An Exploration into Linear Algebra

Yash Money, Imran Iftikar

Idk man, I just work here

1 The Inverse of a Matrick

Ok friends, we are going to be finding the inverse of a matrix today and it will be lots of fun. we will code this in python. so first we will go over what an integer is. ok so basically an integer is a number that isn't really a fraction or something[1]. ok so to solve the inverseof a matrix we will row reduce the matrix until it is the identity matrix and has diagonal of 1s and rest are 0s. and we will perform row operations to do this. but we will do the same row operations on another identity matrix, and whatever that becomes will be the invser[2]. ok thank you so much for watching, next episode we will create self driving car from scratch

```
def inverse(matrix):

'''
finds the inverse of a matrix, uses very similar code to echelon
it takes reduces a matrix to upper triangle and applies the
    same transforms to an indentity matrix
it then reflects both of them, and puts the reflect matrix into
    upper triangle as well, and also
applies the changes to the identity matrix. we reflect again,
    then we divide each each diagonal
and apply that to the identity as well. the original matrix
    becomes an identity matrix,
    array = [
            [1,0,0,0...],
            [0,1,0,0...],
            [0,0,1,0...],
```

[1]the universe et al, basic 1st grade math, 1000BCE oor smth
[2]*ibid.*

```
        [0,0,0,0...1]
    ]
and the identity matrix we initialized becomes the matrices
    inverse. we return the identity matrix

'''

identity = make_identity(len(matrix)) # O(n**2) creates an
    identity matrix

for i in range(2): # O(1), we run through this twice, because
    we upper triangle and reflect twice

    ''' the following code is copypastad from echelon save for
        a few lines
    so consult echelon for more detailed documentation'''

    for col_index in range(len(matrix[0])): #O(n) this first
        for loop handles zeroes that might potentially lead to
        div by 0 errors
        col = get_col(matrix, col_index) # O(n)

        if col_index <= len(matrix): # O(1)

            if all((i == 0) for i in col[col_index:]): #O(n)
                continue

            elif col[col_index] == 0: # O(1)
                for i in range(len(col[col_index:])): #O(n)
                    if col[col_index:][i] != 0: # O(1)
                        row_idx = col_index+i # O(1)
                        break

                matrix[col_index], matrix[row_idx] =
                    matrix[row_idx], matrix[col_index] # O(n)


        for row_index in range(len(col)): # O(n)

            if row_index <= col_index: #O(1)
                if row_index == col_index: #O(1)
                    denominator = matrix[row_index][col_index]
                        #O(1)
```

```
                raw_subtractant_row = matrix[row_index] #O(1)
                raw_subtractant_row_identity =
                    identity[row_index] # O(1) this line is
                    one of the main ones that differs from
                    echelon
            pass

        else:

            numerator = matrix[row_index][col_index] #O(1)

            row_to_sub_from = matrix[row_index] # O(1)
            subtractant = row_by_scalar(raw_subtractant_row,
                (numerator/denominator)) # O(n)
            subbed_row = subtract_row(row_to_sub_from,
                subtractant) # O(1)
            matrix[row_index] = subbed_row


            '''
            the following three lines are mainly what
                differs between this and echelon. it simply
                takes
            the operation we did on the row of the argument
                "matrix" and does it to the row of the
                identity
            '''
            subtractant_identity =
                row_by_scalar(raw_subtractant_row_identity,
                (numerator/denominator))
            subbed_row1 = subtract_row(identity[row_index],
                subtractant_identity)
            identity[row_index] = subbed_row1

    # the following reflects the matrices so that we can
        echelon both again. then it reflects it again so we can
        get back to the original matrix
    matrix = reflect(matrix)
    identity = reflect(identity)

# we divide by 1/the diagonal in each row of the matrix
for i in range(len(matrix)):
    identity[i] = row_by_scalar(identity[i], (1/matrix[i][i]))
```

```
    return identity
```