Московский государственный университет
имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики

Отчет по заданию №6

«Сборка многомодульных программ. Вычисление корней уравнений и определенных интегралов.»

Вариант 2 / 2 / 1

Выполнил:
студент 104 группы
Киперь А. Б.

Преподаватель:
Цыбров Е. Г. Кулагин А. В.

Москва
2016

Содержание

Постановка задачи

В данной лабораторной работе мне требовалось реализовать следующие возможности:

- реализация функций на языке ассемблер и последующее их экспортирование в код на Си
- раелизация сборки объектного файлв и его подключения к основному коду
- реализация функций для вычисления корней уравнений
- реализация функций для вычисления определенных интегралов
- реализация функцийб тестриущих функционал двух вышеуказанных
- поддержка тестового режима(флаг -t) и флага -help

Реализовывались следующие функции:

$$y = 3 * \left(\frac{0.5}{x+1} + 1\right)$$
$$y = 2.5 * x - 9.5$$
$$y = \frac{5}{x} \quad (x > 0)$$

Метод приблеженного вычисления корней уравнения для реализации - метод хорд(секущих).

Метод вычисления определенных интегралов - метод прямоугольников.

Математическое обоснование

Набор кривых представляет собой 2 гиперболы и 1 линейную функцию. Заметим, что одна из них определена только при x $>$ 0 $y = \frac{5}{x}$, это я буду учитывать при подсчете площади, определенной пересечением 3-ех функций. При подсчет же простых интегралов в режиме тест буду учитываться промежутки неопределенности функций, но они будут определены на всей оси Ox.

Реализация нахождения точек происходит автоматически, если интервал не был задан, либо был задан неточно. Начальными значениями берутся 0.5 и 1.0 с положительной строны(если правая часть интеграла лежала правее нуля) и с -1 - eps и -1 - 2 * eps дабы уловить момент, достаточно малый для точного нахождения корня.

Такие значения были подобраны для оптимального по скорости и точности нахождения корней. Так же программв проверяет корректность интервала и наличие на нем точек, в которых функции неопределны. Эпсилон для поиска корней был выбран равным 1e-3, а эпсилон для поиска интеграла был выбран равным 1e-5. Их можно менять в тестовом режиме.

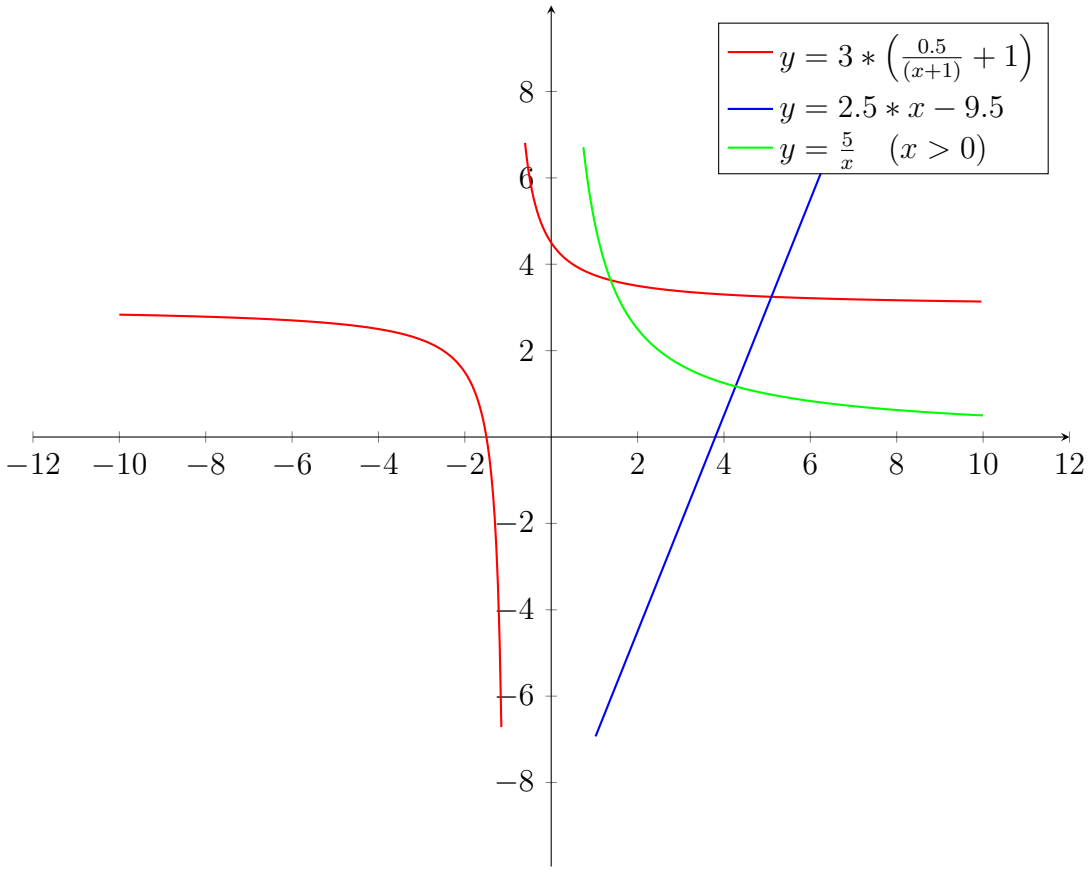

Рис. 1: Плоская фигура, ограниченная графиками заданных уравнений

Результаты экспериментов

Здесь представлена табличка с точками пересечения кривых и значениями в этих точках.

Кривые	x	y
1 и 2	5.09839	3.24597
2 и 3	4.26854	1.17136
1 и 3	1.37701	3.63104

Таблица 1: Координаты точек пересечения

Площадью фигуры, ограниченной графиками заданных уравнений, является площадь = 5.087758

Структура программы и спецификация функций

Здесь будут представны функции, которые были использованы в программе, за исключением функций переменных, так как их нет смысла описывать. Они писались по формуле, записанной на бумажке.

Начнем тогда с функций, которые были использованы для подсчета интеграла, кроме напианный на си.

```
// подынтегральные функции
double F1(double x) {
    return f1(x) - f3(x);
}

double F_1_5(double x) {
    return f2(x) - f3(x);
}

double F2(double x) {
    return f1(x) - f2(x);
}
```

Данные функции были использованы для вычисления площади искомой фигуры.

```
// функция для подсчета интеграла в тестовом режиме с 2-мя функциями
double F(double (*f)(double), double (*g)(double), double x) {
    if (x == 0 || x == -1) return NAN;

    return f(x) - g(x);
}
```

Эта функция является вспомогательной и нужна для подсчета интеграла в тестовом режиме между двумя функциями.

```
// подбираем достаточно маленький интервал на котором будет точка пересечения д
void find_interval_positive(double (*f)(double), double (*g)(double), double* a
    double x1 = 0.5, x2 = 1.0;

    while (F(f, g, x1) * F(f, g, x2) > 0) {
        x1 += 0.5;
        x2 += 0.5;
    }

    *a = x1;
    *b = x2;
}

void find_interval_negative(double (*f)(double), double (*g)(double), double* a
    double x1 = -1 - eps, x2 = -1 - 2 * eps;
```

```
    while (F(f, g, x1) * F(f, g, x2) > 0) {
        x1 -= eps;
        x2 -= eps;
    }

    *a = x1;
    *b = x2;
}
```

Эти функции необходимы для поиска интервала, на котором будет точка пересечения двух функций. Они используются для определения области, в которой будет происходить интегрирование.

```
int check_functions_on_interval(double (*f)(double), double (*g)(double), doubl
    if (F(f, g, *a) * F(f, g, *b) > 0) {
        printf("There is no intersection point on the interval [%f, %f]\n", *a,
        printf("Program will try to search it automaticly.\n");

        if (*b < 0) {
            find_interval_negative(f, g, a, b);
        }
        else {
            find_interval_positive(f, g, a, b);
        }

        return -1;
    }

    printf("Have an intercestion point on the interval [%f, %f]\n", *a, *b);

    return 1;
}
```

Эта функция проверяет, есть ли точка пересечения двух функций на заданном интервале. Если нет, она ищет интервал, на котором будет точка пересечения.

```
// функция для проверки интервалов
void check_interval(double a, double b) {
    if (a > b) {
        printf("Interval is wrong!!!! You need to enter numbers in proper order

        exit(-1);
    }

    if (a > -1 + eps && a <= 0.2 || a <= -1 && b >= -1) {
        printf("Given interval include points 0 and -1 and their small areas wh
```

```
        printf("Change input interval\n");

        exit(-1);
    }

    printf("Interval is valid! Continue counting with interval from %f to %f\n"
}
```

Эта функция проверяет, является ли заданный интервал правильным(в значении наличия точки пересечения на нем). Если нет, она запустит автоматический поиск интервала, основываясь на заданном. Если интервал был неправильным(перепутан порядок, либо был включен интервал неопределенности), то программа завершится досрочно с выдачей ошибки текстом.

```
// будем искать корень методом касательных(секущих)
double root(double f(double x), double g(double x), double* a, double* b, doubl
    check_interval(*a, *b);

    check_functions_on_interval(f, g, a, b);

    double xn_prev = *a, xn = *b;
    long max_iter = 1000, count_iterations = 0;

    while (count_iterations < max_iter) {
        double f_prev = F(f, g, xn_prev);
        double f_current = F(f, g, xn);

        if (isnan(f_prev) || isnan(f_current)) {
            printf("Error: x is out of scope!\n");
            return NAN;
        }

        if (fabs(f_current) < eps) break;

        if (fabs(f_current - f_prev) < 1e-10) {
            printf("Division by zero!\n");
            return NAN;
        }

        double xn_next = xn - f_current * (xn - xn_prev) / (f_current - f_prev)
        xn_prev = xn;
        xn = xn_next;
        count_iterations++;
    }

    if (count_iterations >= max_iter) {
        printf("It didn't add up in %ld iterations!\n", max_iter);
```

```
            return NAN;
        }

        printf("Found x = %.10f on the interval [%f, %f], iterations: %ld\n", xn, *
        printf("\n");

        return xn;
    }
```

Функция для поиска корней, думаю по названиям переменных и функций в полне понятно как она работает и что делает. Она сначала проверяет интервал, затем возможно, находит его сама и ищет на нем корень. Далее происходит итеративный метод нахождения корней с приближением. Если за 1000 итераций не найдется корня, то программа завершается с ошибкой.

```
    // метод левых прямоугольников
    double integral(double f(double x), double a, double b, double eps2) {
        check_interval(a, b);

        double s = 0.0;
        int i = 0;
        double step = eps2;

        while (a + step < b) {
            s += f(a + step);

            step += eps2;
        }

        return s * eps2;
    }
```

Эта функция считает интеграл по методу левых прямоугольников. Она проверяет интервал, затем считает. Так как в задании было сказано про метод прямоугольников, то я решил выбрать тот, который мы использовали на семинарах.

```
    double count_integral(double eps2) {
        double a, b;

        find_interval_positive(f1, f3, &a, &b);
        double first_intersection = root(f1, f3, &a, &b, eps);

        find_interval_positive(f2, f3, &a, &b);
        double second_intersection = root(f2, f3, &a, &b, eps);

        find_interval_positive(f1, f2, &a, &b);
        double third_intersection = root(f1, f2, &a, &b, eps);
```

```
    double first_area = integral(F1, first_intersection, second_intersection, e
    double second_area = integral(F2, second_intersection, third_intersection,

    return first_area + second_area;
}
```

Функция, для автоматизации вычисления искомого интеграла, где мы сначала находим точки(наши интервалы интегрирования), а после счиатем.

```
void test_root_func(int f_num1, int f_num2, double* a, double* b, double eps) {
    double intersection_point;

    if (f_num1 == 1 && f_num2 == 2 || f_num1 == 2 && f_num2 == 1) {
        intersection_point = root(f1, f2, a, b, eps);
    }
    else if (f_num1 == 2 && f_num2 == 3 || f_num1 == 3 && f_num2 == 2) {
        intersection_point = root(f2, f3, a, b, eps);
    }
    else if (f_num1 == 1 && f_num2 == 3 || f_num1 == 3 && f_num2 == 1) {
        intersection_point = root(f1, f3, a, b, eps);
    }

    printf("Intersection point on interval [%f, %f] is %f\n", *a, *b, intersect
}

void test_integral_func(int f_num1, int f_num2, double* a, double* b, double ep
    check_interval(*a, *b);

    double integral_result;

    if (f_num1 == 1 && f_num2 == 2 || f_num1 == 2 && f_num2 == 1) {
        integral_result = fabs(integral(F2, *a, *b, eps2));
    }
    else if (f_num1 == 2 && f_num2 == 3 || f_num1 == 3 && f_num2 == 2) {
        integral_result = fabs(integral(F_1_5, *a, *b, eps2));
    }
    else if (f_num1 == 1 && f_num2 == 3 || f_num1 == 3 && f_num2 == 1) {
        integral_result = fabs(integral(F1, *a, *b, eps2));
    }

    printf("Integral result on interval [%f, %f] is %f\n", *a, *b, integral_res
}
```

Функции для тестирования функций root и integral. Работают по принципу взаимодействия с пользователем. При неправильном вводе интервала будет выведено сообщение и предложение его исправить.

```
int main(int argc, char* argv[]) {
    double a, b;  // start and end of the interval

    int test_mode = 0;
    int need_help = 0;
    int f_num1 = 0, f_num2 = 0;
    int test_root = 0;
    int test_integral = 0;


    for (int i = 0; i < argc; i++) {
        if (strcmp(argv[i], "-help") == 0) {
            printf("You can use this folowing flags to work with program:\n");
            printf("-help -> give you info about flag and vars providing to thi
            printf("-t    -> test mode, take this parameters ((r/i), f_num1, f_

            exit(1);
        }

        if (strcmp(argv[i], "-t") == 0) {
            test_mode = 1;
            if (argc - i < 6) {
                printf("You need to provide at least 4 parameters: f1_num, f2_n
                printf("Also will be cool if there will be eps\n");
                exit(-1);
            }

            if (strcmp(argv[i + 1], "r") == 0) {
                test_root = 1;
            }
            if (strcmp(argv[i + 1], "i") == 0) {
                test_integral = 1;
            }

            f_num1 = atoi(argv[i + 2]);
            f_num2 = atoi(argv[i + 3]);

            if(f_num1 > 3 || f_num1 < 1 || f_num2 > 3 || f_num2 < 1) {
                printf("Numbers of functions are not exists. Please enter corre

                exit(-1);
            }

            a = atof(argv[i + 4]);
            b = atof(argv[i + 5]);
```

```
            if (test_integral) {
                eps2 = atof(argv[i + 6]);
            }

            if (test_root) {
                eps = atof(argv[i + 6]);
            }
        }
    }

    if (test_mode && test_root) {
        test_root_func(f_num1, f_num2, &a, &b, eps);
    }

    if (test_mode && test_integral) {
        test_integral_func(f_num1, f_num2, &a, &b, eps2);
    }


    if (argc == 1) {
        printf("Programm is running in standart mode.\n");
        printf("Will be print every intersection point, and interval will be se
        printf("\n");

        double integral_value = count_integral(eps2);

        printf("Value of integral is %f\n", integral_value);
    }

    return 0;
}
```

Функция запуска всей программы. ПРинимает на вход аргументы командной строки, которые определяют режим работы программы. Если режим работы программы обычный то считаеся площадь искомого интеграла. Если же он тестовый, то считаеся интеграл для тех двух функций, которые были заданы в аргументах командной строки. Так же видно что пользователь задает еще и эпсилоны.

Сборка программы (Make-файл)

В этом разделе требуется привести свой make файл. Мои зависимости слишком просты, чтобы для них потребовалась диаграмма. Так что опишу процесс здесь. Ассемблерные функции —> объектный файл с этими функциями —> main.c —> выполнение программы

```
all:
    @nasm -f elf32 functions.asm -o functions.o
    @gcc -m32 -no-pie main.c functions.o -o main
    @./main

test:
    @nasm -f elf32 functions.asm -o functions.o
    @gcc -m32 -no-pie main.c functions.o -o main
    @./main -t r 1 2 5 6 0.0001

clean:
    @rm main functions.o
```

Отладка программы, тестирование функций

Так как у программы два режима, то пользователь, заходя в режим теста, имеет право писать свои интервалы и эпсилон, но интервал всегда проверяется на наличи в нем интервала, на котором могут быть неопределны функции. Если такой обнаружен то интервал пользователя считается неправильным и программа сама выбирает интервал(ищет нужный, либо завершает работу).

Пример некоторых тестов, которые были проведены: ./main -t r 1 3 -2.5 -2 0.001 x = -1.210349 ./main -t r 1 3 -7 -6 0.0000001 x = -1.210348 (значение стало точнее) ./main -t i 1 3 7 10 0.001 S = 7.694387 ./main -t i 1 3 -6 -4 0.001 S = 7.261195 ./main -t r 2 3 2 7 0.001 x = 4.268572 ./main -t r 1 2 1 2 0.001 x = 5.098485 ./main -t i 2 3 3 10 0.001 x = 41.224469

Функции данные мне имеют следующие первообразные:

$$\int 3\left(\frac{0.5}{x+1}+1\right)\,dx = 1.5\ln(x+1)+3x \tag{1}$$

$$\int 2.5x - 9.5\,dx = 1.25x^2 - 9.5x \tag{2}$$

$$\int \frac{5}{x}\,dx = 5\ln(x) \tag{3}$$

Подставляя точки в полученные выражения получим искомые ответы

Программа на Си и на Ассемблере

Тексты программы на Си и на Ассемблере представлены в архиве, которые был направлен в телеграм преподавателю.

Анализ допущенных ошибок

Из ошибок мог бы выделить следующее:

- Пусть метод левых прямоугольников и считает вполне домтойно, но все же метод равномерных прямоугольников был бы лучше(пусть и не сильно отличался бы от нашего ответа, так как все равно учитывалась погрешность)
- Скорее всего тестовые функции можно было бы сделать более сложными
- Объектный файл не хотел долго настраиваться(классика).
- Так как у функций есть точки неопределености, то из-за метода хорд требуется исключить не только их но и их окрестности(пусть и не сильно маленькие).