Tyler Salas

1:30-2:50 Class

Dr. Fuentes

80596221

Lab 8 Report

Lab 8, the lab assigned this week, aimed to implement some of the five different methods of programming. For one part, we were required to test the trigonometric identity of different equations that would be inputted into the program. For the second part, we were required to take a subset S and determine if there existed a partition of S so the sum of each subset equaled each other and the union of the two subsets was equal to S.

**Discovering Trigonometric Identities**

From what I understood in the lab instructions on the class website, this portion of the lab was supposed to receive an equation in terms of a variable x and determine the trigonometric identity of the equation.

The program I wrote includes a lot of repeating the same algorithm but using a different equation to insert, it also comes with a lot of declaring the different trigonometric identities for use. The main method starts by declaring the trigonometric equations that are listed in the lab instructions. From here the main method prints each equation and prints the equal(x) method after each equation where x is the different equations to input.

In the equal method, the code starts by declaring all the trigonometric equations in the lab instructions. From there a method called listMaker() is called. In the list maker function, the program creates a list that contains all the trigonometric identities that can possibly be detected by the program. This includes things like sin(t), cos(t), sec(t) and others that can be obtained.

The program then returns to the equal method where a for loop is started. The for loop will run in range of the variable tries that is always declared to be 1000. The higher number this variable is made, the more accurate results of the program will be. A variable t is declared as a random number from –pi to pi, and this variable will be used to evaluate all of the equations in one loop. T is set to a new random float point in each loop. Next a variable y1 is set to be the eval(f1) where f1 is the equation that is to be questioned in its identity. The program then sets a variable y2 equal to the evaluation of each possible trigonometric identity. If the absolute value of y1-y2 is less than the tolerance (a variable declared in the beginning that sits at .0001. The lower this number the more accurate the program will be.) then that means that this identity is the one to be found, and a variable det is set to the string variable of y2. A variable add count is called with the list of identities that holds an int number next to its trigonometric identity, one is added to the respective identity.

From here a method called detIden() is returned and called with the list of identities. What this method does is iterate through the numbers in the identity list and returns the max number found. The max number means that this is the trigonometric identity to be found. And in finding the max number, the identity is returned. It is then printed in the main method.

Output

Equation: sin(t)

Program's Determined Trigonometric Identity: sin(t)

Equation: cos(t)

Program's Determined Trigonometric Identity: cos(t)

Equation: tan(t)

Program's Determined Trigonometric Identity: tan(t)

Equation: -sin(t)

Program's Determined Trigonometric Identity: -sin(t)

Equation: -cos(t)

Program's Determined Trigonometric Identity: -cos(t)

Equation: -tan(t)

Program's Determined Trigonometric Identity: -tan(t)

Equation: sin(t)/cos(t)

Program's Determined Trigonometric Identity: tan(t)

Equation: (2\*sin(t/2))\*(cos(t/2))

Program's Determined Trigonometric Identity: sin(t)

Equation: sin(t)\*sin(t)

Program's Determined Trigonometric Identity: sin(t)\*sin(t)

Equation: 1 - (cos(t)\*cos(t))

Program's Determined Trigonometric Identity: sin(t)\*sin(t)

Equation: (1 - cos(2\*t))/2

Program's Determined Trigonometric Identity: sin(t)\*sin(t)

Equation: 1/cos(t)

Program's Determined Trigonometric Identity: sec(t)

As we can see in the results the program yields, the simpler trigonometric identities that are entered such as sin(t) or cos(t) are equal to themselves. Though the more complex ones such as 1/cos(t) are returned as their identity being sec(t) which is exactly what the identity of that equation is. Another example being (1 - cos(2\*t))/2 which is equal to sin(t) squared or sin(t)\*sin(t)

RunTimes

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test 1 | Test 2 | Test 3 |
| 500 Tries | 0.9427661895751953 | 0.9429869651794434 | 0.9430079460144043 |
| 1000 Tries | 1.989344596862793 | 1.9498422145843506 | 1.890965461730957 |
| 1500 Tries | 2.8244595527648926 | 2.9368152618408203 | 2.848506450653076 |

As we can see in the runtime results, the variation in running times between tries of the same number seem to remain nearly the same. This is due to the fact that the for loop to determine the identities runs O(1).

**Equal Subsets**

The instructions for this lab said that we had to write a program that takes a subset and returns if a partition in the subset exists such that s1 = s2 and the union of the tow subset is equal to S. I could not figure out how to do this method with backtracking though I could only think of the dynamic programming type solution to solve the problem.

The method starts in the main method by just declaring two different sets and the function subsetEqual is evaluated in each set and if the return statement is not None then the two subsets are printed. (The output will be none if no partition exists that follows the rules.)

The subsetEqual() method starts by finding the sum of the subset S that is declared in the main call of the method, it then checks if the sum of S is odd. If so, the method returns that that there is no partition because the subsets sum must be even. The method then sets a variable possible sum to the sum of the set divided by 2. This variable is used in the dynamic programming table to be used as the max number of what the set numbers should equal.

A list named pastCalc creates an mxn list of boolean variables all set to false. This is the dynamic programming table that will be used to store the already calculated variable. (This is similar to the coin table we did in class for subset sum) The program then sets the 0 row and 0 column to true as these will always be obtainable by just taking zero coins or reaching the sum of 0 coins. The program then fills out the remainder of the table placing true in the place of the already false variable if there exists a sum that can be obtained from the coins being counted at that point in the list. The program then checks if the element in the bottom right corner is false. If so, then no partition exists, and the program declares so.

Two lists s1 and s2 are declared so that the respective elements could be added to their partitions. The program runs a while loop that finds the elements that equal to the sum that makes the elements of each subset equal to each other. If the number doesn’t contribute to the sum that is needed it is added to s2, else it is added to s1. Once the loop ceases by reaching the last row and column, then the program returns s1 and s2. The program then returns to the me=ain method evaluating so.

Output

The subset S is: [4, 5, 9, 2, 12]

The two partitions in S are:

([2, 9, 5], [4, 12])

The subset S is: [4, 2, 5, 13, 9]

No partition exists

RunTimes

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test 1 | Test 2 | Test 3 |
| 5 Elements | 0.0039951801300048 | 0.0039658748977968 | 0.0039976756538790 |
| 10 Elements | 0.0045897548905720 | 0.0048985742097902 | 0.0041213984097159 |
| 15 Elements | 0.0059878093576908 | 0.0054959958720994 | 0.0054876987059870 |

As we can see, the runtimes for the dynamic programming type code is fairly efficient in its runtime.

In conclusion this lab taught me a lot in terms of different ways to approach different ways in programming things. This entire class has been very helpful to me.

#CS2302

#Tyler Salas

#Lab8

#Dr.Fuentes

#Anindita Nath

#Detects trigometric identities

import matplotlib.pyplot as plt

import numpy as np

import random

import time

from math import \*

import math

#Creates a list that holds identities and its count

def listMaker():

identity = [[] for i in range(9)]

a = "sin(t)"

b = "cos(t)"

c = "tan(t)"

d = "sec(t)" #Secant = 1/cos(t)

e = "-sin(t)"

f = "-cos(t)"

g = "-tan(t)"

h = "sin(−t)" # -sin(t)

i = "cos(−t)" #cos(t)

j = "tan(−t)" #-tan(t)

k = "sin(t)/cos(t)" #tan(t)

l = "(2\*sin(t/2))\*(cos(t/2))" #sin

m = "sin(t)\*sin(t)"

n = "1 - (cos(t)\*cos(t))" # sin^2

o = "(1 - cos(2\*t))/2" #Equal to sin^2

p = "1/cos(t)"

identity[0].append(a)

identity[1].append(b)

identity[2].append(c)

identity[3].append(d)

identity[4].append(e)

identity[5].append(f)

identity[6].append(g)

identity[7].append(h)

identity[8].append(m)

for i in range(len(identity)):

identity[i].append(0)

return identity

#adds one to appropriate point in list holding identites

def addCount(identities,det):

a = "sin(t)"

b = "cos(t)"

c = "tan(t)"

e = "-sin(t)"

f = "-cos(t)"

g = "-tan(t)"

h = "sin(−t)" # -sin(t)

m = "sin(t)\*sin(t)"

d = "1/cos(t)"

if det == a:

identities[0][1] += 1

if det == b:

identities[1][1] += 1

if det == c:

identities[2][1] += 1

if det == d:

identities[3][1] += 1

if det == e:

identities[4][1] += 1

if det == f:

identities[5][1] += 1

if det == g:

identities[6][1] += 1

if det == h:

identities[7][1] += 1

if det == m:

identities[8][1] += 1

#iterates through list to determine which identity had the highest number of matches

def detIden(identities):

max = identities[0]

for i in range(len(identities)):

if identities[i][1] >= max[1]:

max = identities[i]

#print(max)

return max

#Program that evaluates the trigometric identity of a function

def equal(f1,tries=1000,tolerance=0.0001):

det = ""

a = "sin(t)"

b = "cos(t)"

c = "tan(t)"

d = "sec(t)" #Secant = 1/cos(t)

e = "-sin(t)"

f = "-cos(t)"

g = "-tan(t)"

h = "sin(−t)" # -sin(t)

i = "cos(−t)" #cos(t)

j = "tan(−t)" #-tan(t)

k = "sin(t)/cos(t)" #tan(t)

l = "(2\*sin(t/2))\*(cos(t/2))" #sin

m = "sin(t)\*sin(t)"

n = "1 - (cos(t)\*cos(t))" # sin^2

o = "(1 - cos(2\*t))/2" #Equal to sin^2

p = "1/cos(t)"

identity = listMaker()

#For Loop to determine the trignometric identity according to above identities

for i in range(tries):

t = random.uniform(-math.pi,math.pi)

y1 = eval(f1)

y2 = eval(a)

if np.abs(y2-y1) <= tolerance:

det = a

y2 = eval(b)

if np.abs(y2-y1) <= tolerance:

det = b

y2 = eval(c)

if np.abs(y2-y1) <= tolerance:

det = c

y2 = eval(p)

if np.abs(y2-y1) <= tolerance:

det = p

y2 = eval(e)

if np.abs(y2-y1) <= tolerance:

det = e

y2 = eval(f)

if np.abs(y2-y1) <= tolerance:

det = f

y2 = eval(g)

if np.abs(y2-y1) <= tolerance:

det = g

y2 = eval(e)

if np.abs(y2-y1) <= tolerance:

det = e

y2 = eval(f)

if np.abs(y2-y1) <= tolerance:

det = f

y2 = eval(g)

if np.abs(y2-y1) <= tolerance:

det = g

y2 = eval(c)

if np.abs(y2-y1) <= tolerance:

det = c

y2 = eval(a)

if np.abs(y2-y1) <= tolerance:

det = a

y2 = eval(m)

if np.abs(y2-y1) <= tolerance:

det = m

y2 = eval(n)

if np.abs(y2-y1) <= tolerance:

det = n

y2 = eval(o)

if np.abs(y2-y1) <= tolerance:

det = m

y2 = eval(p)

if np.abs(y2-y1) <= tolerance:

det = p

addCount(identity,det)

#print(det)

#print(identity)

return detIden(identity)[0]

t = 5

a = "sin(t)"

b = "cos(t)"

c = "tan(t)"

d = "sec(t)" #Secant = 1/cos(t)

e = "-sin(t)"

f = "-cos(t)"

g = "-tan(t)"

h = "sin(−t)" # -sin(t)

i = "cos(−t)" #cos(t)

j = "tan(−t)" #-tan(t)

k = "sin(t)/cos(t)" #tan(t)

l = "(2\*sin(t/2))\*(cos(t/2))" #sin

m = "sin(t)\*sin(t)"

n = "1 - (cos(t)\*cos(t))" # sin^2

o = "(1 - cos(2\*t))/2" #Equal to sin^2

p = "1/cos(t)"

start = time.time()

print("Equation:",a)

print("Program's Determined Trigonometric Identity: ",equal(a))

print()

print("Equation:",b)

print("Program's Determined Trigonometric Identity: ",equal(b))

print()

print("Equation:",c)

print("Program's Determined Trigonometric Identity: ",equal(c))

print()

print("Equation:",e)

print("Program's Determined Trigonometric Identity: ",equal(e))

print()

print("Equation:",f)

print("Program's Determined Trigonometric Identity: ",equal(f))

print()

print("Equation:",g)

print("Program's Determined Trigonometric Identity: ",equal(g))

print()

print("Equation:",k)

print("Program's Determined Trigonometric Identity: ",equal(k))

print()

print("Equation:",l)

print("Program's Determined Trigonometric Identity: ",equal(l))

print()

print("Equation:",m)

print("Program's Determined Trigonometric Identity: ",equal(m))

print()

print("Equation:",n)

print("Program's Determined Trigonometric Identity: ",equal(n))

print()

print("Equation:",o)

print("Program's Determined Trigonometric Identity: ",equal(o))

print()

print("Equation:",p)

print("Program's Determined Trigonometric Identity: ",equal(p))

print()

end = time.time()

print(end-start)

#CS2302

#Tyler Salas

#Lab8

#Dr.Fuentes

#Anindita Nath

#Solves subset partition problem

import time

def subsetEqual(S,last):

#Sums all the elements of S

sumArray = sum(S)

#Checks if the sum is odd if so partitioning is impossible

if sumArray % 2 == 1:

print(" No partition exists")

return

#Used to estimate the possible sum and to use as last element in pastCalc list

posSum = sumArray//2

#Stores calculations already made by program (coins diagram we did in class)

pastCalc = [[False for i in range(posSum+1)] for i in range(last+1)]

#Sets all zero points to false

for i in range(1, posSum+1):

pastCalc[0][i] = False

#Sum of 0 is always True

for i in range(last + 1):

pastCalc[i][0] = True

# Fills out the pastCalc table storing their true or false value (That sum is obtainable with set numbers)

for i in range(1, last + 1):

for j in range(1, posSum + 1):

pastCalc[i][j] = pastCalc[i-1][j]

if S[i - 1] <= j:

pastCalc[i][j] = pastCalc[i][j] or pastCalc[i - 1][j - S[i - 1]]

#If theres no s1=s2 no partition exists

if pastCalc[last][posSum] == False:

print(" No partition exists")

return

s1, s2 = [], []

#While loop to determine elements of subset

while last>= 0 and posSum >=0:

# if the current number in the subset doesnt contribute to reaching the possible sum and it is true it is put into s2

if pastCalc[last - 1][posSum]:

last -= 1

s2.append(S[last])

# else if it is true and not put into s2 it is put into s1

elif pastCalc[last - 1][posSum - S[last - 1]]:

last -= 1

posSum -= S[last]

s1.append(S[last])

return s1,s2

start = time.time()

S = [4,5,9,2,12,14,7,11,14,15,17]

print("The subset S is:",S)

if subsetEqual(S,len(S)-1) is not None:

print("The two partitions in S are:")

print(" ",subsetEqual(S,len(S)-1))

end = time.time()

print(end-start)

S = [4,2,5,13,9]

print("The subset S is:",S)

if subsetEqual(S,len(S)-1) is not None:

print("The two partitions in S are:")

print(" ",subsetEqual(S,len(S)-1))

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.



-Tyler Salas