Lab #8

CS 2302

Ana Luisa Mata Sánchez

Introduction

This lab's prompt is to write a program to "discover" trigonometric identities. The program should test all combinations of the trigonometric expressions shown in the pdf and use a randomized algorithm to detect the equalities. For equality testing, random numbers in the $-\pi$ to π range must be generated.

The second portion is the partition problem. It consists of determining if there is a way to partition a set of integers S into two subsets S1 and S2 such that $\sum S1 = \sum S2$. Recall that S1 and S2 are a partition of S if and only if $S1 \cup S2 = S$ and $S1 \cap S2 = \{\}$. Write a function that solves the partition problem using backtracking. If a partition exists, the program should display it; otherwise it should indicate that no partition exists. For example, if S = {2, 4, 5, 9, 12}, the program should output the partition S1 = {2, 5, 9} and S2 = {4, 12} and if S = {2, 4, 5, 9, 13} the program should indicate that no partition exists.

Proposed solution design and implementation

Module 1 – identityTries

This method receives the list of trigonometric identities (all of them strings), and the expression to be evaluated (as an index number). It can also receive the number of tries, that is the number of times that the expressions should be compared. The more tries, the more exact the result will be. The last item it can receive is the tolerance, that decides how exact the solutions must match for both expressions to be considered equal.

The method starts by going through the array of expressions, it does not evaluate the current expression evaluated against itself as it is a known fact that they are equal. As it goes through each different expression, it will evaluate the current expression and the expression to be compared for the set number of tries using the eval function and a random number between the range of $-\pi$ to π. After that, it will take the absolute value of the subtraction of one evaluation from the other, if this value surpasses the tolerance level the two expressions are not equal. If the two expressions are not equal, it will set the

Boolean marker “equal” to False and stop the tries loop. After the loops ends or is broken, the method will check the “equal” marker, if it is True it will add that expression as equal to the current expression.

The method will return an array with all expression equal to the current expression.

Module 2 – partition

The method’s purpose it to partition a set in two parts, such that each part has the same summation. It will only receive sets that have a summation that can be divided in two equal parts.

It receives the set, the last index in the set, half of the summation of the set, the first set that is empty and the second set that is a copy of the original set. It first checks if the summation is 0, if it is it returns True, s1 and s2. If the sum is less than 0 or it has gone through the entirety of the set, it means that there is no existent partition and it will return False and both sets.

It will attempt all combination by taking one element, subtracting it from the sum and moving to the next item in the set. If it finds a way for one set to add up to half of the sum, the other set must also add up to this. Meaning that it has found a successful partition. If this is the case, the method will append each item that fits the set to the empty set 1 and take it out from set 2. If it finds that a number does not fit the requirement it will not add it to the set and it will not subtract it from the sum.

Experimental results

identityTries

Method call	Output
F = ['sin(t)', 'cos(t)' , 'tan(t)', 'sec(t)' , '-sin(t)' , '-cos(t)' , '-tan(t)' , 'sin(-t)' , 'cos(-t)' , 'tan(-t)' , 'sin(t)/cos(t)' , '2*sin(t/2)*cos(t/2)', 'sin(t)*sin(t)' , '1 - (cos(t)*cos(t))' , '(1-cos(2*t))/2' , '1/cos(t)'] for i in range(len(F)): print(F[i], " = ", identityTries(F,i), "\n")	sin(t) = ['2*sin(t/2)*cos(t/2)'] cos(t) = ['cos(-t)'] tan(t) = ['sin(t)/cos(t)'] sec(t) = ['1/cos(t)'] -sin(t) = ['sin(-t)'] -cos(t) = [] -tan(t) = ['tan(-t)'] sin(-t) = ['-sin(t)']

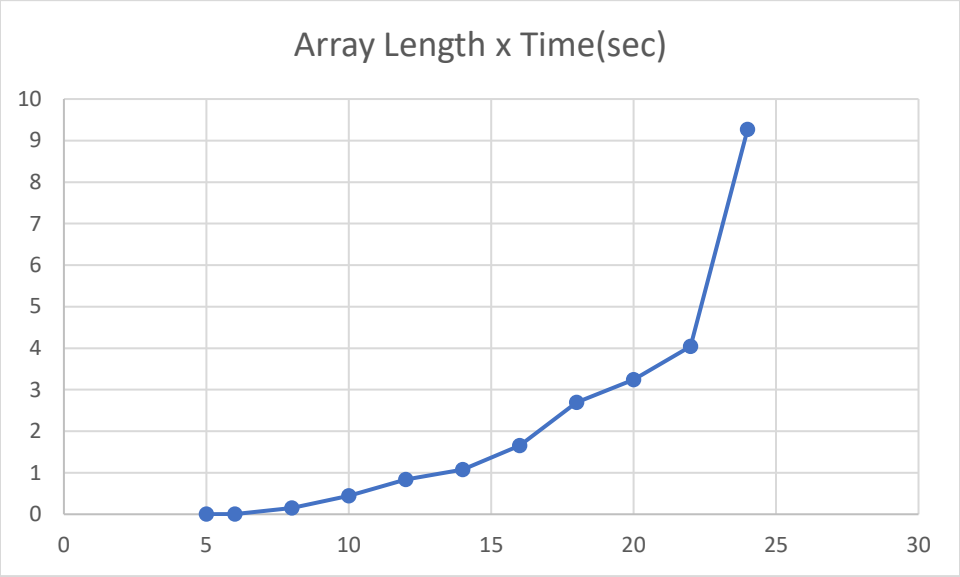
	cos(-t) = ['cos(t)'] tan(-t) = ['-tan(t)'] sin(t)/cos(t) = ['tan(t)'] 2*sin(t/2)*cos(t/2) = ['sin(t)'] sin(t)*sin(t) = ['1 - (cos(t)*cos(t))', '(1-cos(2*t))/2'] 1 - (cos(t)*cos(t)) = ['sin(t)*sin(t)', '(1-cos(2*t))/2'] (1-cos(2*t))/2 = ['sin(t)*sin(t)', '1 - (cos(t)*cos(t))'] 1/cos(t) = ['sec(t)']
F2 = ['sin(t)', 'cos(t)' , 'tan(t)', 'sec(t)' , '-sin(t)' , '-cos(t)' , '-tan(t)', 'sin(-t)' , 'cos(-t)' , 'tan(-t)' , 'sin(t)/cos(t)' , '2*sin(t/2)*cos(t/2)'] for i in range(len(F2)): print(F2[i], " = ", identityTries(F2,i), "\n")	sin(t) = ['2*sin(t/2)*cos(t/2)'] cos(t) = ['cos(-t)'] tan(t) = ['sin(t)/cos(t)'] sec(t) = [] -sin(t) = ['sin(-t)'] -cos(t) = [] -tan(t) = ['tan(-t)'] sin(-t) = ['-sin(t)'] cos(-t) = ['cos(t)'] tan(-t) = ['-tan(t)'] sin(t)/cos(t) = ['tan(t)'] 2*sin(t/2)*cos(t/2) = ['sin(t)']

Running Times

Testing the running time of comparing all expressions to each other without counting the time to print.

Array Length	Time(sec)
4	0.000999928
5	0.001989365
6	0.002988338

8	0.153622389
10	0.443817139
12	0.834736824
14	1.07215929
16	1.656563997
18	2.692813873
20	3.244358063
22	4.04019928
24	9.269181967



partiton

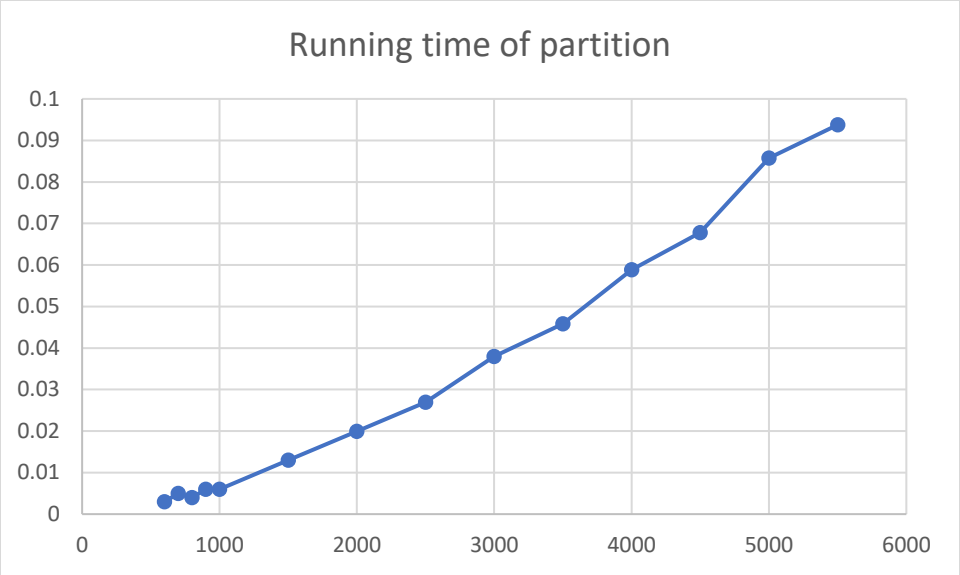
Sets	**Output**
S = [2, 4, 5, 9, 12] S1 = [] S2 = [2, 4, 5, 9, 12] res, s1, s2 = partition(S,len(S)-1,sum(S)/2, S1, S2)	Set 1: [4, 12] Set 2: [2, 5, 9] Time it took to find partition: 0.0
S = [2, 4, 5, 9, 12, 7, 3, 4] S1 = [] S2 = [2, 4, 5, 9, 12, 7, 3, 4]	Set 1: [9, 7, 3, 4] Set 2: [2, 5, 12, 4] Time it took to find partition: 0.0

res, s1, s2 = partition(S,len(S)-1,sum(S)/2, S1, S2)	
S = [2, 4, 5, 9, 12, 7, 3, 4, 10, 5, 5] S1 = [] S2 = [2, 4, 5, 9, 12, 7, 3, 4, 10, 5, 5] res, s1, s2 = partition(S,len(S)-1,sum(S)/2, S1, S2)	Set 1: [2, 4, 3, 4, 10, 5, 5] Set 2: [9, 12, 7, 5] Time it took to find partition: 0.0

Running Times

Testing the running time without printing.

Array Length	Time(s)
600	0.00298119
700	0.00499272
800	0.00399303
900	0.0059824
1000	0.00600696
1500	0.01297021
2000	0.01993155
2500	0.02693152
3000	0.0379312
3500	0.04586601
4000	0.05884743
4500	0.06779885
5000	0.08576202
5500	0.09375429



Conclusion

The more expressions identityTries has to compare, the time increases exponential. The longer the set, the more time it takes to find the partition.

"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."
-Ana Luisa Mata Sánchez

Appendix

```
#   Author: Ana Luisa Mata Sanchez
#   Course: CS2302
#   Assignment: Lab #8
#   Instructor: Olac Fuentes
#   Description: Program to find identities and partitions
#   T.A.: Anindita Nath, Maliheh Zargaran
#   Last modified: 05/09/2019
#   Purpose: Practice randomization and backtracking

import random
```

```
import numpy as np
from mpmath import *
#from math import *

#Method to find the identities of an expression
def identityTries(F,evaluatee,tries=1000,tolerance=0.0001):
    equal = True
    Identities = []
    for i in range(len(F)):
        #To not repeat the expression that is being evaluated
        if i != evaluatee:
            for j in range(tries):
                t = random.uniform(-pi, pi)
                #Evaluate the current expression
                y1 = eval(F[evaluatee])
                #Evaluate one of the other expressions
                y2 = eval(F[i])
                #Check if the results are equal, if not stop evaluating them
                if np.abs(y1-y2)>tolerance:
                    equal = False
                    break
            #If the expressions are equal, add to the identities list
            if equal:
                Identities.append(F[i])
            equal = True
    return Identities

#Method to partition S into two subsets
def partition(S,last,summ, s1, s2):
    #If one of the two sets adds up to half of the sum... This means that the
remaining set has the other half of the sum
    #Meaning sum(s1)==sum(s2)
    if summ == 0:
        return True, s1, s2
    #If it doesn't reach the sum or if we went through the list go back
    if summ<0 or last<0:
        return False, s1, s2

    #Take S[last]
    res, s1, s2 = partition(S,last-1,summ-S[last], s1, s2)

    if res:
        #If S[last] works, add it to the empty set
```

```
        s1.append(S[last])
        #Now remove from the full set
        if S[last] in s2:
            s2.remove(S[last])
        return True, s1, s2
    else:
        #Don't take S[last]
        return partition(S,last-1,summ, s1, s2)

F = ['sin(t)','cos(t)','tan(t)','sec(t)','-sin(t)','-cos(t)','-tan(t)',
'sin(-t)','cos(-t)','tan(-t)','sin(t)/cos(t)','2*sin(t/2)*cos(t/2)',
'sin(t)*sin(t)','1 - (cos(t)*cos(t))','(1-cos(2*t))/2','1/cos(t)']

for i in range(len(F)):
    #Find identities for each of the expressions in the array
    print(F[i], " = ", identityTries(F,i), "\n")

S = [2, 4, 5, 9, 12]
S1 = []
S2 = []
for j in range(len(S)):
    S2.append(S[j])

if sum(S)%2 == 0:
    res, s1, s2 = partition(S,len(S)-1,sum(S)/2, S1, S2)
    if sum(s1) == sum(s2) and res:
        print("Set 1: ", s1,"\nSet 2: ", s2)
    else:
        print("No partition exists")
else:
    print("No partition exists")
```