Санкт–Петербургский государственный университет
Факультет математики и компьютерных наук

Алексей Лесник Алексеевич

Выпускная квалификационная работа

Тема работы: Разработка метода анализа изменения производительности ПО на основе хвостов распределений замеров скорости работы

Уровень образования: бакалавриат
Направление 01.03.02 «Прикладная математика и информатика»
Основная образовательная программа СВ.5156.2019
«Современное программирование»

Научный руководитель:
доцент, факультет математики и компьютерных наук, к.ф.-м.н. Д. С. Шалымов

Рецензент:
Инженер по тестированию
ООО «Яндекс Технологии»
С. А. Бальзитова

Санкт-Петербург
2023 г.

Содержание

Введение

Тестирование программного обеспечения является неотъемлемой частью процесса разработки ПО. Оно позволяет не только убедиться в корректности работы определённых модулей или даже целых программ, но и отслеживать время за которое они работают. Первым этапом, с которым сталкиваются разработчики, является создание прототипа продукта и исправление ошибок в поведении программы. Второй же этап это оптимизация скорости работы созданной программы. В зависимости от задачи ПО второй этап может как отсутствовать, так и занимать большую часть процесса разработки. В случае, когда данный этап достаточно долгий или вообще не заканчивается, хорошей практикой считается создать автоматическое тестирование производительности. Данное тестирование позволяет отслеживать прогресс оптимизации, как по отдельным модулям, так и по сценариям работы ПО.

В сфере производительности ПО и находится тема данного дипломного проекта. Для автоматического анализа результатов тестов создано немало библиотек [1.2], которые имеют инструменты для отслеживания времени работы. Функционал такого отслеживания практически у всех этих библиотек останавливается на возвращении времени работы на каждом тесте, если те работают и не превысили временное ограничение заданное тестировщиком. К несчастью фиксированные ограничения на время работы очень редко связанны с реальным временем выполнения теста, а иногда вообще остаётся по умолчанию. Из-за этого плохо работающая программа может пройти тестирование и надежда заключается в том, что последующий анализ времени работы выявит проблему.

В ходе выполнения этого проекта будут исследованы подходы и эвристики, позволяющие автоматически определять является ли время, которое работает тест недопустимым. В результате исследований будет создан метод позволяющий автоматически оценивать временные ограничения для тестов.

Постановка задачи

Данная работа заключается в поиске метода, позволяющем автоматически определять является ли изменение допустимым на основании данных о предыдущем тестировании, при этом учитывая желаемую частоту срабатывания при отсутствии изменений.
Этапы выполнения данной задачи включают:

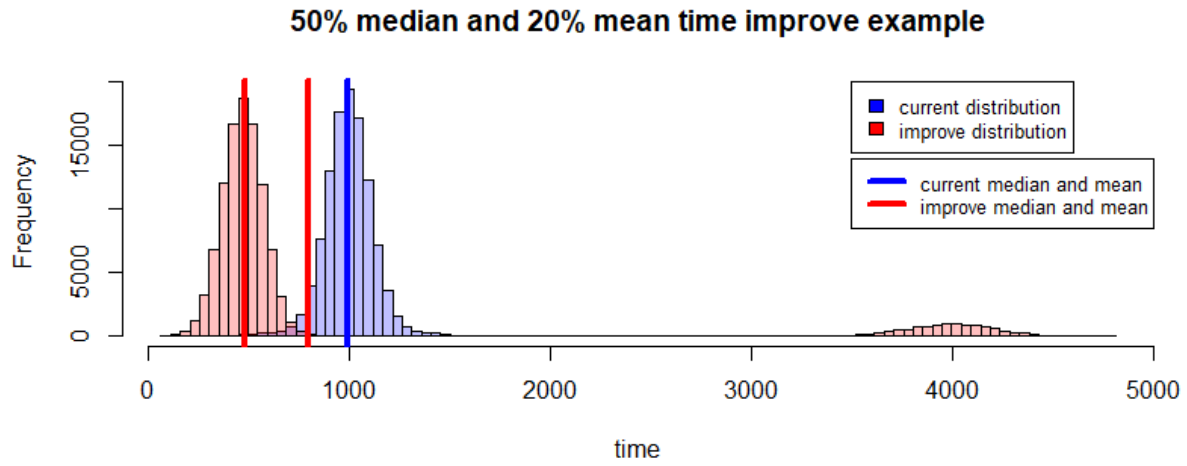
- Изучение подходов к тестированию производительности программного обеспечения.
- Изучение задачи и приведение её до строгой математической формы.
- Поиск возможных методов решения подобных задач в других областях.
- Сравнение найденных подходов.
- Использование полученных сравнительных данных для создания программы способной автоматически решать задачу поиска временного ограничения наилучшим способом из рассмотренных.

1. Предметная область и подход к тестированию

1.1. Мотивация

Для должной мотивации как проявляется проблема с точки зрения взаимодействия пользователя и ПО. Пользователь знает, что некоторая операция работает менее чем за какое-то ожидаемое им время. Заметное увеличение времени это крайне нежелательное событие, которое мы хотим избежать, так как пользователь может негативно отреагировать на изменение и перестать пользоваться нашими услугами. Для мобильных платформ так же появляется риск отрицательного отзыва от пользователя.

Рассмотрим теперь с точки зрения процесса разработки ПО.
Рассмотрим на примере:
Тестировщик создал тест на некоторый модуль ПО. Он добавил возможность отслеживание среднего времени работы, отслеживание медианного, при этом зафиксировав временное ограничение на тест со значительным запасом. Через какое-то время разработчик нашёл способ оптимизировать эту часть программы. После чего загрузил своё обновление в тестирующую систему, которая сообщила ему, что улучшение среднего 20%, улучшение медианы 50%. Посмотрим на распределение, которое может скрываться за таким изменени-



ем.
В данном примере 10% пользователей получили замедление данного модуля в 4 раза. В зависимости от важности конкретного модуля это может привести к различным последствиям вплоть до потерь в аудитории или рейтинге.

Проблема представленная в этом примере заключается не в том, что

существуют улучшения по некоторым статистикам, которые могут дать негативный эффект в реальной жизни, а в том, что разработчик не получил полноценной информации от тестирующей системы, так как временное ограничение для теста было сильно завышено. Для того, чтоб подобные ситуации не происходили нужно создавать системы с автоматической обработкой времени работы теста. В случае превышения тест упадёт и разработчик будет оповещён, что потенциально в модуле появилась ошибка.

Проблема задавания фиксированного временного ограничения не останавливается на неверно поставленном значении. Проблема развивается с последовательным улучшением ПО и соответственно деградацией качества временного ограничения.

1.2. Обзор конкурентов

В данный не существует популярных открытых прямых конкурентов, которые использовали бы автоматические ограничения по времени для тестов. Популярные программы для тестирования используют константное время на тест, задаваемое тестировщиком. К таким относятся:

- Jestjs – система временных ограничений выставляемых пользователем, причём по умолчанию стоит на 5 секунд.
- Selenium – аналогичная система Jestjs, но есть несколько видов тестов с разным ограничением по времени по умолчанию или вовсе без него.
- MSTest – аналогичная система Jestjs с ограничением по умолчанию в 30 минут.

Эти примеры ПО для тестирования лишь часть из огромного множества и как правило значение определять максимальное время работы теста возложено на человека создающего тест.

1.3. Глобальная математическая модель

Для создания математической модели необходимо определиться с терминами и обозначениями:

- X – распределение из допустимых изменений, которые не должны повлиять на ощущения конечного пользователя.

- B – распределение, содержащее плохие изменения. Это изменения, которые ухудшат опыт взаимодействия пользователя с приложением в определённом проценте случаев. Именно эти изменения в рамках данной задачи желательно избегать.

- p – желательная минимальная вероятность вызова персонала при допустимых изменениях. Это вероятность определяется тем как много в команде программистов и тестировщиков готовых проверять ложные падения тестов.

- $A(x_1, ..., x_n, b, p)$ – алгоритм, который выдаёт 0, если тест должен упасть и необходимо вызвать человека и 1 если всё хорошо по его мнению.

Тогда глобальная математическая модель задачи выглядит так:

$$\mathbf{E}_X A(X_1, ..., X_n, X_{n+1}, p) \leq p$$

$$\min_A \mathbf{E}_{X,B} A(X_1, ..., X_n, B, p)$$

Если описывать требования словами, то необходимо, чтоб тест падал в режиме без изменений не чаще чем некоторая вероятность и так же желательно, чтоб алгоритм отлавливал как можно больше плохих изменений.

1.4. Математическая модель

Но в данной работе рассматривается менее общий случай. А именно более узкое семейство алгоритмов, которое помогает значительно упростить задачу тестирования алгоритмов в рамках этого проекта. Так же избавит нас от зависимости от B, которое нам не известно. И дополнительно поможет сделать конечный метод совместимым с многими ПО для тестирования, так как будет непосредственно предсказывать временное ограничение.

Изменение, которое предлагается рассматривать это преобразование $A(x_1, ..., x_n, b, p)$ в более узкое семейство $[b \geq F(x_1, ..., x_n, p)]$, путём удаления

элемента b из параметров. То есть предлагается предсказание временного ограничения для теста, после чего будет проведено сравнение.

Рассмотрим как меняется математическая модель, если применить это изменение:

$$\mathbf{E}_X A(X_1, ..., X_n, X_{n+1}, p) \leq p$$

$$\min_A \mathbf{E}_{X,B} A(X_1, ..., X_n, B, p)$$

$$\mathbf{E}_X \left[X_{n+1} \geq F(X_1, ..., X_n, p)\right] \leq p$$

$$\min_A \mathbf{E}_{X,B} B \geq F(X_1, ..., X_n, p)$$

К первому неравенству к левой части под математическим ожиданием можно применить X^{-1}, где X^{-1} - функция обратная распределению X, так как она монотонна и не повлияет на бинарное отношение под математическим ожиданием.

И теперь можно просто убрать B из второй формулы и минимизировать $F(X_1, ..., X_n, p)$, так как в новых ограничениях это одно и тоже.

$$\mathbf{E}_X \left[X^{-1}(X_{n+1}) \geq X^{-1}(F(X_1, ..., X_n, p))\right] \leq p$$

$$\min_F \mathbf{E}_X F(X_1, ..., X_n, p)$$

$$\mathbf{E}_X X^{-1}(F(X_1, ..., X_n, p)) \geq 1 - p$$

$$\min_F \mathbf{E}_X F(X_1, ..., X_n, p)$$

Что приводит к условию, что хочется, чтоб $F(X_1, ..., X_n, p)$ выдавал значения, которые более экстремальные чем квантиль $1 - p$. Но при этом вторая строчка требует минимизировать выдаваемые значения. Из этого можно заключить, что идеальным ответом для текущей задачи является квантиль $1 - p$ распределения X, который необходимо получить по выборке.

1.5. Проблемы модели и существующих решений

Полученная математическая модель имеет один существенный недостаток, который не позволяет просто использовать квантиль. Дело в том, что многие методы по поиску квантилей построены так, что:

$$\mathbf{E}_X(F(X_1, ..., X_n, p)) = quantile(X, 1 - p)$$

, где $quantile$ – функция квантиля от распределения и вероятности.

Что подходит под математическую модель, если стандартное отклонение предсказания будет нулевым, но к несчастью это не так, а значит распределение ошибки нужно будет так же учитывать для этой задачи, так как после перевода из чисел в доли математическое ожидание сместится в меньшую сторону.

1.6. Особенности среды тестирования

Ссылка на хранилище кода: GitHub

Наиболее реалистичными параметрами размера выборки было принято 100 элементов, и вероятность падения теста по ограничению на время работы в случае если ничего не произошло $\frac{1}{1000}$.
Для тестирования найденных алгоритмов достаточно уметь генерировать данные по выборке и иметь обратную функцию на большем хвосте распределения. Разумеется, со сгенерированными данными по выборкам можно просто воспользоваться обратной функцией распределения, но для реальных данных было решено использовать большую выборку и линейную аппроксимацию по 7 ближайшим известным элементам.
Для хорошей аппроксимации размер выборки для тестирования на реальных данных был зафиксирован на размере 50000, что покрывает целевой квантиль примерно 50 элементами. А так же проводится по 10000 тестов, которые обеспечивают схождение метрик.
Искусственными данными, использованными для тестирования являются:

- norm – нормальное распределение с рандомными параметрами. Необ-

ходимо оговориться, что на самом деле неважно положительные или отрицательные числа и даже есть ли хвост в $-\infty$, так как все алгоритмы использованные в работе работают со всеми распределениями.

- lnorm – логарифм от нормального распределения тоже с рандомными параметрами.
- stud – распределение Стьюдента с рандомными параметрами.
- cauchy – распределение Коши с рандомными параметрами.

Реальные данные для теста:

- ping – это время за которое рандомные адреса отвечали на запрос ping с верхней границей в 7 секунд (да, такие тоже есть).
- file – это время записи в файл, ничем не ограниченное.
- pca – это время работы алгоритма главных компонент на 50000 элементах, ничем не ограниченное.
- sort – это время сортировки случайного массива в 10000 элементов, ничем не ограниченное.

2. Базовые алгоритмы поиска квантилей

2.1. Максимум выборки как квантиль

Обозначение на графиках: max

Казалось бы как максимум может быть приближением квантиля, но оказывается может. Если рассмотреть стандартные квантили по выборке, они, как правило, используют два ближайших элемента и линейную аппроксимацию, построенную на них. К несчастью такие квантили не подходят и более того многие из них, при квантиле выходящем за размер выборки, даже не достигают максимума, в то время как задача работы состоит в анализе поведения распределения за максимумом. Естественным первым приближением получается максимум. Хорошее свойство данного метода заключается в том, что довольно просто определить к какому же квантилю распределения соответствует выборочный максимум, а именно $\frac{n}{n+1}$, что в данной работе равняется $\frac{100}{101} \approx 0.99001$.

При целевом квантиле в 0.999 это не является большим достижением, но будет являться критерием минимально возможного качества. И не маловажное свойство максимума простота в расчёте.

А так же именно этот подход даёт распределение ошибки, которое является несмещённым для задачи этого диплома. Если рассмотреть к чему стремится среднее значение максимума получается число близкое к 0.9957, что совершенно не равно 0.99001.

Результаты сравнения базовых алгоритмов: 1.

2.2. Линейная регрессия

Обозначение: $linear\_regression$

Данный способ заключается в аппроксимации хвоста распределения полиномом и использованием построенной модели для получения квантиля, который лежит за максимумом.

Для экспериментов была использована стандартная линейная регрессия из пакета R lqmix с функцией $c_0 + c_1x + c_2x^2 + c_3x^4 + c_5x^8$. Степени были взяты, как степени двойки, так как для распределений с тяжёлыми хвостами

необходимы очень большие степени. Для обучения алгоритм получал 10% максимальных элементов выборки.

Положительные свойства данного подхода это намного более реалистичные ответы, чем метод максимума. Так же он не сильно медленнее обычного максимума, особенно если сравнивать со следующей группой алгоритмов. Но главная его проблема это качество ответов в целом, хоть алгоритм среднее алгоритма по значениям близко к квантилю, но на практике он имеет большое стандартное отклонение, которое существенно мешает среднему после перевода в проценты. Для модификаций алгоритмов было принято решение сосредоточиться на более продвинутых алгоритмах.
Результаты сравнения базовых алгоритмов: 1.

2.3. Квантили из теории экстремальных значений

Обозначение: gpd_****

Это способы основанные на обобщённом распределении Парето.
А именно из выборки выбирается часть, по которой строят распределение хвоста аппроксимируя подвыборку обобщённым распределением Парето.
Есть два популярных способа находить такую подвыборку это:

- gpd_pot – берут некоторый процент самых больших элементов.
- gpd_mle – выборка разбивается на примерно равные части и из этих частей берут некоторый процент самых больших элементов, таким образом в выборку попадают и не самые большие элементы.

К несчастью, как и в других сферах, в задаче этого диплома оба разбиения имеют распределения с которыми они справляются лучше, а значит будут рассматриваться оба подхода.
Для gpd_mle использовалась библиотека extRemes, так как не удалось найти других реализаций.
Для gpd_pot использовалась та же библиотека, но было проведено сравнение с конкурентными библиотеками fExtremes и evir, и не было замечено существенной разницы.

Для нахождения оптимальной границы для отсечения подвыборки была использована библиотека threshr.

Результаты сравнения базовых алгоритмов: 1.

2.4. Таблица сравнения базовых алгоритмов

sd %	max	linear regression	gpd pot	gpd mle
lnorm	0.0097527	0.0091269	0.0062582	0.0079027
ping	0.0099017	0.0093008	0.0073583	0.0073479
file	0.0097501	0.009163	0.011059	0.0081788
student	0.0097347	0.00893	0.0063632	0.0074607
norm	0.010249	0.0099401	0.0082819	0.0067571
pca	0.0095787	0.0089117	0.0048133	0.0071046
sort	0.009749	0.0087874	0.0073713	0.0083644
cauchy	0.0097924	0.0085929	0.0059423	0.007965
mean val	max	linear regression	gpd pot	gpd mle
lnorm	0.99587	0.99817	0.99854	0.99927
ping	0.99539	0.9983	0.99775	0.99898
file	0.99966	0.99985	0.99796	0.99869
student	0.99768	0.99903	0.99867	0.99894
norm	0.99365	0.99809	0.99943	0.99864
pca	0.99533	0.99643	0.99592	0.99976
sort	0.9961	0.99729	0.99709	0.99727
cauchy	0.99845	0.99921	0.99754	0.99879

Таблица 1: Сравнение результатов.
sd % – отвечает за среднеквадратическое отклонение предсказаний, цель 0.
mean val – отвечает за среднее по предсказаниям с переводом в %, цель 0.999.

3. Модификации gpd

3.1. Среднее на полной выборке

Обозначение: gpd_mean

Так как оба способа показывают хорошие результаты на разных данных, то было решено рассмотреть сумму. Для этой эвристики оба метода были запущены независимо на полной выборке. После чего их результаты были сложены с равным весом $\frac{1}{2}$.

Этот способ показал потенциал смешения двух уже хороших алгоритмов. Во-первых, он работал на достойном уровне на всех данных, так как позволял быть всегда лучше, чем один из внутренних алгоритмов.
Во-вторых, благодаря смешению разных ошибок уменьшается стандартное отклонение конечного предсказания. В текущей модели уменьшение отклонение сопровождается улучшением процентного среднего.
Но разумеется из-за необходимости аппроксимировать 2 раза скорость алгоритма заметно снизилась.
Сравнительные результаты можно посмотреть в 2 таблице.

3.2. Среднее на половинках выборок

Обозначение: gpd_half_mean

Данный способ был вдохновлён gpd_mean. Но главным изменением в этот раз стало использование каждым внутренними алгоритмами своей половины выборки. Такое разделение исключает пересечение между подвыборками для обучения двух алгоритмов, а значит делает их ответы более независимыми. В то же время увеличивая стандартное отклонение из-за уменьшения выборки для обучения.

Тем не менее способ показал насколько независимость выборок улучшает результат среднего. Так как стандартное отклонение среднего на половинах выборок дало ещё более маленькую дисперсию чем просто среднее и даже сравнимую с минимальной дисперсией базовых алгоритмов на полной выборке.
К несчастью уменьшение выборок для обучения дало и отрицательное влия-

ние, которое выразилось в менее точном среднем на значениях. И, разумеется, время работы относительно базовых gpd тоже увеличилось, ведь необходимо проводить 2 аппроксимации.
Сравнительные результаты можно посмотреть в 2 таблице.

3.3. Взвешенное среднее

Обозначение: gpd_w_mean

Идея заключается во взятии среднего с коэффициентами, которые получаются путём использования среднего на половинках выборок. В теории такой подход должен смешивать ответы двух алгоритмов в пропорции более правильной чем просто среднее.

Во время тестирования алгоритм был в числе лучших и проявил себя довольно сбалансировано. Для некоторых наборов данных даже имея наилучшие результаты по среднему на значениях.
Сравнительные результаты можно посмотреть в 2 таблице.

3.4. Медиана полных выборок и среднего на половинках

Обозначение: gpd_median

В данном алгоритме вычисляются 3 статистики gpd_pot, gpd_mle и gpd_hulf_mean. После чего из этих 3 статистик берётся медианная. Данная стратегия призвана уменьшить стандартное отклонение и увеличить количество случаев с допустимым значением метрики.

Медиана показывает очень хорошие результаты по стандартному отклонению на распределении Стьюдента, и в целом имеет хорошие показатели на многих наборах данных.
К минусам можно отнести плохие показатели по среднему квантилю, которые находятся на уровне gpd_pot. И так как для вычисления нужно было произвести аппроксимацию 4 раза скорость данного алгоритма такая же низкая как у gpd_w_mean.
Сравнительные результаты можно посмотреть в 2 таблице.

3.5. Выбор по среднему на половинках

Обозначение: gpd_pick

Алгоритм очень похож на взвешенное среднее. Он так же опирается на разность двух подходов gpd_pot, gpd_mle. Но вместо того, чтоб из двух ответов брать сумму, он доводит данную идею до максимума используя только 1 из двух алгоритмов. Получается, что gpd_half_mean выбирает какой из двух алгоритмов прав.

Положительной чертой данного алгоритма является более стабильное уменьшение стандартного отклонения особенно в случае с набором данных $file$.

Минусом являются самые плохие показатели по среднему значению среди представленных модификаций. Так же к минусам можно отнести время работы, которое аналогично gpd_w_mean и gpd_median.

Сравнительные результаты можно посмотреть в 2 таблице.

3.6. Таблица сравнения комбинированных методов

sd %	gpd_mean	gpd_half_mean	gpd_w_mean	gpd_median	gpd_pick
lnorm	0.0066654	0.0061883	0.0068742	0.0065724	0.0068454
ping	0.0069868	0.0067659	0.0070336	0.0068088	0.0071537
file	0.0087831	0.0083989	0.0091135	0.008829	0.0078969
student	0.0066101	0.0064363	0.006124	0.0059616	0.0061542
norm	0.0066709	0.006218	0.0064795	0.0069576	0.0069913
pca	0.0054851	0.0068033	0.005375	0.0055614	0.00555
sort	0.0072076	0.0054769	0.0072734	0.007306	0.0074512
cauchy	0.0065625	0.0065827	0.0065287	0.0065123	0.0065454
mean val	gpd_mean	gpd_half_mean	gpd_w_mean	gpd_median	gpd_pick
lnorm	0.99915	0.99859	0.9985	0.99846	0.99848
ping	0.99905	0.99819	0.99765	0.9977	0.99759
file	0.99784	0.99759	0.99794	0.9979	0.99805
student	0.99933	0.99888	0.99802	0.99803	0.99799
norm	0.99918	0.99962	0.99925	0.99932	0.99933
pca	0.99969	0.99676	0.99591	0.99591	0.99591
sort	0.99721	0.99601	0.99708	0.99708	0.99706
cauchy	0.99848	1	0.99772	0.99775	0.99786

Таблица 2: Сравнение результатов.
sd % – отвечает за среднеквадратическое отклонение предсказаний, цель 0.
mean val – отвечает за среднее по предсказаниям с переводом в %, цель 0.999.

4. Выравнивание ошибки

4.1. Выравнивание ошибки

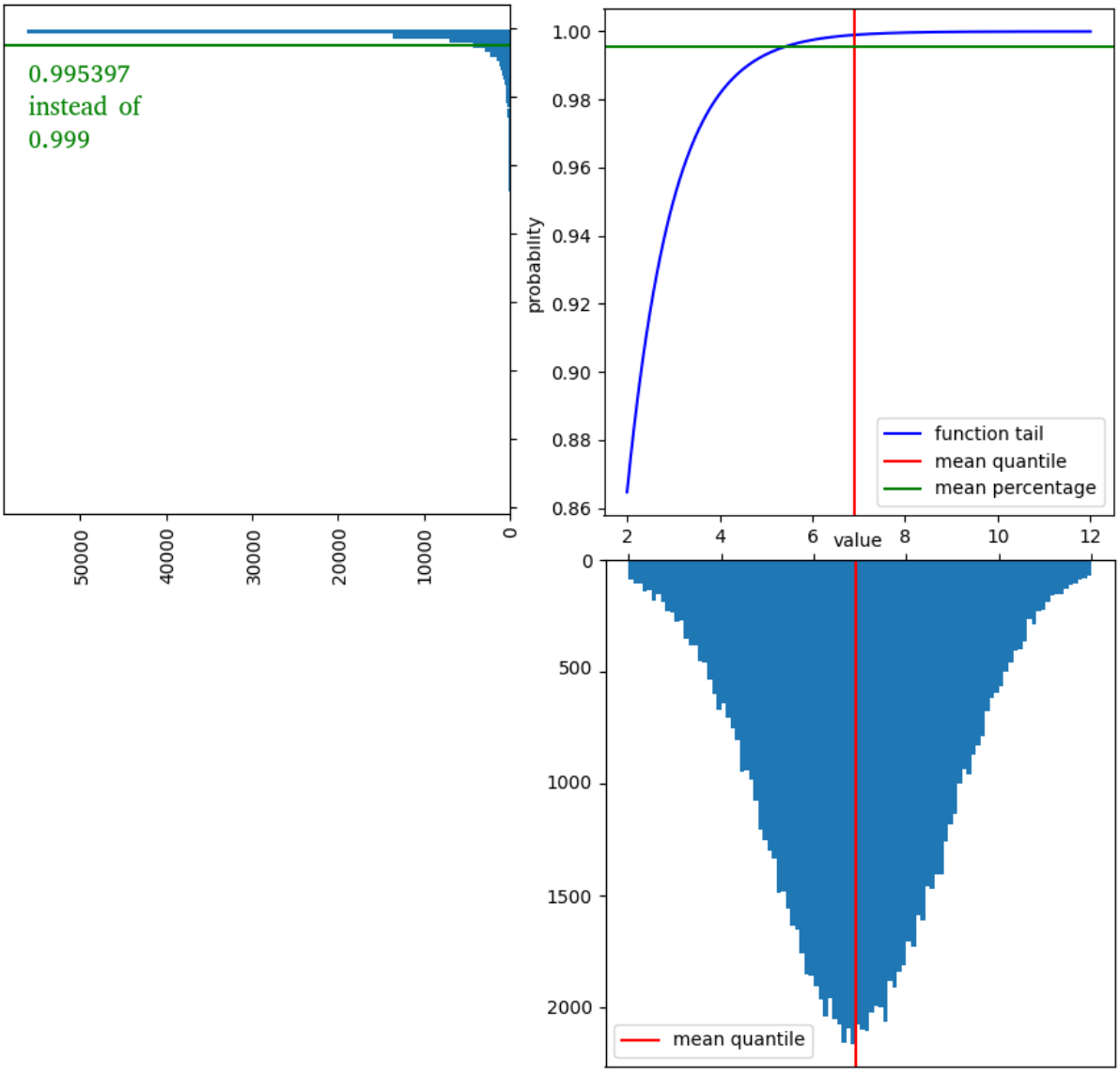
Метрика работы:

$$\mathbf{E}_X X^{-1}(F(X_1, ..., X_n, p)) \geq 1 - p$$

Метрика gpd квантилей:

$$\mathbf{E}_X F(X_1, ..., X_n, p) = quantile(X, 1 - p)$$

Как это выглядит:



Для учёта данной разницы необходимо знать распределение X, а так

же распределение ошибки, назовём распределение ошибки $D(F, p)$.
Тогда идеальным ответом было бы преобразование ответа из $F(X_1, ..., X_n, p)$ в $F(x_1, ..., x_n, p) + f(X, D(F, p))$, где f – функция дающая поправочный коэффициент для предсказания, такой, что математическое ожидание после преобразования в проценты давало бы целевые 0.999
Для поиска этой константы было аппроксимировано $D(F, p)$, как нормальное распределение с отклонением = $quantile(X, 0.999) - quantile(X, 0.985)$.
Так как у алгоритма нет доступа к распределению X, то необходимо использовать его аппроксимации. Благо gpd и так приближает хвост распределения.
Теперь, зная распределение ответов в процентном пространстве, достаточно выбрать идеальный квантиль, который после преобразования будет давать в среднем 0.999. Этой задачей занимается бинарный поиск минимизирующий расстояние среднего на процентах до 0.999 квантиля с границами $[quantile(X, 0.99), quantile(X, 0.9999)]$.

4.2. Таблица сравнения полученных алгоритмов

mean %	gpd_pot_fix	gpd_mle_fix	gpd_mean_fix	gpd_half_fix	gpd_w_fix	gpd_median_fix	gpd_pick_fix
lnorm	0.99778	0.99616	0.99733	0.99749	0.99728	0.99778	0.99647
ping	0.9953	0.9947	0.99527	0.9956	0.99516	0.99538	0.99466
file	0.99333	0.99412	0.99429	0.99506	0.99425	0.99345	0.99398
student	0.99768	0.99588	0.99734	0.99752	0.99763	0.99796	0.99632
norm	0.99662	0.99779	0.99761	0.99835	0.99737	0.9964	0.99714
pca	0.99716	0.99645	0.997	0.99668	0.99697	0.99724	0.99661
sort	0.99634	0.99526	0.99623	0.99631	0.99616	0.99635	0.99547
cauchy	0.99669	0.9953	0.99636	0.99613	0.99638	0.99672	0.99557
sd %	gpd_pot_fix	gpd_mle_fix	gpd_mean_fix	gpd_half_fix	gpd_w_fix	gpd_median_fix	gpd_pick_fix
lnorm	0.0048789	0.0062962	0.0050005	0.0045699	0.0051762	0.0048789	0.0060435
ping	0.0071283	0.0067224	0.0063582	0.0056532	0.0067003	0.0067603	0.0070554
file	0.011057	0.0072944	0.0080729	0.007591	0.008221	0.010998	0.0087425
student	0.0047848	0.0057548	0.004756	0.0044399	0.0046172	0.0047568	0.0056577
norm	0.0073911	0.005508	0.0056856	0.0047318	0.0059064	0.0072694	0.0063634
pca	0.0037603	0.0055071	0.0040481	0.0038968	0.0042814	0.0034337	0.0051164
sort	0.0062192	0.0063176	0.0057491	0.0056423	0.0058575	0.0063244	0.0065683
cauchy	0.0045241	0.006707	0.0050794	0.0048454	0.0049777	0.0044675	0.0064412
mean val	gpd_pot_fix	gpd_mle_fix	gpd_mean_fix	gpd_half_fix	gpd_w_fix	gpd_median_fix	gpd_pick_fix
lnorm	2001.1	3742	2554.7	3203.5	1948.3	2007.7	2458.5
ping	2926.9	3973.2	5206.5	2848.3	2856.6	2894.1	5573.7
file	12543	13231	13153	12481	12406	12546	14513
student	641.36	682.31	744.81	639.57	636.84	640.05	1167.7
norm	7.5055	6.3462	6.9036	8.1678	6.9453	7.4692	6.6971
pca	90781	115030	121970	89005	90807	91251	118510
sort	68625	76642	76642	71985	67934	68604	72603
cauchy	459.95	983.3	534.63	500.36	454.7	450.39	972.89

Таблица 3: Сравнение результатов.
mean % – отвечает за метрику, цель 0.999.
sd % – отвечает за среднеквадратическое отклонение предсказаний, цель 0.

4.3. Выбор оптимального алгоритма

По итогам работы было написано 7 алгоритмов достойных финального рассмотрения. Сравнительные результаты можно посмотреть в 3. Проведём попарное сравнение для того, чтоб убрать явно отстающие алгоритмы.

- gpd_pick

  Сравнение будет проводиться с gpd_w_fix алгоритмом. gpd_pick имеет меньший показатель по среднему после перевода в проценты на всех

наборах данных. Он так же всегда имеет большее стандартное отклонение. С минимизацией среднего по значению для нормального распределения действительно алгоритм действительно справился лучше, чем gpd_w_fix, но стоит заметить, что, во-первых, совсем немного, а, во-вторых, он имеет и меньшее среднее после приведения в проценты, что является более приоритетным для данной задачи.

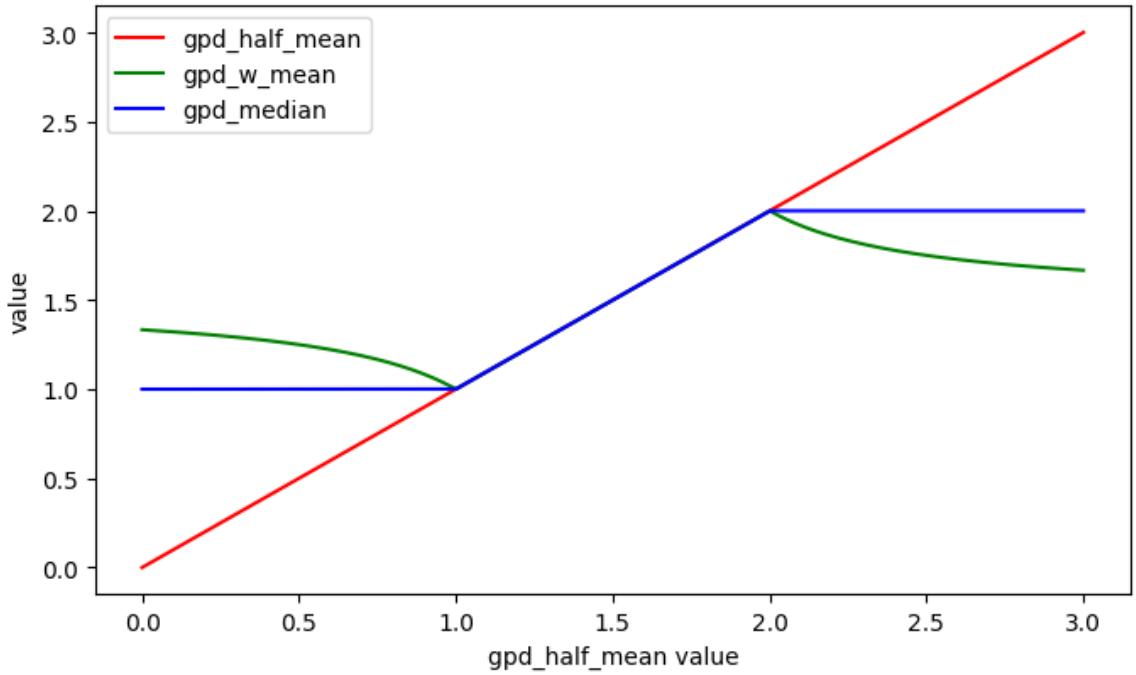
- gpd_mle
  Сравнение будет проводиться с gpd_half_fix алгоритмом. Все значения за исключением двух меньше у gpd_mle. Первое хорошее значение это отклонение в наборе данных file, что не помогло gpd_mle получить хороших результатов, которые реально являются метриками. Второе значение это действительно достойное поведение на наборе данных нормального распределения и совершенно небольшая разница в средних после переведения в проценты вполне себе покрывается разницей в средних значениях, тем не менее на остальных наборах данных gpd_mle слишком сильно отстаёт от своей производной.

- gpd_pot
  Сравнение будет проводиться с gpd_median_fix алгоритмом. Алгоритмы ведут себя очень похоже, но всё-таки почти везде и чуть-чуть gpd_median_fix лучше.

- gpd_mean
  Сравнение будет проводиться с gpd_w_fix алгоритмом. К несчастью сильно разные алгоритмы кончились, поэтому тут речь будет идти о сходстве ответов. Значения mean % очень схожи, как и sd %, но со средним значением gpd_mean является намного более слабым алгоритмом. Единственное среднее значений в котором он чуть-чуть лучше это нормальное распределение, но и там разница довольно не велика и разница в средних значениях во всех остальных наборах данных сильно перекрывает этот маленький успех.

Рассмотрим оставшиеся 3 алгоритма и чем они отличаются. Все 3 алгоритма ведут себя идентично, если $median(gpd\_half\_fix, gpd\_pot, gpd\_mle) =$

gpd_half_fix, но если значение gpd_half_fix выходит за рамки остальных алгоритмов поведение меняется. Рассмотрим графики, где значения $gpd\_pot, gpd\_mle = 1, 2$, порядок для всех алгоритмов не важен.



Из чего можно сделать вывод, что gpd_half_fix наиболее экстремальный, это можно видеть по средним значениям на распределениях lnorm и norm. У gpd_median_fix тоже есть проблемы в виде набора данных file и распределения norm. В это время gpd_w_fix всегда достаточно близок к наилучшему ответу в выборке по статистике среднего после перевода в проценты и почти всегда лучший или близок к лучшему по среднему значению. Соответственно из рассмотренных алгоритмов стабильным алгоритмом с очень хорошими метриками получается исправленное взвешенное среднее.

4.4. Советы по применению

Во время изучения было обнаружено, что если есть информация, что данные в тесте будут меняться не сильно, допустим не будет изменений в 2 раза, то тогда рекомендуется использовать gpd_w_mean с очень сильно уменьшенным отклонением. Уменьшить отклонение в этом случае можно используя систему момента, а именно значение временного ограничения на текущем шаге = значению квантиля $\cdot 0.01$ + значение временного ограничения на предыдущем шаге $\cdot 0.99$. Эта схема способна переживать небольшие

изменения в скорости.

К несчастью даже алгоритмы приведённые в данной работе рассчитаны на качественное предсказание лишь не слишком экстремальных квантилей, поэтому увеличение выборки так же помогает в получении лучшего качества квантилей. Замечу, что для достаточно высоких квантилей изменение среднего времени теста на несколько процентов всё ещё является несущественным и это позволяет увеличить размер выборки.

Так же ещё одним ключевым изменением, которое можно использовать, это добавить в задачу знание о среднем отклонении предсказаний, что может быть использовано для более качественного перевода предсказания из значений квантилей в проценты распределения.

И, разумеется, можно использовать это ограничение по времени, как сигнал к запуску этого теста несколько раз для более тщательного исследования. Таким образом уменьшится и частота падений без причин и временное ограничение на тест.

Заключение

По итогам работы был разработан алгоритм превосходящий базовые алгоритмы, но всё ещё не являющийся идеальным. При размерах выборки в 100 элементов и целевом математическом ожидании вероятности падения теста на фиксированном распределении равном 0.999, достичь удалось $0.996-0.997$. При этом значения временных ограничений, полученные при помощи алгоритма, имеют низкие стандартные отклонения, что добавляет программе предсказуемости.

Список литературы

[1] Bernhard Pfaff [aut, cre], Eric Zivot [ctb], Alexander McNeil [aut] (S original (EVIS)), Alec Stephenson [trl] (R port of EVIS). Extreme Values in R. 2018.

[2] Diethelm Wuertz [aut], Tobias Setz [aut], Yohan Chalabi [aut], Paul J. Northrop [cre, ctb]. Rmetrics - Modelling Extreme Events in Finance. 2022.

[3] Eric Gilleland. Extreme Value Analysis. 2022.

[4] Berry Boessenkool. Extreme Value Statistics and Quantile Estimation. 2023.

[5] Paul J. Northrop [aut, cre, cph], Nicolas Attalides [aut]. Threshold Selection and Uncertainty for Extreme Value Analysis. 2023.

[6] Maria Francesca Marino [aut, cre], Marco Alfo' [aut], Nicola Salvati [aut], Maria Giovanna Ranalli [aut]. Linear Quantile Mixture Modelss. 2023.

[7] Paul J. Northrop, Nicolas Attalides, Philip Jonathan. Cross-Validatory Extreme Value Threshold Selection and Uncertainty with Application to Ocean Storm Severity. 2017.