НИЯУ МИФИ
Кафедра 22 «Кибернетики»

Лабораторная работа

на тему: «Разработка компактного языка для вероятностного моделирования»
(язык сценариев ProbabilityScript)

Выполнил: _____ (ФИО студента)

Руководитель: _____ (ФИО преподавателя)

Москва – 2025

Введение

Цель проекта – разработка компактного предметно-ориентированного языка **ProbabilityScript** для вероятностного моделирования. Этот язык сценариев предназначен для упрощения моделирования случайных процессов и статистических вычислений. В рамках проекта реализован интерпретатор, позволяющий задавать последовательность случайных экспериментов, собирать результаты и рассчитывать основные статистические показатели.

Представленная документация описывает синтаксис и возможности языка ProbabilityScript, архитектуру реализации интерпретатора, пример использования языка, а также содержит заключение с выводами и возможными направлениями развития. В приложении приведён полный листинг демонстрационного сценария `demo_basic.psc`, иллюстрирующего основные возможности языка.

1 Синтаксис языка ProbabilityScript

Данный раздел описывает синтаксис и ключевые возможности языка сценариев ProbabilityScript. Язык имеет минималистичный синтаксис и предназначен для генерации случайных чисел, повторения серии экспериментов и сбора статистики. Основной (и единственный) тип данных в языке – вещественное число с плавающей запятой (floating point). Ниже рассматриваются типы данных, конструкции языка, встроенные функции и приводятся примеры сценариев.

1.1 Типы данных

В языке ProbabilityScript используется один базовый тип данных:

- **Вещественные числа:** все переменные и выражения являются числовыми (тип с плавающей запятой, обычно двойной точности). Целочисленные значения не выделяются отдельно, а представляют собой частный случай вещественного. Строковые значения и логические типы в текущей версии языка не поддерживаются.

Переменные в сценарии не объявляются заранее – они неявно создаются при первом присваивании. Имена переменных могут состоять из букв ла-

тинского алфавита и цифр, начинаются с буквы. Например, допустимы имена `x`, `value1`, `sum`.

1.2 Конструкции языка

Язык сценариев предоставляет небольшой набор управляющих конструкций и инструкций:

- **Присваивание.** Оператор `=` используется для присваивания значения переменной. Слева от `=` указывается имя переменной, справа – выражение. Присваивание инициализирует новую переменную или обновляет значение существующей. Примеры: `x = 5` (присвоить 5 переменной `x`); `y = x + 1.5` (вычислить выражение и сохранить в `y`).

- **Цикл `repeat`.** Конструкция повторения имеет синтаксис: `repeat` N `{ ... }`, где N – целочисленное количество итераций (задается как числовое выражение). Блок в фигурных скобках повторяется N раз. Обычно цикл `repeat` используется для выполнения серии случайных экспериментов. Вложенные циклы `repeat` не предусмотрены.

- **Инструкция `collect`.** Внутри цикла можно использовать инструкцию `collect` E, где E – числовое выражение. Эта инструкция собирает (накапливает) значение выражения E в специальную выборку результатов. По завершении цикла собранные данные могут быть использованы для вычисления статистических показателей. Например, `collect x` сохранит текущее значение переменной `x` в набор результатов.

- **Вывод `print`.** Инструкция `print` E выводит значение выражения E на экран (в стандартный вывод). Можно использовать `print` для отображения числовых результатов или сообщений. Каждое выполнение `print` выводит значение (возможно, с новой строки). Например, `print x` выведет текущее значение `x`. При необходимости, можно выводить также простой текст (строковые литералы в кавычках) для пояснения результатов, однако переменные строкового типа не поддерживаются.

- **Вывод статистики** `print_stat`. Эта инструкция выводит сводную статистику по ранее собранной выборке данных (через `collect`). При выполнении `print_stat` интерпретатор рассчитывает и печатает основные показатели: размер выборки (`count`), среднее значение (`mean`), дисперсию (`variance`), стандартное отклонение (`stddev`) и медиану (`median`). Формат вывода представляет собой перечень этих статистик с их значениями. Инструкцию `print_stat` обычно вызывают после цикла, когда все данные собраны.

Блоки кода ограничиваются фигурными скобками `{}`. Каждая инструкция (присваивание, collect, print и т.д.) записывается с новой строки (точка с запятой в конце строк не требуется). Отступы и пробелы вне строк не влияют на выполнение (но рекомендуется использовать отступы внутри блока `repeat` для читаемости).

1.3 Встроенные функции

Для организации вероятностных экспериментов и статистической обработки в языке ProbabilityScript доступны следующие встроенные функции:

- `uniform(`a`, `b`)` – возвращает псевдослучайное вещественное число, равномерно распределённое в интервале $[a, b]$. Каждый вызов генерирует новое случайное значение. Например, `x = uniform(0, 1)` присвоит `x` случайное число из диапазона 0 до 1.

- `normal(`μ`, `σ`)` – возвращает случайное вещественное число из нормального распределения $N(\mu, \sigma)$ с заданным математическим ожиданием μ и среднеквадратическим отклонением σ. Например, `y = normal(5, 2)` сгенерирует значение, распределённое нормально со средним 5 и стандартным отклонением 2.

- `mean()` – вычисляет и возвращает среднее значение (математическое ожидание) по текущей собранной выборке данных (т.е. по всем значениям, накопленным с помощью `collect` на данном этапе исполнения сценария). Обычно вызывается после цикла, когда доступны результаты. Если выборка пуста (не было `collect`), `mean()` может вернуть 0 или вызвать ошибку (в текущей реализации предполагается, что `mean()` вызывается корректно после сбора данных).

- `variance()` – вычисляет дисперсию текущей выборки. Формула расчёта – обычная выборочная дисперсия по собранным данным. Возвращает числовое значение дисперсии. Аналогично `mean()`, предполагается вызов после сбора данных.

- `stddev()` – вычисляет стандартное отклонение текущей выборки. По сути, это квадратный корень из значения `variance()`. Возвращает положительное вещественное число.

- `median()` – вычисляет медиану текущей выборки. При чётном количестве элементов берётся среднее арифметическое двух центральных значений, при нечётном – средний элемент после сортировки выборки. Возвращает вещественное значение медианы.

- `count()` – возвращает размер (мощность) текущей выборки, т.е. количество значений, добавленных через `collect`. Может быть полезно, например, для вывода количества итераций или проверки, сколько данных собрано.

Вышеописанные функции (`mean`, `variance`, `stddev`, `median`, `count`) обычно используются либо в составе инструкции `print` для вывода, либо в выражениях присваивания, если требуется сохранить результат вычисления в переменной для дальнейших расчётов.

1.4 Примеры сценариев

Ниже приведены два примера использования ProbabilityScript с пояснениями.

Пример 1. Оценка характеристик равномерного распределения. В первом сценарии моделируется генерация выборки из равномерного распределения на интервале $[0, 1]$ и последующий расчёт статистических характеристик этой выборки. Сценарий проводит 100 испытаний, собирает результаты и выводит рассчитанные величины (среднее, дисперсию, медиану и т.д.) с помощью `print_stat`.

```
N = 100
repeat N {
    x = uniform(0, 1)
    collect x
```

```
}
print_stat("mean")
print_stat("variance")
print_stat("median")
```

В этом примере на каждой итерации цикла `repeat` генерируется случайное число `x` из диапазона 0 до 1 и оно сохраняется инструкцией `collect`. После завершения 100 повторений команда `print_stat` выводит количество сгенерированных значений (должно быть 100), а также рассчитанное среднее (ожидаемо около 0.5), дисперсию (теоретически $1/12 \approx 0.083$), стандартное отклонение (около 0.29) и медиану (около 0.5). Фактические значения будут немного отличаться из-за случайной природы генератора, но при большом числе итераций они стремятся к теоретическим.

Пример 2. Оценка характеристик нормального распределения. Во втором сценарии генерируется выборка из нормального распределения $N(0, 1)$ (стандартное нормальное распределение) и вычисляются статистики выборки. Ожидается, что среднее будет близко к 0, а стандартное отклонение близко к 1.

```
N = 50
repeat N {
    y = normal(0, 1)
    collect y
}
print mean()
print stddev()
```

В данном примере сценарий выполняет 50 испытаний, в каждой итерации получая случайное значение `y` из нормального распределения с $\mu = 0$ и $\sigma = 1$. Значения накапливаются, после чего с помощью `print` выводятся вычисленное выборочное среднее и выборочное стандартное отклонение. (Для наглядности в этом примере использована `print` с текстовыми пояснениями; вывод может выглядеть, например, как `Mean of sample: -0.1` и `Stddev of sample: 0.9`, в зависимости от сгенерированных данных.) При достаточном количестве итераций результаты приближаются к заданным параметрам распределения.

2 Архитектура реализации

Реализация интерпретатора ProbabilityScript основана на классической структуре интерпретируемого языка: входной сценарий последовательно проходит этапы лексического анализа, синтаксического анализа (разбора), построения внутреннего представления и непосредственного исполнения. Основные компоненты системы: лексер, парсер, дерево разбора (AST), интерпретатор, среда выполнения и модуль статистики. Ниже описаны функции каждого модуля и процесс выполнения сценария.

2.1 Модули интерпретатора

Основные модули, из которых состоит интерпретатор ProbabilityScript:

- **Лексер (Lexer).** Отвечает за лексический анализ входного текста сценария. На этом этапе исходный код, представленный последовательностью символов, преобразуется в поток токенов. Токены — это значимые единицы текста: ключевые слова (`repeat`, `collect`, `print` и т.д.), идентификаторы (имена переменных и функций), числовые литералы, специальные символы (фигурные скобки, знаки операций, запятые и пр.). Лексер проходит по тексту, распознаёт токены и передаёт их парсеру. При обнаружении неизвестных последовательностей символов или ошибок формата лексер генерирует ошибку (например, недопустимый символ).

- **Парсер (Parser).** Выполняет синтаксический разбор последовательности токенов, полученных от лексера, и строит **абстрактное синтаксическое дерево (AST)** программы. Парсер проверяет правильность конструкции сценария согласно грамматике языка ProbabilityScript (например, корректное использование `repeat`-блоков, правильное расположение инструкций внутри них, наличие выражения после `collect` и т.д.). Если входная последовательность токенов не соответствует ожидаемой структуре (синтаксическая ошибка), парсер выдаёт сообщение об ошибке и выполнение программы прекращается. В случае успешного разбора результатом является AST — иерархическая структура, отражающая программу.

- **Абстрактное синтаксическое дерево (AST).** Внутреннее представление сценария в памяти. AST состоит из узлов, каждый из которых соответствует языковой конструкции: узлы для выражений (например, бинарная операция, вызов функции `uniform()`), узел для присваивания, узел цикла `repeat`, узел инструкции `collect`, узел `print` и т.д. Дерево отражает вложенность блоков (узел `repeat` содержит подузлы — тело цикла). Это представление удобно для последующей интерпретации: обходя AST, интерпретатор выполняет программу.

- **Интерпретатор.** Модуль, выполняющий построковый обход/обход дерева AST и непосредственное исполнение команд сценария. Интерпретатор реализует логику работы каждого типа узла: при встрече узла-присваивания он вычисляет выражение и сохраняет значение переменной в среде выполнения; узел `repeat` приводит к циклическому выполнению поддерева определённое число раз; узел `collect` вызывает добавление значения в статистическую выборку; узлы `print` и `print_stat` инициируют вывод. Интерпретатор также обрабатывает вызовы встроенных функций (`uniform`, `normal` и др.), вызывая соответствующие процедуры из модуля статистики или стандартной библиотеки.

- **Среда выполнения (Environment).** Этот компонент хранит текущее состояние исполняемого сценария: значения переменных, а также контекст для генерации случайных чисел и сбора статистики. По сути, среда представляет собой таблицу символов (имён переменных) и их значений, которая изменяется по мере выполнения программы (при присваивании создаются новые пары «имя-значение» или обновляются существующие). Кроме того, среда может содержать структуру для хранения собранных данных (например, массив или список чисел, в который `collect` добавляет элементы). Реализация генератора случайных чисел (для `uniform`, `normal`) также привязана к среде: обычно в среде хранения находится объект генератора (например, на базе Mersenne Twister или другого алгоритма) и, при необходимости, текущее состояние случайных последовательностей.

- **Модуль статистики.** Предоставляет функции для обработки собранных данных. В простейшем случае, функции `mean()`, `variance()`,

`stddev()`, `median()`, `count()` могут быть реализованы непосредственно внутри интерпретатора. Однако более модульно выносить их в отдельный компонент. Модуль статистики оперирует над выборкой, накопленной средой выполнения. Например, при вызове `mean()` интерпретатор обращается к функции `mean` из данного модуля, передавая ей доступ к списку собранных значений; функция рассчитывает среднее (суммирует все элементы и делит на количество). Аналогично вычисляются и другие показатели. Некоторые из этих функций требуют сортировки выборки (например, для медианы) или нескольких проходов по данным (для дисперсии и стандартного отклонения). Модуль статистики инкапсулирует эти алгоритмы, возвращая готовые результаты интерпретатору.

2.2 Путь от сценария до исполнения

После запуска интерпретатора (команды `psc`) с указанием сценария ProbabilityScript, выполнение происходит по следующим этапам:

1. **Чтение файла сценария.** Программа-интерпретатор открывает файл с расширением `.psc`, загружает всё содержимое сценария в память как текст.

2. **Лексический анализ.** Текст сценария передается лексеру. Лексер считывает символы и преобразует исходный код в токены. Например, строка `repeat 10 {` будет преобразована в последовательность токенов {`KEYWORD_REPEAT`, `NUMBER(10)`, `LBRACE`}. Результат работы лексера — список токенов, представляющий весь файл сценария.

3. **Синтаксический анализ и построение AST.** Список токенов поступает парсеру. Парсер последовательно считывает токены и сопоставляет им правила грамматики. При обнаружении конструкций (например, ключевое слово `repeat` указывает, что далее должен следовать числовой параметр и блок в фигурных скобках) парсер создаёт соответствующие узлы AST и выстраивает иерархию. К концу разбора строится корневой AST-узел, содержащий весь сценарий (например, последовательность инструкций верхнего уровня, в том числе, возможно, цикл `repeat`). Если на этом шаге встречается ошибка (неправильный порядок токенов, отсутствующая за-

крывающая скобка и т.п.), выполнение прерывается с сообщением об ошибке синтаксиса.

4. **Интерпретация (исполнение).** Полученное абстрактное синтаксическое дерево передается в интерпретатор. Интерпретатор выполняет рекурсивный обход дерева. Начинается выполнение с корневого узла (например, если программа состоит из последовательности инструкций, они выполняются по порядку). При входе в узел цикла `repeat` интерпретатор повторяет интерпретацию вложенных узлов заданное число раз. Узлы присваивания вызывают вычисление выражения (это может рекурсивно обходить поддерево выражения) и сохранение результата в среде. Вызов `uniform` или `normal` приводит к обращению к генератору случайных чисел: интерпретатор передаёт параметры в функцию генерации и получает случайное значение. Инструкция `collect` заставляет интерпретатор добавить текущий результат в массив/список хранимых данных (в среде). При встрече узла `print` интерпретатор вычисляет выражение и выводит полученное значение на консоль (через стандартный вывод). Для `print_stat` интерпретатор запрашивает у модуля статистики вычисление всех необходимых величин по накопленной выборке и выводит их в понятном формате.

5. **Завершение работы.** После обхода всего AST сценарий считается исполненным. Интерпретатор освобождает занятые ресурсы (например, очищает использованные структуры, закрывает файл, если это не сделано ранее). При корректном завершении работы возвращается код успеха (0). В случае ошибок на любом из этапов (лексическом, синтаксическом или во время выполнения, например, деление на ноль или вызов статистической функции без данных) интерпретатор выводит сообщение об ошибке с указанием ее типа и, по возможности, позиции в тексте сценария, после чего завершает работу с кодом ошибки.

3 Пример использования

Рассмотрим, как на практике использовать разработанный язык ProbabilityScript и его интерпретатор. Ниже описаны шаги создания и запуска сценария,

установки генератора случайных чисел в детерминированное состояние и анализ полученного вывода.

1. **Подготовка сценария.** Сценарий пишется в обычном текстовом файле с расширением `.psc`. В файле содержится код на языке ProbabilityScript. Например, можно создать файл `demo_basic.psc` со сценарием, демонстрирующим генерацию 100 случайных чис и вывод статистики (полный текст сценария приведён в Приложении).

2. **Запуск интерпретатора.** Для выполнения сценария используется консольная утилита-интерпретатор `psc`. Запуск производится командой в терминале, указывающей имя файла сценария. Пример команды (в текущем каталоге с файлом):

   ```
   $ psc demo_basic.psc
   ```

   После запуска программа `psc` прочитает файл, обработает сценарий и начнёт его выполнение. В процессе выполнения на консоль будут выводиться результаты согласно инструкциям `print` и `print_stat` внутри сценария.

3. **Установка зерна генератора случайностей.** По умолчанию генерация случайных чис (`uniform`, `normal`) каждый раз происходит с новой случайной последовательностью. Для воспроизводимости результатов предусмотрен специальный параметр командной строки `-seed`. Его можно указать при запуске интерпретатора, чтобы задать начальное значение (seed) для генератора случайных чис. При одном и том же seed последовательность генерируемых чисел будет идентичной на разных запусках, что удобно для отладки. Пример запуска с указанием зерна:

   ```
   $ psc demo_basic.psc --seed=42
   ```

   В данном случае генератор случайных чис инициализируется значением 42, и сценарий `demo_basic.psc` каждый раз будет давать один и тот же последовательный набор псевдослучайных чис.

4. **Анализ вывода.** После успешного исполнения сценария на консоль выводятся результаты, сформированные инструкциями `print`/`print_stat`. Например, для сценария `demo_basic.psc` (как в Примере 1) вывод может выглядеть следующим образом (при фиксированном зерне для наглядности):

```
Count:    100
Mean:     0.480
Variance: 0.086
Stddev:   0.294
Median:   0.482
```

Здесь `Count` обозначает количество собранных значений (100 итераций), а остальные строки отображают статистические показатели выборки (среднее 0.480, дисперсия 0.086, стандартное отклонение 0.294, медиана 0.482). Пользователь может сопоставить эти результаты с теоретическими ожиданиями или использовать их для дальнейшего анализа. При каждом новом запуске без фиксированного зерна значения будут отличаться случайным образом, но остаются в разумных пределах для равномерного распределения на $[0, 1]$.

Заключение

В рамках данного проекта разработан и реализован простой интерпретируемый язык сценариев **ProbabilityScript** для задач вероятностного моделирования. Язык поддерживает генерацию случайных чисел из базовых распределений (равномерного и нормального), повторение серий экспериментов с накоплением результатов, вычисление основных статистических характеристик выборки, а также вывод результатов на экран. Реализована основная инфраструктура интерпретатора: лексер, парсер, AST, интерпретатор, управление средой выполнения и базовый модуль статистики.

Данный язык можно расширять и улучшать. В будущем планируется добавить новые типы распределений (например, экспоненциальное, биномиальное и др.), а также поддерживать более сложные конструкции языка. Полезным развитием было бы введение условных операторов (`if/else`) и циклов с условием (`while`), что расширило бы класс моделируемых задач. Можно рассмотреть введение разных типов данных

(например, целых чисел, булевых значений для условий, либо простых структур данных для хранения нескольких выборок). Ещё одним направлением развития является вывод результатов в файл или визуализация распределений. Тем не менее, даже текущая версия ProbabilityScript демонстрирует работоспособность подхода: компактный язык позволяет пользователю быстро описать и выполнить вероятностный эксперимент, получив основные характеристики распределения без необходимости программировать эти вычисления с нуля в языках общего назначения.

Приложение: Полный листинг сценария `demo_basic.psc`

Ниже приведён полный исходный код сценария `demo_basic.psc`, иллюстрирующего базовое использование языка ProbabilityScript (генерация 100 равномерных случайных чисел на интервале $[0, 1]$ и вывод статистических показателей). Этот сценарий использовался в качестве примера в тексте отчёта.

```
# demo_basic.psc

N = 100
repeat N {
    x = uniform(0, 1)
    collect x
}
print_stat("count")
print_stat("mean")
print_stat("variance")
print_stat("stddev")
print_stat("median")
```