Отчет о практической работе по программированию

ученика 10 «М» класса

Московской государственной

Пятьдесят седьмой школы

Урманова Максима Тимуровича

Тема работы:

Символьное дифференцирование

Руководитель практики: Суханов Александр Алмазович

24 июня 2016 г.

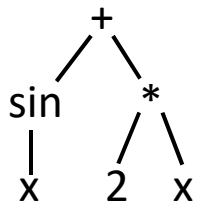
Постановка задачи

На вход программе подается функция, записанная в виде аналитического выражения (строки символов). Для ввода разрешается использование целых чисел, простых дробей (запись вида «a/b», где a, b – целые числа), переменной x, знаков +, -, *, /, ^ (возведение в степень), круглых скобок: ( ), тригонометрических функций sin, cos, tan, cat, а также функций loga(…) (взятие логарифма по основанию a, где a – натуральное число), ln(…) – взятие натурального логарифма.

Программа должна вывести на экран результат вычисления производной от заданной функции, либо сообщение об ошибке, если входная функция введена некорректно.

Метод решения

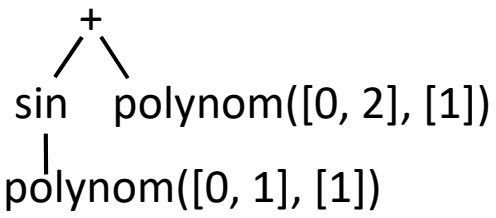
Решение задачи начинается с синтаксического анализа введенного математического выражения (parsing) [1]. Анализ заключается в выделении арифметических операций и функций внутри математического выражения. Каждая арифметическая операция имеет 2 операнда: например, в выражении «3 + 2» операндами арифметической операции «+» являются числа 3 и 2. Каждая функция имеет один или несколько аргументов, а также имя. Например, функция sin(x + 1) имеет аргумент (x + 1) и имя «sin», а функция loga(3x – 5) имеет имя log и аргументы a и (3x – 5). В ходе синтаксического анализа строится дерево, узлами которого являются арифметические операции и функции, при этом аргументами для каждой операции (функции) являются узлы – потомки данного узла (каждый узел содержит ссылки на своих потомков). Например, результатом анализа строки «sin(x) + 2*x» является дерево:



Также в ходе анализа осуществляется проверка корректности входных данных. Если в ходе анализа определяется, что выражение введено некорректно, программа выводит на экран соответствующее сообщение и завершает работу.

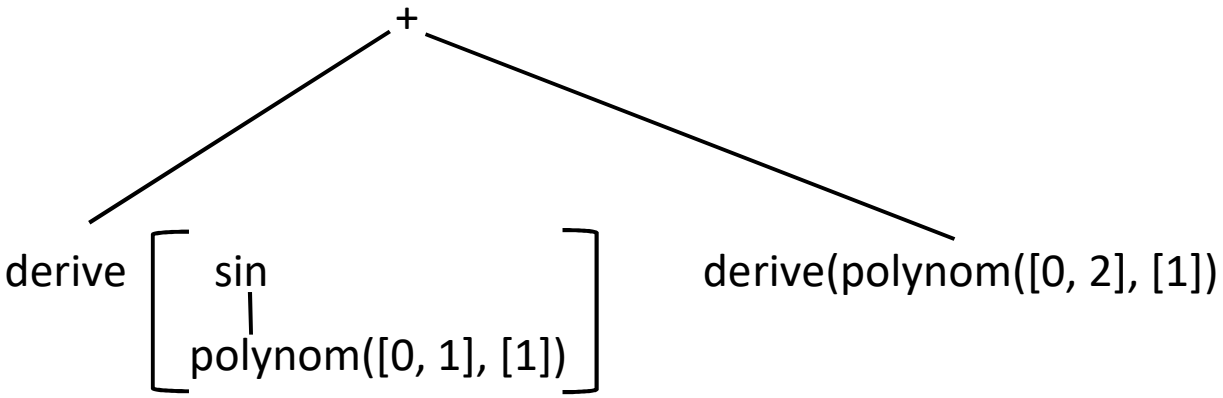
Вторым этапом в решении является упрощение построенного дерева. В рассматриваемом выше примере

дерево после упрощения будет выглядеть следующим образом:



где polynom([0, 2], [1]) – объект класса polynom, специально созданного для хранения рациональных функций.

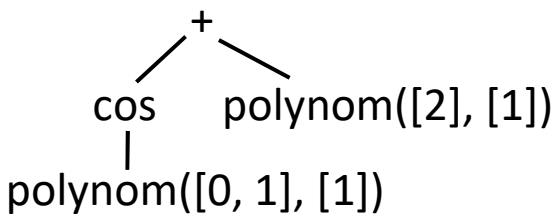
Далее программа приступает непосредственно к дифференцированию. Процедура дифференцирования имеет рекурсивный характер. Изначально она вызывается от корнегого узла и в конце возвращает производную рассматриваемой функции в виде дерева. В нашем примере функция нахождения производной derive() при вызове от корневого узла дерева в соответствии с правилом дифференцирования суммы вернет следующее дерево:



Т.е. функция derive() строит дерево производной в соответствии с правилами дифференцирования, при необходимости вызываясь от поддеревьев текущего узла. При вызове от объекта класса polynom функция derive возвращает объект класса polynom – производную исходного объекта, не вызываясь рекурсивно далее.

После завершения работы функции derive() программа упрощает дерево производной, так же, как это было сделано в этапе 2 (см. выше) с деревом исходной функции.

Финальной частью решения является построение символьной записи производной на основе дерева (операция, обратная синтаксическому анализу). Данную операцию осуществляет функция ptf(), которая имеет рекурсивный характер. Функция изначально вызывается от корневого узла дерева производной и в итоге возвращает строку – символьную запись производной. В нашем примере дерево производной выглядит так:



При вызове от корневого узла функция ptf() вернет следующую строку:

ptf(cos — polynom([0, 1], [1]), далее знак «+», затем – результат функции ptf(polynom([2], [1])).

При вызове от объекта класса polynom функция ptf() возвращает строку – производную рациональной функции, при этом не вызываясь рекурсивно.

Последним шагом является упрощение символьной записи производной с помощью функции simplify(), которая удаляет из строки лишние скобки, идущие подряд знаки «+» (т.е. вместо «2++x» получается «2+ч»), заменяет два подряд идущих знака «-» на один знак «+», и т.д.

В конце программа выводит на экран строку – результат работы функции simplify() на символьной записи производной. Эта строка и является результатом работы программы.

Требования задачи к ОС и машине

Единственным требованием задачи к ОС является возможность запуска программы, написанной на языке Python3 версии не ранее 4.1, т.е. наличие интерпретатора данного языка. Особых требований к производительности машины нет, требуемая оперативная память не превышает 512 мегабайт.

Руководство пользователя

Для начала работы программы необходимо запустить файл программы с помощью интерпретатора Python. В частности, удобный способ запустить программу – открыть файл в среде разработки Wing Ide версии не ранее 5.0. После запуска программы (в Wing IDE для запуска программы необходимо нажать курсором мыши кнопку «run») на экране появится надпись – «Введите функцию:». После появления этой надписи пользователь может ввести функцию в поле ввода в виде строки из допустимых символов (см. Постановка задачи). Для завершения ввода

функции пользователь должен нажать клавишу enter. После этого программа в случае корректного формата ввода выведет функцию, введенную пользователем, в виде строки после обработки программой (чтобы пользователь мог убедиться в том, что программа корректно распознала функцию), и затем, в новой строке – производную введенной пользователем функции. В случае неверного формата ввода, невозможности распознать введенную функцию либо деления на 0 и т.п. программа выведет на экран сообщение об ошибке. В обоих случаях по выполнении вышеописанных процедур программа завершит работу.

Комментированный код программы

В силу того, что код программы на языке Python.3.4 занимает 700 строк и представляется слишком объемным для помещения в отчет, здесь будут о общих чертах разобраны функции и методы, использующиеся в программе, без предоставления кода самой программы (см. Внутренняя спецификация программы). Текстовый файл с расширением .py с полным кодом программы можно найти в приложенном электронном носителе. Также существует версия на английском.

Внутренняя спецификация программы

Решение задачи начинается с построения дерева на основе полученного на вход математического выражения. Для этого используется функция parse() (строки 548 – 660). Данная функция берет на вход строку, а возвращает объект одного из классов функций (см. дальше) – корневой узел дерева. Сначала Функция parse() сканирует строку на предмет наличия суммы/разности функций. В случае успеха parse() строит разбивает строку на 2 части в том месте, где стоит знак арифметического действия, и создает новый узел дерева - объект класса mixed_func(l1, l2, mark), где mark – операция (строка вида «+» или «-»), l1, l2 – левая и правая подстроки исходной строки. При инициализации объекта класса mixed_func у данного объекта создаются атрибуты self.f1 и self.f2 – результаты работы функции parse(l1) и parse(l2) соответственно. Таким образом, parse() завершается созданием нового узла дерева, а при создании рекурсивно вызывает себя от правой и левой подстрок исходной строки.

После этого аналогично происходит сканирование на предмет наличия произведения и отношения двух функций (создается объект вида mixed_func(l1, l2, «*») или mixed_func(l1, l2, «/»).

Далее parse() ищет степенные, показательные и степенно-показательные функции. В случае успеха создается объект класса power либо polynom.

После этого происходит поиск на предмет наличия тригонометрических функций и логарифмов.

В конце проверяется, является ли строка числом или переменной x. В случае успеха parse() возвращает объект класса polynom, при инициализации которого parse() не вызывается. Если ни то, ни другое не верно, parse() осуществляет выход из рекурсии и меняет значение переменной F, отвечающей за корректность ввода данных, с True на False, что приводит к выводу на экран сообщения об ошибке.

При создании узлов функция parse() может создавать объекты следующих классов:

mixed_func (строки 195 – 250) – представляет арифметические операции +-*/

power (строки 178 – 193) – показательные, степенные, степенно-показательные функции

trig (строки 107 – 125) – тригонометрические функции

log (строки 127 – 144) – логарифмы по основаниям – натуральным числам

ln (строки 157 – 169) – натуральные логарифмы

polynom (строки 1 – 91) – рациональные функции

Каждый из этих классов (кроме polynom) содержит свои узлы – потомки в виде атрибутов (self.arg у trig, self.base и self.power у power и т.д.)

Кроме того, для каждого класса работают методы:

__init__() (инициализация объекта класса)

derive() – построение дерева производной

ptf() – построение символьной записи объекта класса

В соответствии с алгоритмом решения, далее дерево, полученное в результате работы функции parse(), упрощается посредством функции count() (строки 401 – 420). Данная функция основана на функциях add() (строки 424 – 439), sub() (строки 442 – 450), multiply() (строки 453 – 481), divide() (строки 484 – 515), осуществляющих для двух узлов дерева сложение, вычитание, умножение и деление соответственно. В зависимости от типов узлов данные функции работают по-разному. В частности, в случае подсчета результата для двух объектов класса polynom данные функции обращаются к библиотеке методов для работы с классом polynom (строки 277 – 329), которая, в свою очередь, основана на библиотеке для работы с простыми дробями (строки 333 – 401).

Также функция count() в некоторых случаях обращается к функции eq() (строки 520 – 547), которая проверяет два узла на эквивалентность (например, это необходимо при делении).

Далее в действие вступает метод derive(), описание которого для каждого типа узлов уникально и находится в описании соответствующего класса. Данный метод, так же, как и count(), во многом опирается на библиотеку для работы с классом polynom.

После построения дерева производной методом derive() происходит упрощение дерева методом count().

Для получения строки - символьного представления производной - используется метод ptf(), описание которого для каждого типа узлов уникально и находится в описании

соответствующего класса. Полученную строку обрабатывает функция simplify() (строки 668 – 689).

Финальная часть программы (строки 692 – 702) – команды для взаимодействия с пользователем и вызов основных функций (parse(), count(), derive(), ptf()).

Процесс тестирования

Для достижения правильной и оптимальной работы программы в качестве тестов использовались различные функции, известные из курса математического анализа 10 класса. Правильность работы программы тестировалась при помощи интернет-ресурса wolframalpha.com.

Примеры тестов:

Входная функция	Результат работы программы
f(x) = ln(x)/sin(x)	(sin(x)*1/x-ln(x)*cos(x))/((sin(x))^2)
cos(sin(x))	-cos(x)*sin(sin(x))
(x + 1)^2/(x^2 - 1)	-2/(x^2-2x+1)
tan(x + sin(x))	(1+cos(x))/((cos(x+sin(x)))^2)

Использованная литература

[1] Статья о написании компилятора с интернет-ресурса habrahabr.ru (ссылка: https://habrahabr.ru/post/133780/)

Оглавление