Assignment 2 – Slice of Pi

Aditya Bhaskar

CSE 13S – Spring 2023

Purpose

This section describes the implementation of several mathematical functions in C programming language. These functions are used to approximate the values of mathematical constants like e and π, and to compute the square root of a given number. The functions have been implemented using different algorithms, such as **Taylor series, Madhava series, Bailey-Borwein-Plouffe formula, and Newton-Raphson method.** Each function is implemented in a separate C file, and has two sub-functions: one to approximate the value of the constant or function, and another to track the number of terms or iterations taken to achieve the desired level of accuracy. This code can be used in various industries, such as finance, engineering, and scientific research, where accurate calculations of mathematical functions are required.

How to Use the Program

Welcome to our mathematical library program! Here's a guide on how to use it effectively.

- **Compiling the program:**
  To compile the program, navigate to the directory where the program files are located in a terminal window, and type "make". This will compile all the necessary files and create the executable "mathlib-test".
- **Running the program:** To run the program, type "./mathlib-test -h" in the terminal. You will then be presented with a menu of options to choose from. And using the command line options from below (check footnote) to check for approximations according to the preferred method.

For **option 1**[1], you can approximate the value of e using the *Taylor series method*, and the program will output the value of e along with the number of computed terms. **Option 2**[2] approximates π using the *Bailey-Borwein-Plouffe* formula, and outputs the approximation of π and the number of computed terms. **Option 3**[3] approximates π using the *Madhava series* and outputs the approximation of π and the number of computed terms. **Option 4**[4] approximates π using *Euler's* solution to the *Basel problem* and outputs the approximation of π and the number of computed terms. **Option 5**[5] approximates π using *Viete's* formula and outputs the approximation of π and the number of computed factors. **Option 6**[6] approximates π using *Wallis's* formula and outputs the approximation of π and the number of computed factors. **Option 7**[7] approximates π allows you to approximate the square root of a number using the *Newton-Raphson* method, and outputs the approximation and the number of iterations taken. **Option 8**[8] allows you to run all the tests.

[1] $-e$: Runs e approximation test.
[2] $-b$: Runs Bailey-Borwein-Plouffe π approximation test
[3] $-m$: Runs Madhava π approximation test.
[4] $-r$: Runs Euler sequence π approximation test
[5] $-v$: Runs Viete π approximation test.
[6] -w : Runs Wallis π approximation test.
[7] $-n$: Runs Newton-Raphson square root approximation tests, calling sqrtnewton() with various inputs for testing. This option does not require any parameters, and will only test within the range of $[0, 10)$ (it will not test the value 10).
[8] $-a$: Runs all tests

Option 9[9] Display a help message detailing program usage. **Option 10**[10] Enables printing of statistics to see computed terms and factors for all tested functions. To remove the files created during compilation, navigate to the folder in your terminal and run the command ”make clean”. This will remove all object files and executables created during compilation.

[9] $-h$: Display a help message detailing program usage.
[10] $-s$: Enable printing of statistics to see computed terms and factors for all tested functions.

Program Design

The program is organized into separate files for each mathematical formula used to approximate the value of π or the square root of a number. Each file contains two functions that calculate the approximation and track the number of computed terms or factors. The main algorithms used in the program are the *Taylor series* for e, the *Madhava* series, Euler's solution to the *Basel problem*, the *Bailey-Borwein-Plouffe* formula, *Viète's* formula, and *Wallis's* formula for π. For the square root approximation, the program uses the *Newton-Raphson* method. The program also includes a header file, **mathlib.h**, which contains the definition of a floating-point value EPSILON, used as the stopping criteria for each of the functions. It also includes the macro **absolute()**, used to compute the absolute value of a number and various function definitions that are used in other programs. The **mathlib-test.c** file is the entry point of the program and contains the main function, which takes command line. Four C files (**bbp.c, madhava.c, euler.c, viete.c, wallis.c**) contain the implementation of the algorithms used to estimate the value of π. One file (**e.c**) to estimate the value of e using Taylor Series. And lastly, **sqrt-newton()** which approximates the square root of a number. Each of these files contains a function that implements the algorithm and another function that returns the number of iterations performed. The program also includes a **makefile**, which is a file that specifies the dependencies and compilation commands needed to build the program. The **makefile** ensures that the necessary files are compiled in the correct order and that the program is built correctly. In summary, the program is organized into separate C files that contain functions to implement the different algorithms used to estimate π and e. The math functions used in these algorithms are defined in a separate **mathlib.h** file, and the **makefile** ensures that the program is built correctly.

Data Structures

We use **argv** which is an array of pointers to string that contains the parameters entered when the program was ran.

Algorithms

The following snippet contains the pseudocode of the **main()** of **mathlib-test.c**:

```
main():
    opt = 0;
    aflag = False, eflag = False, bflag = False, mflag = False, eflag = False
    vflag = False, wflag = False, nflag = False, sflag = False, hflag = False

    while (opt = getopt(argc, argv, COMMAND != -1):
        switch (opt):
            case 'a':
                aflag = True
                break
            case 'e':
                eflag = True
                break
            case 'b':
                bflag = True
                break
            case 'm':
                mflag = True
                break
            case 'r':
                rflag = True
                break
            case 'v':
                vflag = True
                break
            case ’w’:
                wflag = True
                break
            case ’n’:
                nflag = True
                break
            case ’s’:
                sflag = True
                break
            case ’h’:
                hflag = True
                break
            default:
                hflag = True
                break

     if (no commands are passed):
        hflag = 1

    if (sflag is True) and (other flags are false):
        hflag = 1

    if (hflag is True):
        printf("Usage: %s\n", argv[0])
        printf("Options:\n")
        printf("-a: Run all the tests\n")
        printf("-b: Calculate pi using the Bailey-Borwein-Plouffe method\n")
        printf("-m: Calculate pi using the Madhava method\n")
        printf("-r: Calculate pi using the Euler method\n")
        printf("-v: Calculate pi using the Viete method\n")
        printf("-w: Calculate pi using the Wallis method\n")
        printf("-n: Calculate the square root using the Newton method\n")
        printf("-s: Display the number of terms\n")
        printf("-e: Approximate e using Taylor Series\n")
        printf("-h: Display this help message\n")

        aflag = False, eflag = False, bflag = False, mflag = False, rflag = False
        vflag = False, wflag = False, nflag = False, sflag = False

    if (aflag is True):
        eflag = True, bflag = True, mflag = True, rflag = True, vflag = True
        wflag = True, nflag = True

    if (eflag is True):
        eval = e()
        eterms = e_terms()

        printf("e() = %, M_E = %, diff = %\n", eval, e, e - eval)
        if (sflag is True):
            printf("e() terms = %\n", eterms - 1)

    if (bflag is True):
        bval = pi_bbp()
        bterms = pi_bbp_terms()

        printf("pi_bbp() = %, M_PI = %, diff = %\n", bval, pi, pi - bval)
        if (sflag is True):
            printf("pi_bbp() terms = %\n", bterms - 1)

    if (mflag is True):
        mval = pi_madhava()
        mterms = pi_madhava_terms()

        printf("pi_madhava() = %, M_PI = %, diff = %\n", mval, pi, pi - mval);
        if (sflag is True):
            printf("pi_madhava() terms = %d\n", mterms)

    if (rflag is True):
        rval = pi_euler();
        rterms = pi_euler_terms();

        printf("pi_euler() = %16.15lf, M_PI = %16.15lf, diff = %16.15lf\n", rval, pi, pi - rval);
        if (sflag is True)
            printf("pi_euler() terms = %d\n", rterms - 1);

    if (vflag is True) {
        vval = pi_viete();
        vterms = pi_viete_factors();

        printf("pi_viete() = %16.15lf, M_PI = %16.15lf, diff = %16.15lf\n", vval, pi, pi - vval);
        if (sflag is True) {
            printf("pi_viete() terms = %d\n", vterms - 1);
        }
    }

    if (wflag is True) {
        wval = pi_wallis();
        wterms = pi_wallis_factors();

        printf("pi_wallis() = %16.15lf, M_PI = %16.15lf, diff = %16.15lf\n", wval, pi, pi - wval);
        if (sflag is True) {
            printf("pi_wallis() terms = %d\n", wterms - 1);
        }
    }

    if (nflag is True) {
        for (float i = 0.0; i < 9.95; i = i + 0.1) {
            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 (sflag is True) {
                printf("sqrt_newton() terms = %d\n", sqrt_newton_iters());
            }
        }
    }
```

Function Descriptions

e()

- No input
- Estimates the value of e using taylor series
- To precisely calculate e

eterms()

- No input
- Calculates the terms/iterations used to reach the value
- No. of terms/iterations

pibbp()

- No input
- Estimates the value of pi using bbp formula
- To precisely calculate pi

pibbpterms()

- No input
- Calculates the terms/iterations used to reach the value
- No. of terms/iterations

pimadhava()

- No input
- Estimates the value of pi using madhava series
- To precisely calculate pi

pimadhavaterms()

- No input
- Calculates the terms/iterations used to reach the value
- No. of terms/iterations

pieuler()

- No input
- Estimates the value of pi using basel problem
- To precisely calculate pi

pieulerterms()

- No input
- Calculates the terms/iterations used to reach the value
- No. of terms/iterations

piviete()

- No input
- Estimates the value of pi using viete formula (products)
- To precisely calculate pi

pivietefactors()

- No input
- Calculates the terms/iterations used to reach the value
- No. of terms/iterations

piwallis()

- No input
- Estimates the value of pi using wallis formula (products)
- To precisely calculate e

piwallisfactors()

- No input
- Calculates the terms/iterations used to reach the value
- No. of terms/iterations

sqrtnewton()

- any number
- Estimates the square root of the number using newton method
- To precisely calculate square root

sqrtnewtoniters()

- No input
- Calculates the terms/iterations used to reach the value
- No. of terms/iterations

main()

- command line arguments
- run any of the above mentioned functions using command line options

Results

It accurately estimates the value of pi and e using the different algorithms.

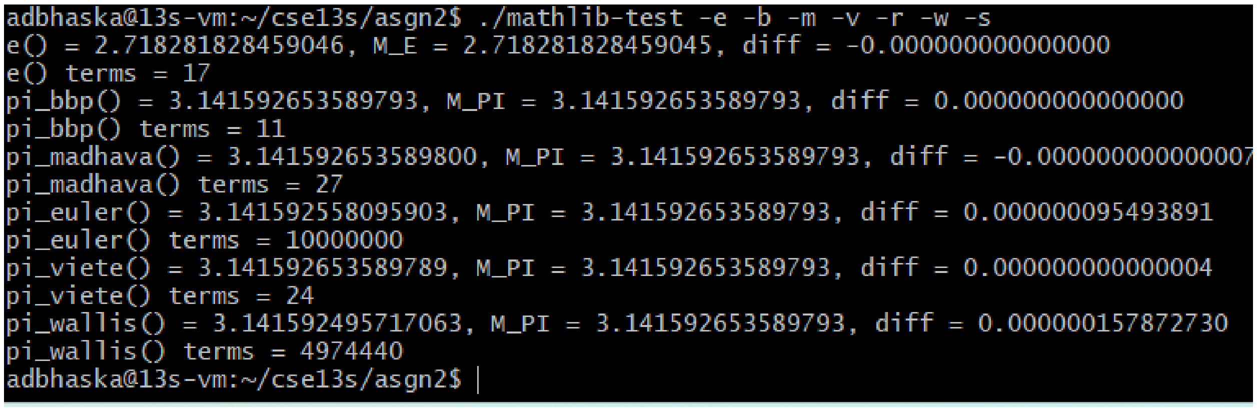
Numeric results

Fig. 1.

Error Handling

There are fairly small differences between some values.

References



```
adbhaska@13s-vm:~/cse13s/asgn2$ ./mathlib-test -e -b -m -v -r -w -s
e() = 2.718281828459046, M_E = 2.718281828459045, diff = -0.000000000000000
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
adbhaska@13s-vm:~/cse13s/asgn2$ |
```

Figure 1: Screenshot of the program running.