# Работа с действительными числами

*Коллоквиум №2*

Кострицкий А. С.

TS1905162322

## Задачи

1) Реализовать функцию для нахождения корней уравнения:

$$a,\, b,\, c \in \mathbb{R} \quad ax^2 + bx + c = 0,$$

с клавиатуры вводятся $a$, $b$, $c$, можно запрашивать у пользователя дополнительную информацию.

2) Реализовать функцию для нахождения корня уравнения:

$$a,\, b \in \mathbb{R},\, f \in C_0[a,\, b],\, f(a)f(b) \leqslant 0 \quad f(x) = 0,$$

с помощью метода бисекции. С клавиатуры вводятся $a$, $b$, можно запрашивать у пользователя дополнительную информацию.

3) Реализовать функцию для вычисления суммы ряда:

$$\sum_{i=1}^{i=j:|a_j|\leqslant\varepsilon} a_i, \quad \forall i \in \mathbb{Z} \quad a_i = (-1)^i \frac{x^{2i+1}}{(2i+1)!}.$$

# Ошибки

1) **Не умеем читать задание.**

   Возможно, эта вечная проблема проблемой не является — с точки зрения эволюции человек пришёл к поведению переобученной нейросети, которая стремится всё новое свести к старому и давно известному; было бы странно ожидать от Вас обратного. В первой задаче коллоквиума **не сказано**, что уравнение квадратное, что у него ровно два корня. В третьей задаче нигде ничего **не написано** про разложения синуса в ряд Маклорена.

2) **Сравнение действительных чисел.**

   Нельзя сравнивать два действительных числа с помощью обычного сравнения. Особенно обидно видеть эту ошибку, ведь на ней мы акцентируем внимание каждый раз, когда видим два действительных числа на лекции. Два числа с плавающей точкой нужно сравнивать с точностью[1] $\varepsilon$ :

   $$a = b \Leftrightarrow \texttt{fabs(a - b)} < \texttt{EPS}.$$

   Причём константа `EPS` обычно выбирается достаточно малой, но не меньше *машинного эпсилон*, о котором Вы должны знать из курса теории информатики[2].

3) **Эпсилон сравнения слишком мала.**

   Эпсилон сравнения не может не попадать в разрядную сетку.

4) **Отрицание действительного числа.**

   Выражение $!a$ для действительного числа выглядит странным в силу сказанного выше, во-первых. Во-вторых, шалости с неявным приведением типов обычно до добра не доводят, сторонний человек может подумать, что $a$ объявлено целым.

5) **Попытка вернуть данные разных типов.**

   Язык C — язык с нестрогой статической типизацией — тип сущности определяется в точке объявления и не может быть изменён, но тип выражения может быть приведён неявно в некоторых случаях (Причём в некоторых - с потерей информации). Рассмотрим функцию

---

[1] Во многих материалах Вы можете найти другой вариант этой оценки, в котором константу эпсилон умножают на максимальное из двух чисел.

$$a = b \Leftrightarrow \texttt{fabs(a - b)} < \texttt{EPS} * \texttt{Max(fabs(a), fabs(b))}).$$

Да, такая оценка является более «правильной», но обычно, если есть возможность, поставленную задачу *нормализуют*, то есть приводят все уравнения к такому виду, чтобы входные и выходные данные были «недалеко от единицы» — разрядная сетка чисел с плавающей точкой тем более плотна, чем ближе она к нулю (Попробуйте понять, почему, если Вам это не читали на ТИ).

[2] Если Вы считаете, что мы врём Вам в этом вопросе, можете попытаться найти информацию о функции/методе `is_equal` во многих языках объектной парадигмы.

```
double FindRoot(double a, /* other arguments*/)
{
 /* ... */
 if (f(a)f(b)>0) return EXIT_FAILURE;
 /* some code */
  return c;
}
```

С точки зрения автора, он возвращает приближение корня по методу бисекции, если ошибок нет, и возвращает код ошибки, если ошибка есть. С точки зрения транслятора, **что тут самое примечательное и опасное**, возвращаться явно из функции должно действительное число. Поэтому целое[3] число, которое будет подставлено на место макроопределения препроцессором в ходе препроцессинга, будет приведено к действительному. *Нельзя из функции возвращать код ошибки в одних случаях и не код ошибки в других.*

6) **Функция, не удовлетворяющая условие.**

Во второй задаче сказано, что целевая функция является непрерывной и имеет нуль на рассматриваемом участке. Но с чего Вы взяли, что входные данные валидны? Уже почти год, как в каждой лабораторной мы проверяем валидность входных данных и выходим с кодом ошибки в некоторых случаях. Точно так же и тут следует проверить ещё перед нахождением приближения корня условие $f(a)f(b) < 0$.

7) **Ошибки округления, выводящие цикл на бесконечность.**

Все арифметические операции с действительными числами выполняются с некоторой погрешностью. Самой опасной операцией является вычитание близких чисел, относительная погрешность которой

$$\delta(a + da - b - db) = \frac{a + da - b - db - (a - b)}{|a - b|} = \frac{da + db}{|a - b|}.$$

Помимо этого, разный порядок выполнения арифметических операций может приводить к разным результатам[4]. Исходя из этих двух фактов, никогда нельзя быть уверенным, что Ваш алгоритм сработает, поэтому ***никогда нельзя оставлять при работе с действительными числами циклы без подсчёта итераций.***

В основном цикле метода бисекции вместо условного цикла `while(fabs(a-b)>delta)` следовало бы писать `while((fabs(a-b)>delta)&& fabs(f(c))>delta && (iter < max_iter_count))` и считать итерации. Максимальное число итераций можно ограничить чем-то большим, чтобы просто пресечь выход на бесконечный цикл, хотя на практике «обычно» «считается», что если алгоритм не сошёлся за двадцать итераций, он не сойдётся никогда.

8) **Операции ввода-вывода внутри вычислительной функции.**

[3] И тут было бы крайне неплохо вспомнить, почему кодом ошибки или адресом не может быть действительное число.

[4] Типичным примером смены знака при вычислении является вычисление значения многочлена по схеме Горнера.

Внимательный студент заметит, что с прошлого коллоквиума этот пункт перекочевал из замечаний в ошибки. Да, с действительными числами упоминаемая проблема становится намного серьёзнее.

Дело в том, что внутри многих процессоров и математических сопроцессоров используется арифметика точности слегка лучше, чем просто double на восьми байтах[5]. То есть, выполнение последовательности операций над числами, пока они лежат на регистрах, может приводить к меньшей погрешности, чем выполнение той же последовательности операций с необходимостью выгружать посчитанный промежуточный результат в память[6]. А такие жирные с точки зрения машины операции, как ввод-вывод, безусловно, вынудят машину освобождать регистры под них.

9) **Использование явных формул вместо рекуррентных при вычислении сумм рядов.**

## Замечания по поводу стиля программирования

В программировании, как и в любом другом ремесле, теория рождалась из практики. Любая попытка описать практический опыт и создать свод догматов либо приводит к созданию новой парадигмы[7], либо к созданию нового языка[8], либо к созданию новой спецификации[9].

Обычно эти догматы представляют собой некоторые запреты на использование языковых конструкций: например, запрет использования оператора `goto`, единственность точки выхода, запрет использования слов `break, continue`, и так далее; или, наоборот, какие-либо общие советы по построению программы: гарантированное копирование сущности при изменении, передача всех аргументов в качестве констант, и так далее.

И, конечно, *из каждого правила можно найти исключение*,

***если уметь аргументировать свою позицию.***

То есть, все описанные ниже ситуации возможны, совершенное студентом в каждом примере действие не являлось бы ошибочным, если бы студент сумел показать невозможность иного подхода. Сразу стоит сказать, что следующие фразы аргументами не являются:

1) Меня так учили.

2) Меня так научили.

3) Я слышал.

4) Я видел.

5) Я пробовал.

[5]Это может быть, например, работа с регистрами в 80 бит.

[6]Вы не обязаны нам верить. Попробуйте сложить три числа и вывести результат, а потом сложить два числа, вывести на экран произвольное сообщение, прибавить третье число и вывести результат.

[7]Как, например, парадигма *структурного программирования*.

[8]Если хочется примеров, можно почитать истории создания языков Java, Haskell, Pascal, Ada.

[9]Причём тут полезно будет почитать про SEI CERT C Coding Standard. Я хотел бы назвать ещё один стандарт C, в котором запрещены все целые, кроме целых с явным указанием размера, но, к сожалению, вспомнить не могу.

6) Я читал.

7) Оно же работает!

8) Оно же транслируется!

9) В другой ситуации...

10) На Хабре...

11) В C++...

12) В задании не написано!

13) Мне так удобнее.

14) Я проверил: *на этой машине* всё работает.

15) Я проверял: *на всех моих машинах* всё работает.

16) Всегда так делали!

Ну а теперь сами примеры:

1) **Магические числа.**

2) **Не знаем о существовании инженерной или экспоненциальной записей числа.**

   Равны ли числа 0.0000000001 и 0.00000000001? А числа 0.00000000001 и 0.00000000001? Порой кажется, что студенты получают некое извращённое удовольствие от написания всех нулей в числе заместо написания того же числа в инженерной форме $1e-8$, $2.8e-12$, $1e5$, etc.