or a complex number that is associated with many properties that the matrix possesses. One way to define the determinant of a (n x n) matrix **A** is the formula:

$$det(A) = \sum_{(i_1 i_2 \cdots i_n)} \pm a_{i_1} a_{i_2} \cdots a_{i_n}$$

where the terms are summed over all permutations, and the sign is positive if the permutation is even, and for odd it is negative.

Working of all Algorithms:

1. **Checking for a Special Matrix**: Upper Triangular, Lower Triangular, and Diagonal matrices are considered to be unique because of their standout symmetry. This outstanding symmetry provides them with exceptional properties. One of these properties being the **determinant equal to the product of elements** that sit on the diagonal of the matrix.

   Algorithm: After the user selects this option from the menu the python script with check if the entered matrix is diagonal, upper triangular or lower triangular, if the kind of inputted matrix is special, we extract the elements that sit on the diagonal and multiply them to find the determinant.

Code:

Checking for LOWER TRIANGULAR MATRIX:

```
def islowertriangular(matrix1):
    for i in range(0, len(matrix1)):
        for j in range(i + 1, len(matrix1)):
            if(matrix1[i][j] != 0):
                return False
    return True
```

Checking for UPPER TRIANGULAR MATRIX:

```
def isuppertriangular(matrix1):
    for i in range(1, len(matrix1)):
        for j in range(0, i):
            if(matrix1[i][j] != 0):
                return False
    return True
```

Checking for Diagonal Matrix: To check if the matrix is a Diagonal MatrixI have check if the inputted matrix is both Upper Triangular and Lower Triangular.

Code for the Main Algorithm:

```
def getdeterminantbySpecialMatrix():
    start = time.time()
    heading1 = Label(root, text = "SPECIAL MATRIX METHOD")
    if (islowertriangular(matrix1) and isuppertriangular(matrix1)):
        label7 = Label(root, text = 'The entered matrix is a DIAGONAL MATRIX \nAnd the diagonal is:\n'+ str(diagonal(matrix2)))
        label8  = Label(root, text = 'Determinant = Product of Matrix Diagonal ='+str(matrix2.diagonal().prod()))
        label7.pack()
        label8.pack()
    elif islowertriangular(matrix1):
        label9 = Label(root,text = 'The entered matrix is a LOWER TRIANGULAR MATRIX \n And the diagonal is:'+str(diagonal(matrix2)))
        label10 = Label(root, text = 'Determinant = Product of Matrix Diagonal ='+str(matrix2.diagonal().prod()))
        label9.pack()
        label10.pack()
    elif isuppertriangular(matrix1):
        label11 = Label(root, text = 'The entered matrix is a UPPER TRIANGULAR MATRIX \n And the diagonal is:'+ str(diagonal(matrix2)))
        label12 = Label(root, text = 'Determinant = Product of Matrix Diagonal ='+str(matrix2.diagonal().prod()))
        label11.pack()
        label12.pack()
    else:
        label13 = Label(root, text = 'The entered matrix is not a special matrix!')
        label13.pack()
    delta1 = time.time() - start
    theLabel6 = Label(root, text="Time taken by the algorithm was:" + str(delta1) +" seconds\n")
    theLabel6.pack()
```

Output:

Output1:

```
The entered Matrix is:
[[1 0 0]
 [0 1 0]
 [0 0 1]]

The entered matrix is a DIAGONAL MATRIX
And the diagonal is:
[1 1 1]

Determinant = Product of Matrix Diagonal =1

Time taken by the algorithm was:0.0022139549255371094 seconds
```

Output 2:

```
The entered Matrix is:
[[1 2 3]
 [1 2 3]
 [2 3 4]]

The entered matrix is not a special matrix!

Time taken by the algorithm was:0.0013210773468017578 seconds
```

2. **Determinant by Gaussian Elimination:** Gaussian Elimination is one of the most important algorithms in the field of both Linear Algebra and Computer Science. The algorithm involves applying simple row transformations rules to a matrix and converting it to the reduced form.

   Row Transformation rules involved in the process:

   - We can multiply any row by any non zero constants.
   - We can switch any two rows.
   - You can add two rows together.

Algorithm: To calculate the determinant using the Gaussian elimination process, we use the Gaussian elimination rules to convert the given matrix to a lower or upper triangular matrix, and after the model is converted to a particular pattern we extract its diagonal and find the product of diagonal elements.

Code:

```
def getMatrixDeterminantGaussianElimination(matrix2):
    start = time.time()
    matrix3 = matrix2
    len1 = len(matrix2)
    len2 = len(matrix2[0])
    for i in range(min(len1, len2)):
        #print (map(lambda y: map(lambda x: "{0:.2f}".format(x), y), matrix2))
        for j in range(i + 1, len1):
            for k in range(i, len2):
                matrix3[j][k] = matrix2[j][k] - (matrix2[j][i] / matrix2[i][i]) * matrix2[i][k]
    matrix4 = matrix(matrix3)
    delta2 = time.time() - start
    theLabel3 = Label(root, text="The matrix after Gaussian elinimation is:\n"+str(matrix4))
    theLabel4 = Label(root, text='Determinant = Product of Matrix Diagonal ='+str(matrix4.diagonal().prod()))
    theLabel5 = Label(root, text="Time efficiency of calculating determinant by Gaussian elimination is:" + str(delta2) +" seconds\n")
    theLabel3.pack()
    theLabel4.pack()
    theLabel5.pack()
```

Output:

The entered Matrix is:
[[1 3 2]
[6 9 1]
[2 3 9]]

The matrix after Gaussian elinimation is:
[[1. 3. 2.]
[0. 9. 1.]
[0. 0. 9.]]

Determinant = Product of Matrix Diagonal =81.0

Time efficiency of calculating determinant by Gaussian elimination is:0.0001270771026611328 seconds

3. **Calculating Determinant of a matrix using recursion:**

- **What is recursion?**
- *Recursion is a process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function.* (Recursion, "GeeksforGeeks.org")

Algorithm: The technique of calculating the determinant using the 'Laplacian' method or the minor expansion method is very famous although the Gaussian Elimination is a more efficient way to calculate the determinant.

In the algorithm that my script uses the recursion technique to find the minor expansion of the matrix and then computing the determinant using the formula:

$$|\mathbf{A}| = \sum_{i=1}^{k} (-1)^{i+j} a_{ij} M_{ij},$$

Where Mij is a so-called minor of A, obtained by taking the determinant of A with row i and column j "crossed out."

Code:

```
def getMatrixMinorRecursion(matrix2,i,j):
    return [row[:j] + row[j+1:] for row in (matrix2[:i]+matrix2[i+1:])]

def getMatrixDeterminantRecursion(matrix2):
    if len(matrix2) == 2:
        return matrix2[0][0]*matrix2[1][1]-matrix2[0][1]*matrix2[1][0]

    determinant = 0
    for k in range(len(matrix2)):
        determinant += ((-1)**k)*matrix2[0][k]*getMatrixDeterminantRecursion(getMatrixMinorRecursion(matrix2,0,k))
    return determinant
```

Output:

```
The entered Matrix is:
[[1 2 3]
 [4 3 2]
 [6 6 6]]

The determinant of matrix by recursive method is:0
Time efficiency of calculating determinant using recursion is:0.0007240772247314453 seconds
```

Comparing Efficiencies:

Efficiency of the algorithm is a very important factor that determines which algorithm is the most suitable for using especially when making humungous programs for the industry. Matrices are used in the Computer Science for calculation purposes all the time and therefore it is important to figure out the efficiency of the determinant calculation algorithm efficiency.

Analysis:

From the above mentioned three methods:

- Calculating the determinant by checking for a special matrix is a reasonably inefficient process, and the fact that it does not apply to every possible model makes this approach the most inefficient way of calculating the determinant matrix.
- Gaussian Elimination is the most efficient process of calculating the determinant. The easy algorithm and time efficiency together make Gaussian Elimination a very reliable process of finding the determinant.
- From the results in the output, it is reasonably evident that calculating the determinant using recursion and minor expansion is not the most efficient process. On average, the Gaussian elimination is almost six times faster than the recursive process.

Works Cited

Daddel, Ali A. “Definition of Determinant.” *Definition of Determinant*, 15 Sept. 2000, www.math.ucdavis.edu/~daddel/linear_algebra_appl/Applications/Determinant/Determinant/node3.html.

“Determinant Expansion by Minors.” *From Wolfram MathWorld*, mathworld.wolfram.com/DeterminantExpansionbyMinors.html.

“Recursion.” *GeeksforGeeks*, 17 Apr. 2019, www.geeksforgeeks.org/recursion/.