Lab 8: Algorithm Design Techniques

Manuel Ruvalcaba

CS2302 1:30-2:50

1. Introduction

The purpose of this program is to "discover" the trigonometric identities, given a list of trigonometric functions and to find two subsets of equal value, given a set. For the trigonometric functions, we will use a randomized algorithm technique and for the subsets, we will use backtracking.

2. Proposed Solution Design and Implementation

In order to solve the problem with "discovering" the equalities between different trigonometric functions, I used a single method named *equalities*. It accepts an array *f*, containing the different trigonometric functions, and a tolerance that is set to "0.0001" by default. This method has two *for* loops, one nested in the other, that iterate through the array with the trigonometric functions. Inside the nested loop, there is a condition that the index of the first trigonometric function is less than the index of the second, in order to avoid comparing the same functions twice. There is a variable *same* that is set to true, then the two selected functions are evaluated and compared with different *x* values multiple times. If the difference between both evaluations are greater than the tolerance, *same* is set to *False*. If *same* is still *True* after all the comparisons have been made, an array of size two, containing the functions, is appended to a *result* array. Once the loops are complete, the *result* array, containing the equalities, is returned.

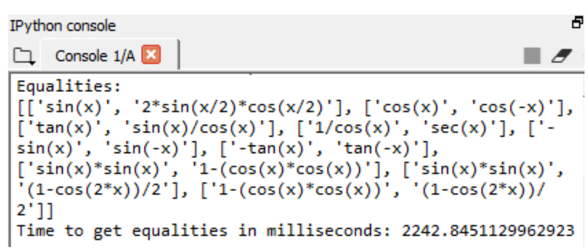
To solve the problem with finding two subsets with equal value from a single set, I utilized two methods named *equalSubsets* and *equalSubsets1*.

The method, *equalSubsets*, receives a set and the length of the set. The sum of the elements in the set is calculated. If the sum is odd, a message is displayed, indicating that there are no equal subsets. Otherwise, the method, *equalSubsets1*, is called. If it returns *True*, the equal subsets are displayed. Otherwise, a message indicating that there are no equal subsets is displayed.

The method, *equalSubsets1*, receives a set, the length of the set, the current sum of the elements in the first subset, the current sum of the elements in the second subset, and an integer *i* for the current index. The method first checks if the entire array has been traversed. If it has, it will check if the sum of the elements of each subset are equal. If they are, it will return *True* and two empty subsets. Otherwise, it will return *False* and two empty subsets. If the whole array has not been traversed, the variables *res*, *sub1*, and *sub2* will equal what is returned by the recursive call to *equalSubsets1*, where the element at position *i* is added to *sum1* and the position is incremented. If it returns *True*, then the element at position *i* will be appended to the first subset. Otherwise, the variables *res*, *sub1*, and *sub2* will equal what is returned by the recursive call to *equalSubsets1*, where the element at position *i* is added to *sum2* and the position is incremented. If it returns *True*, then the element at position *i* will be appended to the second subset. Otherwise, it will return *False* and the two subsets.

3. Experimental Results

To test the code for the equalities, I used an array of different trigonometric functions and tested the equalities with different numbers of tries to see how the runtime changes. I used 250, 500, and 1000 tries.



```
IPython console
Console 1/A
Equalities:
[['sin(x)', '2*sin(x/2)*cos(x/2)'], ['cos(x)', 'cos(-x)'],
['tan(x)', 'sin(x)/cos(x)'], ['1/cos(x)', 'sec(x)'], ['-
sin(x)', 'sin(-x)'], ['-tan(x)', 'tan(-x)'],
['sin(x)*sin(x)', '1-(cos(x)*cos(x))'], ['sin(x)*sin(x)',
'(1-cos(2*x))/2'], ['1-(cos(x)*cos(x))', '(1-cos(2*x))/
2']]
Time to get equalities in milliseconds: 4743.40235001001
```

```
IPython console
Console 1/A
Equalities:
[['sin(x)', '2*sin(x/2)*cos(x/2)'], ['cos(x)', 'cos(-x)'],
['tan(x)', 'sin(x)/cos(x)'], ['1/cos(x)', 'sec(x)'], ['-
sin(x)', 'sin(-x)'], ['-tan(x)', 'tan(-x)'],
['sin(x)*sin(x)', '1-(cos(x)*cos(x))'], ['sin(x)*sin(x)',
'(1-cos(2*x))/2'], ['1-(cos(x)*cos(x))', '(1-cos(2*x))/
2']]
Time to get equalities in milliseconds: 9973.847409011796
```

	Runtime in Milliseconds	Equalities True?
250 Tries	2242.845113	Yes
500 Tries	4743.40235	Yes
1000 Tries	9973.847409	Yes

What is shown is that the runtime doubles every time we double the amount of tries. The equalities discovered are the same throughout all of the test runs.

To test the equal subsets code, I used various different sets with different sizes and recorded how long it took to find the equal subsets.

```
IPython console
Console 1/A
Set:
[2, 4, 5, 9, 12]
Finding equal subsets
Equal Subsets:
[9, 5, 2] [12, 4]
Time to check for subsets in
milliseconds: 0.5073729989817366
```

```
IPython console
Console 1/A
Set:
[2, 4, 5, 9, 13]
Finding equal subsets
There are no equal subsets
Time to check for subsets in
milliseconds: 0.2441769902361557
```

```
IPython console
Console 1/A
Set:
[2, 4, 5, 9, 13, 4, 3, 5, 6, 2, 3]
Finding equal subsets
Equal Subsets:
[5, 3, 9, 5, 4, 2] [3, 2, 6, 4, 13]
Time to check for subsets in
milliseconds: 1.0301680013071746
```

```
IPython console
Console 1/A
Set:
[2, 4, 5, 9, 13, 7, 3, 5, 6, 2, 3]
Finding equal subsets
There are no equal subsets
Time to check for subsets in
milliseconds: 0.24623300123494118
```

	Set	Equal Subsets Exist?	Equal Subsets Found?	Runtime in Milliseconds
Test 1	2,4,5,9,12	Yes	Yes	0.507372999
Test 2	2,4,5,9,13	No	No	0.24417699
Test 3	2,4,5,9,13,4,3,5,6,2,3	Yes	Yes	1.030168001
Test 4	2,4,5,9,13,7,3,5,6,2,3	No	No	0.246233001

It seems that the runtime to find the equal subsets increases when a longer set is processed, but when the set is odd, the runtime is around .25 milliseconds, due to the condition in my method.

4. Conclusion

With this project, I have learned how to use a randomized algorithm to find equalities between different trigonometric functions. I also learned how to use backtracking to find equal subsets in a set. The backtracking method for the subsets is relatively quick with small sets.

I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class.

5. Appendix

```
'''
Course: CS2302 MW 1:30-2:50
Author: Manuel A. Ruvalcaba
Assignment: Lab #8: Algorithm Design Techniques
Instructor: Dr. Olac Fuentes
TA: Anindita Nath, Maliheh Zargaran
Date of Last Modification: May 9, 2019
Purpose of the Program: The purpose of this program is to find 'discover' the
                        the equalities in trignonometric equations and to find
                        two subsets in a set, where the sum of each set are
                        equal to one another.
'''
import math
import timeit
from mpmath import *
from math import *
import random
import numpy as np
def equalities(f, tolerance=0.0001):
    #creates an array of similar functions
    result = []
    for i in range(len(f)):
        for j in range(len(f)):
            if i < j: #used to avoid comparisons that are already made
                same = True
                for h in range(1000):
                    #evaluates the functions to find the similarites
                    x = random.uniform(-math.pi,math.pi)
                    y1 = eval(f[i])
                    y2 = eval(f[j])
                    if np.abs(y1-y2)>tolerance:
                        same = False
```

```
            if same:
                result.append([f[i],f[j]])
    return result

def equalSubsets1(arr, last, sum1, sum2, i):
    #finds two subsets that have an equal sum, if none are found, returns False
    if i == last:
        #checks if the whole set has been traversed
        if sum1 == sum2:
            return True,[],[]
        else:
            return False,[],[]
    res,sub1,sub2 = equalSubsets1(arr, last, sum1 + arr[i], sum2, i + 1)
    if res:
        sub1.append(arr[i])
        return res,sub1,sub2
    res,sub1,sub2 = equalSubsets1(arr, last, sum1, sum2 + arr[i], i + 1)
    if res:
        sub2.append(arr[i])
        return res,sub1,sub2
    return False,sub1,sub2

def equalSubsets(arr,n):
    sumSet = 0
    print('Set:')
    print(arr)
    print('Finding equal subsets')
    for i in range(n):
        sumSet += arr[i]
    if sumSet % 2 != 0:
        #if the sum of the set's elements is odd, there is no equal partition
        print('There are no equal subsets')
    else:
```

```
        #if the sum is even, there is a chance for an equal parition
        s,a,b = equalSubsets1(arr,n,0,0,0)
        if s:
            print('Equal Subsets:')
            print(a,b)
        else:
            print('There are no equal subsets')

#this is where the code is tested

F = ["sin(x)","cos(x)","tan(x)","1/cos(x)","-sin(x)","-cos(x)","-tan(x)","sin(-x)",
     "cos(-x)","tan(-x)","sin(x)/cos(x)","2*sin(x/2)*cos(x/2)","sin(x)*sin(x)",
     "1-(cos(x)*cos(x))","(1-cos(2*x))/2","sec(x)"]

start = timeit.default_timer()
G = equalities(F)
stop = timeit.default_timer()
print("Equalities:")
print(G)
print('Time to get equalities in milliseconds:', (stop - start)*1000)
print()

A = [2,4,5,9,12,14,2,4,8,78]
start = timeit.default_timer()
equalSubsets(A,len(A))
stop = timeit.default_timer()
print('Time to check for subsets in milliseconds:', (stop - start)*1000)
```