Design Document Assignment 2

Abhinav Prasanna

October 8, 2021

Introduction

The core concept of this assignment is to understand the fundamentals of machine learning and introduce fundamental math concepts. We need to code multiple C files including E,Madhava, Euler, BBP, Viete and Newton. We also are required to code a mathlib-test that are responsible to test the functionality of all the C files.

E.c

E.c is a program responsible for approximating the value of E using Taylor Series. We are required to code e() and eterms(). E() will perform the primary computation and ETerms() will $\sum_{n=0}^{\infty} \frac{1}{k!}$ We need to computer this equation in C.

We should use a for loop to compute the summation notation. We can not run our for loop into infinity as that would generate a continuous loop so we will need to set a finite end.

We can do this by checking where the equation converging. We can specifically look at where the equation will converge into ϵ. We will see which term converge at ϵ and take the sum from the beginning term to that term, As a result, we should have calculated our e term in Taylor Series. We also need to compute the factorial in C which will require another for loop.

Pseudocode

```
for (int k; lastterm > epsilon; k++){
     for(int j; j<k;j++){
          factorial*=j
          }

     sum += term
     lastterm = term
     terms++
}
return sum and terms
```

Madhava.c

Madhava.c is a program responsible for approximating π using the Madhava Series. We also need a function to track the number of computed terms with a static variable. $\sum_{k=0}^{\infty} \frac{-3^{-k}}{2k+1}$
We need the functions pimadhava() and pimadhavaterms() where pimadhava approximates the value of π using madhava series and pidmadhavaterms tracks the number of computed terms with a static variable.. We can use a for loop to take a summation but we will need a finite term to end. This is where we need to check if the last term converges onto ϵ. We will need to compute the numerator and denominator separately in C. We will also need a for loop to calculate the -3^{-k}

Pseudocode

```
for (int k; lastterm > epsilon || -last term > epsilon; k++){
     for(int j ; j<k ; j++){
      term*=-3
     }
     numerator = 1/term
     denominator = 2 * k + 1
     lastterm = numerator/denominator
     sum += lastterm
     terms++
}
sum = sqrt(12) * sum
return sum and terms
```

Euler.c

Euler.c is a program responsible for approximating π using the Euler Series. We also need a function to track the number of computed terms with a static variable. $\sum_{k=1}^{\infty} \frac{1}{k!}$
We need the functions pieuler() and pieulerterms() where pieuler approximates the value of π using Euler series and pieulerterms tracks the number of computed terms with a static variable.. We can use a for loop to take a summation but we will need a finite term to end. This is where we need to check if the last term converges onto ϵ.

Pseudocode

```
for (int k; lastterm > epsilon ; k++){
     lastterm = 1/k * k
     sum += lastterm
     terms++
}
return sum and terms
```

bbp.c

BBP.c is a program responsible for approximating π using the Bailey-Borwein-Plouffe Formula. We also need a function to track the number of computed terms with a static variable. $\sum_{k=0}^{\infty} 16^{-k} \frac{k(120k+151)+47}{(k(k(512k+1024)+712)+194)+15)}$
We need the functions pibbp() and pibbpterms() where pibbp approximates the value of π using BBP series and pibbpterms tracks the number of computed terms with a static variable. We can use a for loop to take a summation but we will need a finite term to end. This is where we need to check if the last term converges onto ϵ. We need to use a for loop for power function.

Pseudocode

```
for (int k; lastterm > epsilon ; k++){
    for(int j;j<k;j++){
         power*=16;
    }
    coefficient = 1/power
    numerator = k * ( 120 * k + 151) + 47
    denomonator = k * ( k * ( k * ( 512 * k + 1024) + 712 ) + 194 ) + 15
    lastterm = coefficient * ( numerator / denomonator)
    sum+= lasterm
    terms++
}
return sum and terms
```

viete.c

Viete.c is a program responsible for approximating π using the Viete Formula. We also need a function to track the number of computed terms with a static variable. $\prod_{i=1}^{\infty} \frac{ai}{2}$ We need the functions piviete() and pivieteterns() where piviete approximates the value of π using Viete series and pivieteterms tracks the number of computed terms with a static variable.. We can use a for loop to take a summation but we will need a finite term to end. This is where we need to check if the last term converges onto ϵ.

Pseudocode

```
for (int k; lastterm > epsilon ; k++){
    lasterm = k/2
    sum *= lasterm
    terms++
}
return sum and terms
```

newton.c

Viete.c is a program responsible for approximating the Newton-Raphson method and track the number of iterations taken. We also need a function to track the number of computed terms with a static variable.. $x_{k+1} = x_{k+1} - \frac{f(x_k)}{f'(x_k)}$ We need the functions sqrtnewton() and sqrtnewtonitters() where sqrtnewton approximates the value of π using Newton-Raphson method and sqrtnewtoniters tracks the number of computed terms with a static variable. We can use the sqrt function below to emulate Newton- Raphson Method.

PseudoCode

```
sqrt(x)
x=0
y=1
 while(abs(y-z) > EPSILON
   z=y
   y = 0.5 * (z+x/z)
 return y
```

mathlib-test.c

MathLib-test.c contains different test cases for our c files. The a flag should run all tests. The e flag should run approximation tests. The b flag should run Bailey Borwein Plouffe π approximation test. The m flag runs Madhava π approximation test. The r flag runs Euler sequence π approximation test. The v flag runs Viete sequence π approximation test. The n flag runs Newton-Raphson Square Root approximation tests. The s flag enable printing of statistics to see

computed terms and factors for each tested function. The h flag displays a help message detailing program usage.

PseudoCode

```
switch
case(a)
run tests
case(e)
run approximation tests
case(b)
run bbp.c
case(m)
run madhava.c
case(r)
run euler.c
case(v)
run viete.c
case(n)
run newton.c
case(s)
print all c files
case(h)
print help message
```