Лабораторная работа 13

Цель работы

Изучение математических функций стандартной библиотеки math.h и стандартной библиотеки stdlib.h

Задание

1. Разработать программу и подпрограмму (подпрограммы), вычисляющую корень уравнения с точностью **ε = FLT_EPSILON*2** (значение константы FLT_EPSILON определено в файле float.h).

   Метод приближения, функция и начальные координаты отрезка, содержащего корень, приведены в таблице.

№	Метод приближения	Функция	Границы отрезка
1	деление отрезка пополам	$x+\ln(x+0.5)-0.5$	$0 \div 2$
2	метод хорд	$(2*\sin^2(x))/3-0.75*\cos^2(x)$	$0 \div \pi/2$
3	деление отрезка пополам	$x^2 * 2^x - 1$	$0 \div 1$
4	метод хорд	$x^2-\sin(5x)$	$0.5 \div 0.6$
5	деление отрезка пополам	$(4+x^2)(e^x-e^{-x})-18$	$1.2 \div 1.3$
6	метод хорд	$x^2-1.3*\ln(x+0.5)-2.8x+2.35$	$1.7 \div 2.7$

2. Разработать программу и подпрограмму (подпрограммы), вычисляющую значение интеграла с точностью ε = 0.0005.

   Интегрируемая функция и пределы интегрирования приведены в таблице, начальное число отрезков разбиения **n** = 10.

№	Функция	Границы отрезка
1	$e^{-x}*\cos(\pi x/4)$	$0 \div 2$
2	$\sqrt{tg(x)}$	$0 \div \pi/6$
3	e^x/x	$1 \div 7$
4	$\cos(x)/x$	$\pi/2 \div \pi$
5	$\sqrt{(1+\cos^2 x)}$	$0 \div \pi$
6	$e^x*\sin^2 x$	$0 \div 5$

Теоретическая часть

Функции, предназначенные для преобразования чисел, запроса памяти и других задач объединены в стандартной библиотеке stdlib.h.

Прототип	Описание
double atof(const char *s)	переводит *s* в *double*; эквивалентна strtod(s, (char**) NULL).
int atoi(const char *s)	переводит *s* в *int*; эквивалентна (int)strtol(s, (char**)NULL, 10).
int atol(const char *s)	переводит *s* в *long*; эквивалентна strtol(s, (char**) NULL, 10).
double strtod(const char *s, char **endp)	преобразует первые символы строки *s* в *double*, игнорируя начальные символы-разделители; запоминает указатель на непреобразованный конец в **endp* (если *endp* не NULL), при переполнении она выдает *HUGE_VAL* с соответствующим знаком, в случае, если результат оказывается меньше, чем возможно представить данным типом, возвращается 0; в обоих случаях в *errno* устанавливается *ERANGE*.
long strtol(const char *s, char **endp, int base)	преобразует первые символы строки *s* в *long*, игнорируя начальные символы-разделители; запоминает указатель на непреобразованный конец в **endp* (если *endp* не NULL). Если *base* находится в диапазоне от 2 до 36, то преобразование делается в

Прототип	Описание
	предположении, что на входе - запись числа по основанию *base*. Если *base* равно нулю, то основанием числа считается 8, 10 или 16; число, начинающееся с цифры 0, считается восьмеричным, а с 0x или 0X - шестнадцатеричным. Цифры от 10 до *base-1* записываются начальными буквами латинского алфавита в любом регистре. При основании, равном 16, в начале числа разрешается помещать 0x или 0X. В случае переполнения функция возвращает LONG_MAX или LONG_MIN (в зависимости от знака), а в *errno* устанавливается ERANGE.
unsigned long strtoul(const char *s, char **endp, int base)	работает так же, как и *strtol*, с той лишь разницей, что возвращает результат типа *unsigned long*, а в случае переполнения - ULONG_MAX.
int rand(void)	выдает псевдослучайное число в диапазоне от 0 до RAND_MAX; RAND_MAX не меньше 32767.
void srand(unsigned int seed)	использует *seed* в качестве начального значения для новой последовательности псевдослучайных чисел. Изначально параметр *seed* равен 1.
void *calloc(size_t nobj, size_t sze)	возвращает указатель на место в памяти, отведенное для массива *nobj* объектов, каждый из которых размера *size*, или, если памяти запрашиваемого объема нет, NULL. Выделенная область памяти обнуляется.
void *malloc(size_t size)	возвращает указатель на место в памяти для объекта размера *size* или, если памяти запрашиваемого объема нет, NULL. Выделенная область памяти не инициализируется.
void *realloc(void *p, size_t size)	перевыделяет память заменяет для объекта, на который указывает *p*. Для части, размер которой равен наименьшему из старого и нового (size) размеров, содержимое не изменяется. Если новый размер больше старого, дополнительное пространство не инициализируется, *realloc* возвращает указатель на новое место памяти или, если требования не могут быть удовлетворены, NULL (**p* при этом не изменяется).
void free(void *p)	освобождает область памяти, на которую указывает *p*; эта функция ничего не делает, если *p* равно NULL. В *p* должен стоять указатель на область памяти, ранее выделенную одной из функций: *calloc*, *malloc* или *realloc*.
void abort(void *p)	вызывает аварийное завершение программы, ее действия эквивалентны вызову **raise**(SIGABRT).
void exit(int status)	вызывает нормальное завершение программы. Функции, зарегистрированные с помощью **atexit**, выполняются в порядке, обратном их регистрации. Производится опорожнение буферов открытых файлов, открытые потоки закрываются, и управление возвращается в среду, из которой был произведен запуск программы. Значение *status*, передаваемое в среду, зависит от реализации, однако при успешном завершении программы принято передавать нуль. Можно также использовать значения EXIT_SUCCESS (в случае успешного завершения) и EXIT_FAILURE (в случае ошибки).
int atexit(void (*fcn)(void))	регистрирует *fcn* в качестве функции, которая будет вызываться при нормальном завершении программы; возвращает ненулевое значение, если регистрация не может быть выполнена.
int system(const char *s)	передает строку *s* операционной среде для выполнения. Если *s* – NULL и существует командный процессор, то *system* возвращает ненулевое значение. Если *s* не NULL, то возвращаемое значение зависит от реализации.

Прототип	Описание
char *getenv(const char *name)	возвращает строку среды, связанную с *name*, или, если никакой строки не существует, NULL. Детали зависят от реализации.
void *bsearch(const void *key, const void *base, size_t n, size_t size, int (*cmp)(const void *keyval, const void *datum))	ищет среди *base[0]...base[n-1]* элемент с подходящим ключом **key*. Функция *cmp* должна сравнивать первый аргумент (ключ поиска) со своим вторым аргументом (значением ключа в таблице) и в зависимости от результата сравнения выдавать отрицательное число, нуль или положительное значение. Элементы массива *base* должны быть упорядочены в возрастающем порядке, *bsearch* возвращает указатель на элемент с подходящим ключом или, если такого не оказалось, NULL.
void qsort(void *base, size_t n, size_t size, int (*cmp)(const void *, const void *))	сортирует массив *base[0]...base[n-1]* объектов размера *size* в возрастающем порядке. Функция сравнения *cmp* - такая же, как и в *bsearch*.
int abs(int n)	возвращает абсолютное значение аргумента типа *int*.
long labs(long n)	возвращает абсолютное значение аргумента типа *long*.
div_t div(int num, int denom)	вычисляет частное и остаток от деления *num* на *denom*. Результаты типа *int* запоминаются в элементах *quot* и *rem* структуры *div_t*.
ldiv_t ldiv(long num, long denom)	вычисляет частное и остаток от деления *num* на *denom*. Результаты типа *long* запоминаются в элементах *quot* и *rem* структуры *ldiv_t*.

Рекомендации по выполнению лабораторной работы

Определение корней функции

Численные методы позволяют найти решения определенных задач, заранее зная, что полученные результаты будут вычислены с определенной погрешностью, поэтому для многих численных методов необходимо заранее знать "уровень точности", которому будет соответствовать полученное решение. В связи с этим задача нахождения корней многочлена вида

$$F(x)= a_n x^n+...+a_2 x^2+a_1 x+a_0$$

представляет особый интерес, т.к. формулы нахождения корней даже кубического уравнения достаточно сложны, а если необходимо отыскать корни многочлена большей степени, то без помощи численных методов не обойтись. Рассмотрим два наиболее известных (и простых) методов нахождения корней: метод деления отрезка пополам и метод хорд.

Метод деления отрезка пополам

Метод деления отрезка пополам (известный еще и как метод половинного деления) также является рекурсивным, т.е. предусматривает повторение с учетом полученных результатов.

Исходные данные к методу деления отрезка пополам:

- дана функция f(x);
- определена допустимая погрешность Q;
- определен некоторый интервал [a, b], содержащий корень уравнения.

Алгоритм нахождения корня:

1. Если длина полученного отрезка меньше Q, то за значение корня принимается левая (или правая) граница отрезка. В противном случае
2. Вычисляем значение координаты x, беря середину отрезка [a,b], т.е. x= (a+b)/2
3. Вычисляем значения f(a), f(b), f(x), и осуществляем следующую проверку:

   Если |f(x)| <= Q, то корень с указанной точностью найден.

   Если |f(x)| > Q, т.е. необходимая точность еще не достигнута, то формируем два интервала: [a,x] и [x,b] проверяем знаки f(a), f(b), f(x). На концах одного из этих интервалов знаки функции будут одинаковы, а на другом – различны. Именно тот интервал, на концах которого знаки различны, мы берем за основу при следующей итерации, т.е. приравниваем либо a, либо b к x.
4. Переходим к пункту 1.

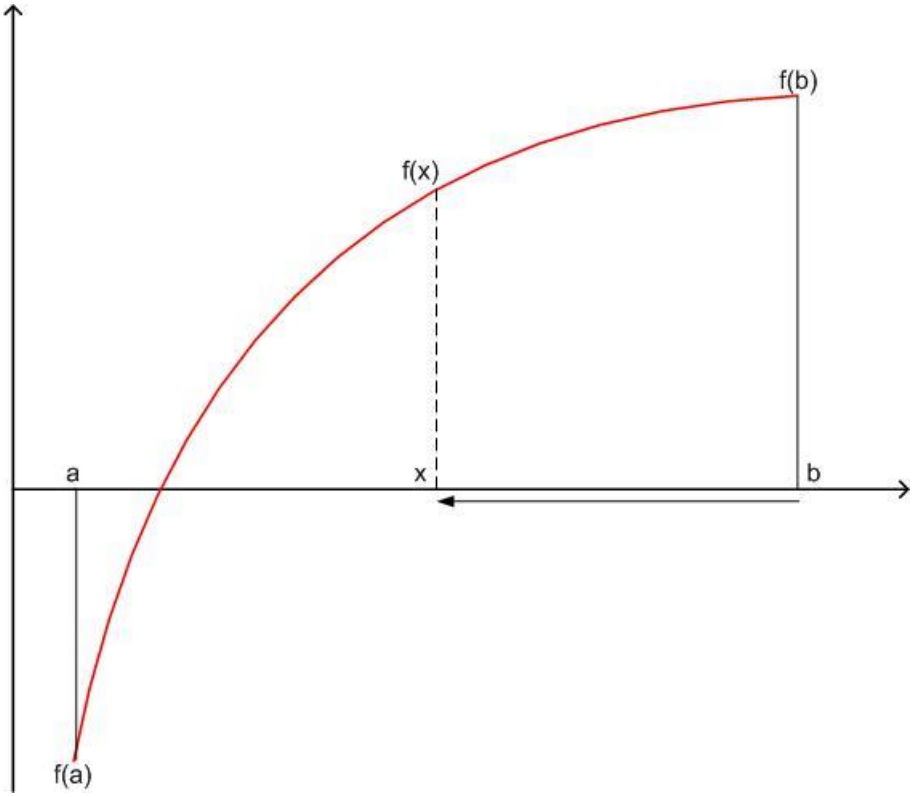

Рис. 9. Метод деления отрезка пополам

Метод хорд

Исходные данные к методу деления отрезка пополам:

- дана функция F(x);
- определена допустимая погрешность Q;
- определен некоторый интервал [a, b], содержащий корень уравнения.

Алгоритм нахождения корня:

1. Если длина полученного отрезка меньше Q, то за значение корня принимается левая (или правая) граница отрезка. В противном случае
2. Строим хорду, проходящую через точки f(a) и f(b) и вычисляем точку пересечения (x) хорды с осью абсцисс.

   Формула прямой, проходящей через точки а и в, т.е. хорды:

   $y=k*x+y_0$, где

   $$k = \frac{f(b)-f(a)}{b-a}$$

   $$y_0 = \frac{b*f(a)-a*f(b)}{b-a}$$

   тогда точка x пересечения хорды с осью абсцисс будет вычисляться по формуле:

   $$x = \frac{a*f(b)-b*f(a)}{f(b)-f(a)}$$

3. Вычисляем значения f(a), f(b), f(x), и осуществляем следующую проверку:

   Если |f(x)| <= Q, то корень с указанной точностью найден.

   Если |f(x)| > Q, т.е. необходимая точность еще не достигнута, то формируем два интервала: [a,x] и [x,b] проверяем знаки F(a), F(b), F(x). На концах одного из этих интервалов знаки функции будут одинаковы, а на другом – различны. Именно то интервал, на концах которого знаки различны, мы берем за основу при следующей итерации, т.е. приравниваем либо a, либо b к x.

4. Переходим к пункту 1.

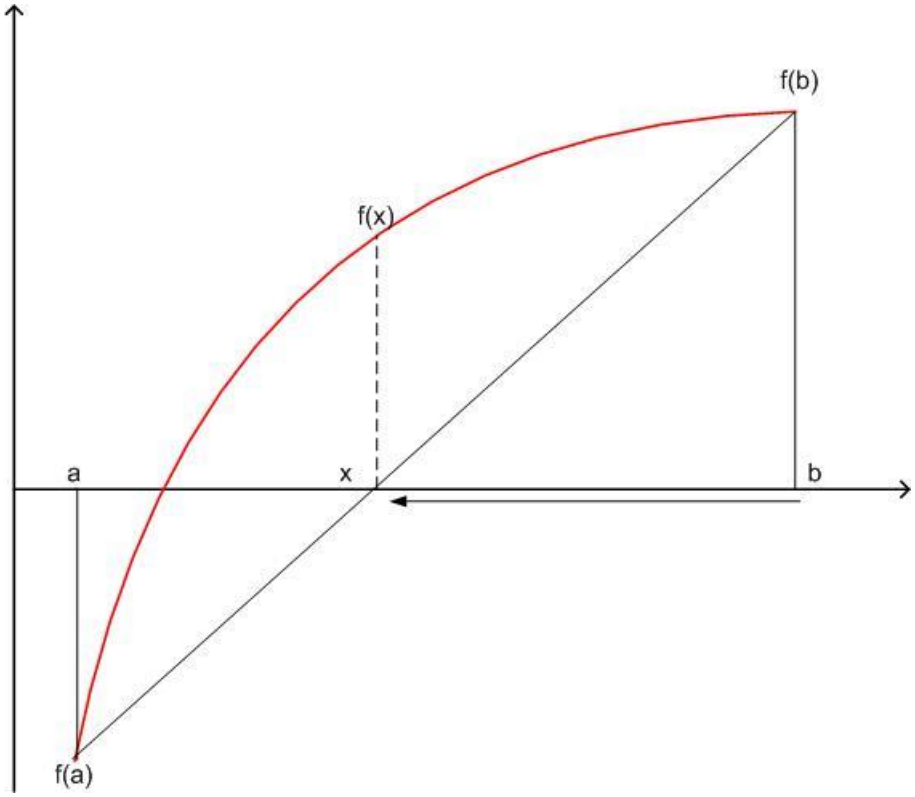

Рис. 10. Метод хорд

Вычисление интеграла

Известно, что определенный интеграл функции f(x) типа на интервале [a,b] численно представляет собой площадь криволинейной трапеции (Рис. 11), ограниченной прямыми x=0, y=a, y=b и функцией y=f(x). Есть два метода вычисления этой площади или определенного интеграла — метод трапеций (Рис. 12) и метод средних прямоугольников (Рис. 13).

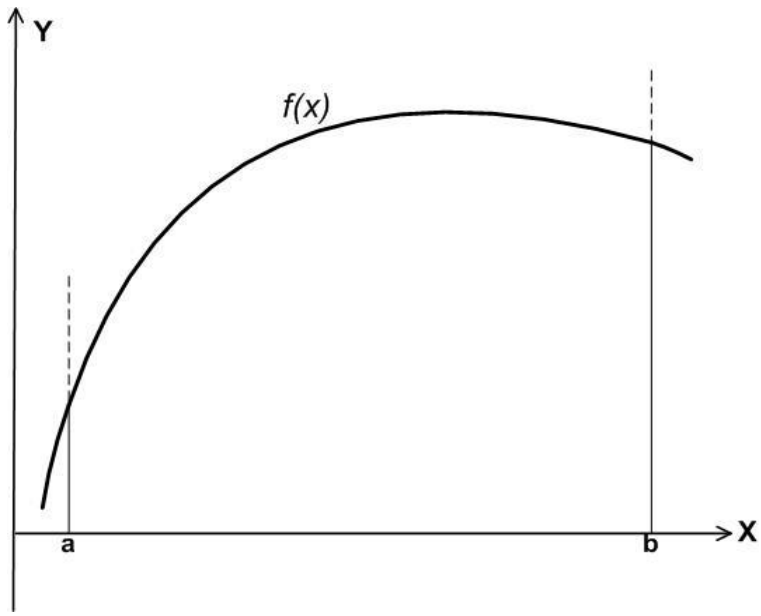

Рис. 11. Криволинейная трапеция

По методам трапеций и средних прямоугольников интеграл соответственно равен сумме площадей прямоугольных трапеций или сумме площадей прямоугольников, в которых высота определяется по точке пересечения верхнего основания прямоугольника, которое график функции должен пересекать в середине. Соответственно получаем формулы площадей

для метода трапеций:

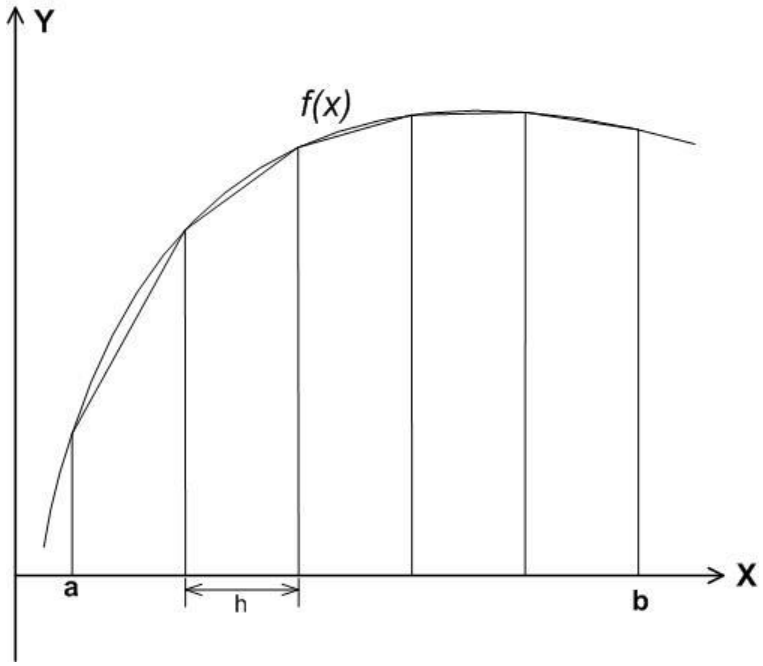

Рис. 12. Метод трапеций

$S = \sum_{i=0}^{n-1} f(a + h * i) * h + \frac{f(a)+f(b)}{2} * h$, где h – длина отрезка, n – число отрезков

и для метода средних прямоугольников:

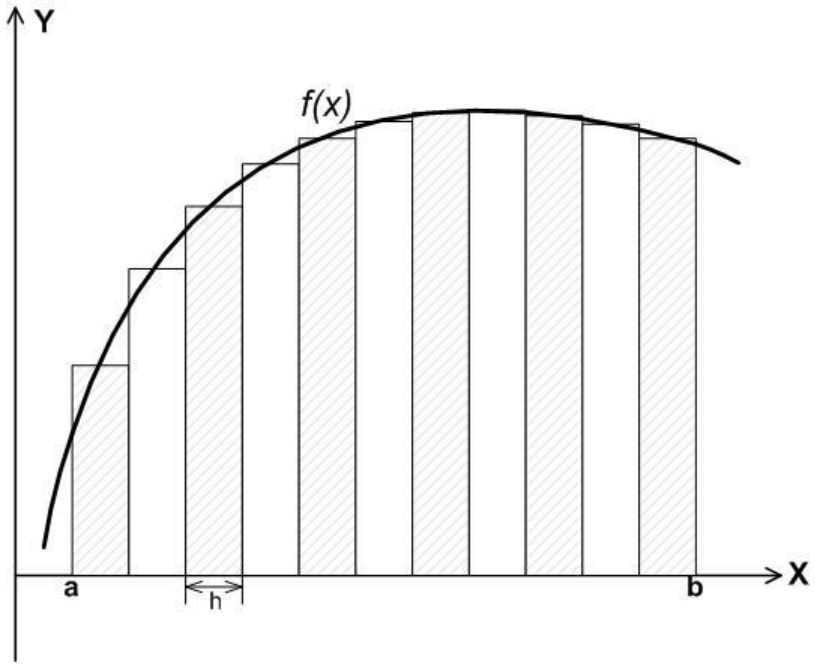

Рис. 13. Метод средних прямоугольников

$S = h * \sum_{i=0}^{n-1} f\left(a + h * i + \frac{h}{2}\right)$, где h – длина отрезка, n – число отрезков

Алгоритм вычисления интеграла по любому из методов будет выглядеть следующим образом:

1. При заданном числе отрезков разбиения вычисляется (по заданному методу) первоначальное приближение S_0
2. Число отрезков разбиения увеличивается на k (например, k=5) и вычисляется новое значение интеграла S_1
3. Если $|S_0-S_1|$ <= заданной точности, то за точное значение интеграла принимается значение S_1

В противном случае за первоначальное приближение S_0 принимается значение S_1 и алгоритм повторяется, начиная с п.2

Содержание отчета

Отчет по лабораторной работе должен содержать:

- задание лабораторной работы, соответствующее варианту
- структурную схему алгоритма программы и подпрограммы (подпрограмм)
- текст программы
- результаты работы программы