Assignment 2 – A Little Slice of Pi

Rahul Amudhasagaran

CSE 13S – Spring 2023

Purpose

This program serves to use famous mathematical models and equations to accurately approximate values. In this case, the program is used to calculate pi.

How to Use the Program

In order to use the program, run the file while choosing which options to select. The options are listed below.

```
Options:
-a: runs all tests
-e: runs the e approximation tests
-b: runs the Bailey-Borwein-Plouffe pi approximation test
-m: runs the Madhava pi approximation test
-r: runs the Euler sequence pi approximation test
-v: runs the Viete pi approximation test
-w: runs the Wallis pi approximation test
-n: runs the Newton-Raphsen square root approximation tests (no inputs needed)
-s: enable printing the statistics of all functions, including factors and terms
-h: displays the help message (will display if no option is given)
```

No test will be run twice. The -s runs for all functions that were chosen to run.
These descriptions were given by the Lab Document [1].

Program Design

There are two parts to this program: the part that handles inputs and the multitude of functions/tests that give out the result. The input section is in the file mathlib-test.c while the functions are in their respective files. Each function file contains their respective mathematical model and the tools to assist them running. The Mathlib-test file contains the input handler for the program as well as the statistics enabler and the help message functions.

Data Structures

I used one data structure in this code. This array contained the Booleans for if a specific option was chosen or not. I used this data structure just as a collection of Booleans.

[1] Assignment 2 A Little Slice of Pi by Prof. Darrell Long edited by Kerry Veenstra and Jess Srinivas[1]

Algorithms

There are multiple functions that use the same loop format because most of them are series-based. The general loop format pseudo code is provided below:

```
Loop Format
    for k = starting index (0 or 1) and check if term ever < EPSILON and iterate k++
        sum += term
        term modified depending on option
```

Function Descriptions

While each mathematical equation is located in a different file, for simplicity's sake I will describe the functions in groups.

Taylor Series for e Approximation:

```
Formula: Sum [0, inf) 1/k!

e ():
Inputs: Epsilon Value
Outputs: Approximation of e
Purpose: This function calculates an approximation of e using Taylor Series.

e_terms ():
Inputs: None
Outputs: Number of terms used to calculate e
Purpose: It tracks the number of terms used to calculate e using Taylor Series.

Psuedocode:

static int ecnt
double e (void)
    double sum = 0, term = 1, den1 = 1
    for (ecnt = 1; absolute (term) >= EPSILON; ++ecnt)
        sum += term
        den1 *= ecnt
        term = 1 / den1
    return sum

int e_terms(void)
    ecnt--
    return ecnt
```

Bailey-Borwein-Plouffe Formula for pi:

```
Formula: Sum [0, inf) 16^-k * (4 / (8k + 1) - 2 / (8k + 4) - 1 / (8k + 5) - 1 / (8k - 6)

pi_bbp ():
Inputs: Epsilon Value
Outputs: Approximation of pi
Purpose: It calculates an approximation of pi using Bailey-Borwein-Plouffe Formula.
```

```
pi_bbp_terms ():
Inputs: None
Outputs: Number of terms used to calculate pi
Purpose: It tracks the number of terms used to calculate pi using Madhava Series.

Pseudocode:

static int bbpcnt
double pi_bbp (void)
    double sum = 0, term = 1, num1 = 1, den1 = 1, den2 = 1 / 16
    for (bbpcnt = 0; absolute (term) >= EPSILON; ++bbpcnt)
        num1 = bbpcnt * (120 * bbpcnt + 151) + 47
        den1 = bbpcnt * (bbpcnt * (bbpcnt * (512 * bbpcnt + 1024) + 712) + 194) + 15
        den2 *= 16
        term = num1 / (den1 * den2)
        sum += term
    return sum
int pi_bbp_terms (void)
    return bbpcnt
```

Madhava Series for pi:

```
Formula: sqrt (12) * Sum [0, inf) ((-3)^-k) / (2k + 1)

pi_madhava ():
Inputs: Epsilon Value
Outputs: Approximation of pi
Purpose: It calculates an approximation of pi using Madhava Series.

pi_madhava_terms ():
Inputs: None
Outputs: Number of terms used to calculate pi
Purpose: It tracks the number of terms used to calculate pi using Madhava Series.

Pseudocode:

static int mdhvcnt
double pi_madhava (void)
    mdhvcnt = 0
    double term = 1, sum = 0, den1 = 1, den2 = 1
    while (absolute (term) >= EPSILON)
        term = 1 / (den1 * den2)
        den1 *= -3
        den2 += 2
        sum += term
        mdhvcnt += 1
    sum *= sqrt_newton (12.0)
    return sum
int pi_madhava_terms (void)
    return mdhvcnt
```

Euler Solution for pi:

```
Formula: sqrt (6 * Sum [1, inf) 1 / k^2)
```

```
pi_euler ():
Inputs: Epsilon Value
Outputs: Approximation of pi
Purpose: It calculates an approximation of pi using Euler Solution.

pi_euler_terms ():
Inputs: None
Outputs: Number of terms used to calculate pi
Purpose: It tracks the number of terms used to calculate pi using Euler Solution.

Pseudocode:

static int elrcnt
double pi_euler (void)
    double sum = 0, term = 1
    for (elrcnt = 1; absolute (term) > EPSILON; elrcnt++)
        term = 1 / elrcnt
        term = term / elrcnt
        sum += term
    sum *= 6
    sum = sqrt_newton (sum)
    return sum
int pi_euler_terms (void)
    elrcnt--
    return elrcnt
```

Viete Formula for pi:

```
Formula: 2 / (prod [1, inf) + a / 2 where a is sqrt (2) pattern

pi_viete ():
Inputs: Epsilon Value
Outputs: Approximation of pi
Purpose: It calculates an approximation of pi using Viete Formula.

pi_viete_factors ():
Inputs: None
Outputs: Number of factors used to calculate pi
Purpose: It tracks the number of terms used to calculate pi using Viete Formula.

Psuedocode:

static int vtcnt
double pi_viete (void)
    double prod = 1, factor = 0, num1 = 0, den1 = 2
    for (vtcnt = 1; absolute (1 - factor) >= EPSILON; ++vtcnt)
        num1 = sqrt_newton (2 + num1)
        factor = num1 / den1
        prod *= factor
    prod = 2 / prod
    return prod
int pi_viete_factors (void)
```

```
    return vtcnt - 1
```

Wallis Series for pi:

```
Formula: 2 Prod [1, inf) 4k^2 / (4k^2 - 1)

pi_wallis ():
Inputs: Epsilon Value
Outputs: Approximation of pi
Purpose: It calculates an approximation of pi using Wallis Series.

pi_wallis_factors ():
Inputs: None
Outputs: Number of factors used to calculate pi
Purpose: It tracks the number of terms used to calculate pi using Wallis Series.

Pseudocode:

static int wllscnt
double pi_wallis (void)
    double prod = 1, factor = 0, num1 = 1, den1 = 1
    for (wllscnt = 1; absolute (1 - factor) >= EPSILON; ++wllscnt)
        num1 = 4.0 * wllscnt * wllscnt
        den1 = 4.0 * wllscnt * wllscnt - 1.0
        factor = num1 / den1
        prod *= factor
    prod *= 2
    return prod
int pi_wallis_factors (void)
    return wllscnt - 1
```

Newton's Method for Square Root:

```
sqrt_newton ():
Inputs: double x
Outputs: Approximation of square root of x
Purpose: It calculates an approximation of the square root of x using Newton's Method.

sqrt_newton_iters ():
Inputs: None
Outputs: Number of terms used to calculate square root of x
Purpose: It tracks the number of terms used to calculate sqrt x using Newton's Method.

Psuedocode:

static int nwtncnt
double sqrt_newton(double x)
    nwtncnt = 0
    double next_y = 1, y = 0
    while (absolute (next_y - y) >= EPSILON)
        y = next_y
        next_y = 0.5 * (y + x / y)
```

```
        nwtncnt += 1
    return next_y
int sqrt_newton_iters (void)
    return nwtncnt
```

Mathlib-test:

```
main ():
Inputs: None
Outputs: Function to run
Purpose: It handles the user input and chooses which function to run.

help ():
Inputs: None
Outputs: None
Purpose: It prints help message.

stats ():
Inputs: None
Outputs: None
Purpose: Enables statistics for functions.

Pseudocode:

#define OPTIONS "aebmrvwnsh"

void help (void)
int main(int argc, char **argv)
    int opt = 0, stats = 0
    int selected_options [7] = {0}
    while ((opt = getopt (argc, argv, OPTIONS)) != -1)
        switch (opt)
        case 'a'
            for (int j = 0; j < 7; ++j)
                selected_options [j] = 1
            break
        case 'e': selected_options [0] = 1
        break
        case 'b': selected_options [1] = 1
        break
        case 'm': selected_options [2] = 1
        break
        case 'r': selected_options [3] = 1
        break
        case 'v': selected_options [4] = 1
        break
        case 'w': selected_options [5] = 1
        break
        case 'n': selected_options [6] = 1
        break
        case 's': stats = 1
        break
        case 'h': help ()
        return 0
```

```
        default: help ()
        return 0

    int sum = 0
    for (int k = 0; k < 7; ++k)
        sum += selected_options [k]

    if (sum == 0)
        help ()
        return 0

    if (selected_options [0])
        printf("e() = %16.15lf, M_E = %16.15lf, diff =
        %16.15lf\n", e (), M_E, M_E - e ())
        if (stats)
            printf("e() terms = %d\n", e_terms ())

    if (selected_options [1])
        printf("bbp() = %16.15lf, M_PI = %16.15lf, diff =
        %16.15lf\n", pi_bbp (), M_PI,
            M_PI - pi_bbp ())
        if (stats)
            printf("bbp() terms = %d\n", pi_bbp_terms ())

    if (selected_options [2])
        printf("madhava() = %16.15lf, M_PI = %16.15lf, diff =
        %16.15lf\n", pi_madhava (), M_PI,
            M_PI - pi_madhava ())
        if (stats)
            printf("madhava() terms = %d\n", pi_madhava_terms ())

    if (selected_options [3])
        printf("euler() = %16.15lf, M_PI = %16.15lf, diff =
        %16.15lf\n", pi_euler (), M_PI,
            M_PI - pi_euler ())
        if (stats)
            printf("euler() terms = %d\n", pi_euler_terms ())

    if (selected_options [4])
        printf("viete() = %16.15lf, M_PI = %16.15lf, diff =
        %16.15lf\n", pi_viete (), M_PI,
            M_PI - pi_viete ());
        if (stats)
            printf("viete() factors = %d\n", pi_viete_factors ())

    if (selected_options [5])
    printf("wallis() = %16.15lf, M_PI = %16.15lf, diff =
    %16.15lf\n", pi_wallis (), M_PI,
            M_PI - pi_wallis ())
        if (stats)
            printf("wallis() factors = %d\n", pi_wallis_factors ())

    if (selected_options [6])
        double i = 0
```

```
        while (i <= 9.9)
            printf("sqrt_newton(%.2lf) = %16.15lf, sqrt(%.2lf) = %16.15lf, diff =
            %16.15lf\n", i, sqrt_newton (i), i, sqrt (i), sqrt (i) - sqrt_newton (i))
            if (stats)
                printf("sqrt_newton() iters = %d\n", sqrt_newton_iters ())
            i = i + 0.1

    return 0

void help (void)
    printf("%s\n",
        "SYNOPSIS\n   A test harness for the small numerical library.\n
        USAGE\n   ./mathlib-test-x86 -[aebmrvnsh]\n\n
        OPTIONS\n
        -a   Runs all tests.\n
        -e   Runs e test.\n
        -b   Runs BBP pi test.\n
        -m   Runs Madhava pi test.\n
        -r   Runs Euler pi test.\n
        -v  Runs Viete pi test.\n
        -w   Runs Wallis pi test.\n
        -n   Runs Newton square root tests.\n
        -s   Print verbose statistics.\n
        -h   Display program synopsis and usage.
        ")

For this main function, I was originally going to create an array of function pointers
to each of the functions and then use the switch statement array that I had made to
easily navigate through the functions. However, the two main things that kept me from
doing this were the newton square root function, that took in an input, as well as the
statistics function, which called extra functions. I then reverted back into basic if
statements and came up with the current version.
```

Results

In the end, my code achieved everything that it was meant for. However, there are many things lacking in this code. First, my code is not that optimized, and I used techniques that did were not really right for the situation. I feel like I used way too many if statements in my code, and that would lead to slower processing.

Error Handling

There are multiple instances of errors that could happen with this code. As seen in figure 2, if the user does not input anything or inputs something that is not supposed to be there, they will be redirected to the help manual. As seen in figure 3, even though the -b option is called twice, it only runs once.

References

[1] Darrell Long, Kerry Veenstra, Jess Srinivas. Assignment 2 a little slice of pi. https://git.ucsc.edu/cse13s/spring2023/resources/-/blob/master/asgn2/asgn2.pdf, 2023. [Online; accessed 30-April-2023].

```
rahula2003@Ubuntu:~/cse13s/asgn2$ ./mathlib-test -a
e() = 2.718281828459043, M_E = 2.718281828459045, diff = 0.000000000000002
pi_bbp() = 3.141592653589793, M_PI = 3.141592653589793, diff = 0.000000000000000
pi_madhava() = 3.141592653589800, M_PI = 3.141592653589793, diff = -0.000000000000007
pi_euler() = 3.141592558095903, M_PI = 3.141592653589793, diff = 0.000000095493891
pi_viete() = 3.141592653589789, M_PI = 3.141592653589793, diff = 0.000000000000004
pi_wallis() = 3.141592495717063, M_PI = 3.141592653589793, diff = 0.000000157872730
sqrt_newton(0.00) = 0.000000000000007, sqrt(0.00) = 0.000000000000000, diff = -0.000000000000007
sqrt_newton(0.10) = 0.316227766016838, sqrt(0.10) = 0.316227766016838, diff = 0.000000000000000
sqrt_newton(0.20) = 0.447213595499958, sqrt(0.20) = 0.447213595499958, diff = -0.000000000000000
sqrt_newton(0.30) = 0.547722557505166, sqrt(0.30) = 0.547722557505166, diff = 0.000000000000000
sqrt_newton(0.40) = 0.632455532033676, sqrt(0.40) = 0.632455532033676, diff = 0.000000000000000
sqrt_newton(0.50) = 0.707106781186547, sqrt(0.50) = 0.707106781186548, diff = 0.000000000000000
sqrt_newton(0.60) = 0.774596669241483, sqrt(0.60) = 0.774596669241483, diff = 0.000000000000000
sqrt_newton(0.70) = 0.836660026534076, sqrt(0.70) = 0.836660026534076, diff = 0.000000000000000
sqrt_newton(0.80) = 0.894427190999916, sqrt(0.80) = 0.894427190999916, diff = 0.000000000000000
sqrt_newton(0.90) = 0.948683298050514, sqrt(0.90) = 0.948683298050514, diff = 0.000000000000000
sqrt_newton(1.00) = 1.000000000000000, sqrt(1.00) = 1.000000000000000, diff = -0.000000000000000
sqrt_newton(1.10) = 1.048808848170152, sqrt(1.10) = 1.048808848170151, diff = -0.000000000000000
sqrt_newton(1.20) = 1.095445115010332, sqrt(1.20) = 1.095445115010332, diff = 0.000000000000000
sqrt_newton(1.30) = 1.140175425099138, sqrt(1.30) = 1.140175425099138, diff = 0.000000000000000
sqrt_newton(1.40) = 1.183215956619923, sqrt(1.40) = 1.183215956619923, diff = -0.000000000000000
sqrt_newton(1.50) = 1.224744871391589, sqrt(1.50) = 1.224744871391589, diff = 0.000000000000000
sqrt_newton(1.60) = 1.264911064067352, sqrt(1.60) = 1.264911064067352, diff = -0.000000000000000
sqrt_newton(1.70) = 1.303840481040530, sqrt(1.70) = 1.303840481040530, diff = 0.000000000000000
sqrt_newton(1.80) = 1.341640786499874, sqrt(1.80) = 1.341640786499874, diff = 0.000000000000000
sqrt_newton(1.90) = 1.378404875209022, sqrt(1.90) = 1.378404875209022, diff = 0.000000000000000
sqrt_newton(2.00) = 1.414213562373095, sqrt(2.00) = 1.414213562373095, diff = -0.000000000000000
sqrt_newton(2.10) = 1.449137674618944, sqrt(2.10) = 1.449137674618944, diff = -0.000000000000000
sqrt_newton(2.20) = 1.483239697419133, sqrt(2.20) = 1.483239697419133, diff = 0.000000000000000
sqrt_newton(2.30) = 1.516575088810310, sqrt(2.30) = 1.516575088810310, diff = 0.000000000000000
sqrt_newton(2.40) = 1.549193338482967, sqrt(2.40) = 1.549193338482967, diff = 0.000000000000000
```

Figure 1: Screenshot of -a option running

```
rahula2003@Ubuntu:~/cse13s/asgn2$ ./mathlib-test idk bruh
SYNOPSIS
   A test harness for the small numerical library.
USAGE
   ./mathlib-test-x86 -[aebmrvnsh]

OPTIONS
  -a   Runs all tests.
  -e   Runs e test.
  -b   Runs BBP pi test.
  -m   Runs Madhava pi test.
  -r   Runs Euler pi test.
  -v   Runs Viete pi test.
  -w   Runs Wallis pi test.
  -n   Runs Newton square root tests.
  -s   Print verbose statistics.
  -h   Display program synopsis and usage.
```

Figure 2: Screenshot of help function running

```
rahula2003@Ubuntu:~/cse13s/asgn2$ ./mathlib-test -v -b -b -s
pi_bbp() = 3.141592653589793, M_PI = 3.141592653589793, diff = 0.000000000000000
pi_bbp() terms = 11
pi_viete() = 3.141592653589789, M_PI = 3.141592653589793, diff = 0.000000000000004
pi_viete() terms = 24
```

Figure 3: Screenshot of repeated options and statistics running

```
rahula2003@Ubuntu:~/cse13s/asgn2$ ./mathlib-test -v -b -b -s -a
e() = 2.718281828459043, M_E = 2.718281828459045, diff = 0.000000000000002
e() terms = 17
pi_bbp() = 3.141592653589793, M_PI = 3.141592653589793, diff = 0.000000000000000
pi_bbp() terms = 11
pi_madhava() = 3.141592653589800, M_PI = 3.141592653589793, diff = -0.000000000000007
pi_madhava() terms = 27
pi_euler() = 3.141592558095903, M_PI = 3.141592653589793, diff = 0.000000095493891
pi_euler() terms = 10000000
pi_viete() = 3.141592653589789, M_PI = 3.141592653589793, diff = 0.000000000000004
pi_viete() terms = 24
pi_wallis() = 3.141592495717063, M_PI = 3.141592653589793, diff = 0.000000157872730
pi_wallis() terms = 4974440
sqrt_newton(0.00) = 0.000000000000007, sqrt(0.00) = 0.000000000000000, diff = -0.000000000000007
sqrt_newton() terms = 47
sqrt_newton(0.10) = 0.316227766016838, sqrt(0.10) = 0.316227766016838, diff = 0.000000000000000
sqrt_newton() terms = 7
sqrt_newton(0.20) = 0.447213595499958, sqrt(0.20) = 0.447213595499958, diff = -0.000000000000000
sqrt_newton() terms = 7
sqrt_newton(0.30) = 0.547722557505166, sqrt(0.30) = 0.547722557505166, diff = 0.000000000000000
sqrt_newton() terms = 6
sqrt_newton(0.40) = 0.632455532033676, sqrt(0.40) = 0.632455532033676, diff = 0.000000000000000
sqrt_newton() terms = 6
sqrt_newton(0.50) = 0.707106781186547, sqrt(0.50) = 0.707106781186548, diff = 0.000000000000000
sqrt_newton() terms = 6
sqrt_newton(0.60) = 0.774596669241483, sqrt(0.60) = 0.774596669241483, diff = 0.000000000000000
sqrt_newton() terms = 5
sqrt_newton(0.70) = 0.836660026534076, sqrt(0.70) = 0.836660026534076, diff = 0.000000000000000
sqrt_newton() terms = 5
sqrt_newton(0.80) = 0.894427190999916, sqrt(0.80) = 0.894427190999916, diff = 0.000000000000000
```

Figure 4: Screenshot of more repeated options and statistics running