DOI 10.37882/2223–2966.2022.04–2.33

ТЕХНИЧЕСКИЙ АНАЛИЗ КОДА БИБЛИОТЕКИ «STATISTICS» ЯЗЫКА ПРОГРАММИРОВАНИЯ NYQUIST, В ЦЕЛЯХ ПОВЫШЕНИЯ ТОЧНОСТИ И ЭФФЕКТИВНОСТИ ОБРАБАТЫВАЕМОГО АУДИОМАТЕРИАЛА

Таран Василий Васильевич
К.культурологии, заведующий Лабораторией компьютерного дизайна и прикладной информатики «SPLASH»; ФГБУН «Всероссийский институт научной и технической информации РАН»
allscience@lenta.ru

Аннотация. Статья посвящена общему научному анализу инженерно-технических возможностей библиотеки «Statistics» языка программирования Nyquist. Произведённый в статье анализ библиотеки «Statistics» раскрывает сущность изложенного авторами библиотеки подхода к статистической обработке аудиоданных с помощью языка Nyquist и его штатного интерфейса прикладного программирования — NyquistIDE. В статье фрагментарно проясняются некоторые особые моменты использования полезных функций и специализированных операторов языка Nyquist в связке с инженерно-компьютерной практикой прикладного редактирования аудиоматериала. В качестве элемента статистического анализа аудиоданных обосновывается применение прямого (нестабильного) алгоритма коэффициента корреляции Пирсона, теста Уэлча и критериев Левена-Брауна-Форсайта. Проблематика статьи также затрагивает вопросы ультратонкого редактирования аудиоматериала с использованием статистических методов в программе Audacity® через приглашение языка программирования Nyquist, позволяющего учитывать широкий спектр акустических нюансов во избежание ошибок, возникающих при конвертации, очистке, сведении, мастеринге и ремастеринге аудиоматериала, а также при прочих инженерно-технических практиках.

Ключевые слова: библиотека «Statistics», язык программирования Nyquist, NyquistIDE, элементы статистики в обработке аудиоданных, математическая статистика, коэффициент корреляция Пирсона, тест Уэлча, критерии Левена-Брауна-Форсайта.

TECHNICAL ANALYSIS OF THE «STATISTICS» LIBRARY CODE WITHIN THE NYQUIST PROGRAMMING LANGUAGE IN ORDER TO IMPROVE THE ACCURACY AND EFFICIENCY OF THE PROCESSED AUDIO MATERIAL

V. Taran

Summary. The article deals with general scientific analysis of engineering and technical potential of the «Statistics» library associated with Nyquist programming language. The «Statistics» library analysis performed in the article reveals the essence of the approach to statistical processing of audio data presented by the authors of the library using the Nyquist language and its standard application programming interface — NyquistIDE. Some special aspects of the useful functions and specialized operators of the Nyquist language usage bundled with the engineering and computer practice of applied audio editing are clarified separately in the article. The application of the direct (unstable) algorithm for Pearson correlation coefficient, Welch test, Levene-Brown-Forsythe tests as the component of audio data statistical analysis is reasoned. The article also addresses the issues of ultra-fine editing of audio material using statistical methods in the Audacity® program. This is ensured by the invocation of the Nyquist programming language, which allows taking into account a wide range of acoustic nuances in order to avoid errors that occur during conversion, cleaning, mixing, mastering and remastering of audio material and also other engineering and technical practices.

Keywords: «Statistics» library, Nyquist programming language, NyquistIDE, statistical components in audio data processing, mathematical statistics, Pearson correlation coefficient, Welch test, Levene-Brown-Forsythe tests.

При проведении процедур, связанных с компьютерной обработкой аудиоматериала, важным фактором является точность настройки всех модулей, предусмотренных программным обеспечением обработки звука, что в свою очередь заметно сказывается на качестве вывода уже обработанного (финализированного)[1] аудиоматериала. Для профессионального звукоинженера или программиста, работающего в сфере компьютерной аудиоиженерии, не является секретом что *точность* и последующая *эффективность* обработки аудиосигнала во многом зависят от *статистических данных* обрабатываемой алгоритмом аудиоформы. К статистическим данным в условиях обработки аудиоматериала относится следующее: учёт колебания амплитуды аудиосигнала (для определения плотности или разреженности частоты сокращений сонограммы), представление аудиосигнала в виде математических значений в соответствии со спектром частот (нижние, верхние и средние частоты 5905 Гц = — 42,4 дБ), построение гистограмм, корреляционный анализ и прочие тестовые функции. Библиотека «Statistics»[2] является частью общей архитектуры языка программирования Nyquist (далее по тексту — язык Nyquist), который в свою очередь тесно связан со свободно распространяемым программным обеспечением в области обработки аудиоматериала Audacity®[1,2,3].

Взаимосвязь с аудиоредактором Audacity® позволяет производить визуализацию вычислительных процессов, видоизменять первоначальную структуру аудиоматериала, вести текстово-графический учёт обрабатываемого аудиоматериала. Все перечисленные преимущества определяют высокую степень актуальности рассматриваемых нами вопросов, касающихся совершенствования прикладных практик обработки аудиоматериала, направленных, прежде всего, на повышение эффективности технико-технологических операций по коррекции *больших* и *малых* массивов аудиоданных. Важно отметить, что совмещение интерфейсно-ориентированных функций программы обработки звука (в нашем случае — Audacity®) с автономной средой прикладного программирования (NyquistIDE) помогает разрабатывать принципиально новые программно-зависимые[3] технические инструменты, позволяющие оператору технологического процесса решать конкретно поставленные задачи. В аудиоредакторе Audacity® имеется приглашение языка Nyquist (NyquistPrompt), которое позволяет фрагментарно тестировать разработанный программный код, варьировать введённые выражения непосредственно в окне программы[4]. Приглашение поддерживает ввод данных с *переменным синтаксисом* и даёт возможность запустить программный код в разных вариациях синтаксических конструкций[5,6,7,8]. Расширенные выражения на LISP дают возможность символьных вычислений, что в свою очередь может способствовать проведению общего музыкального анализа и статистического анализа аудиоматериала[9,10]. Библиотека «Statistics», как один из основных компонентов языка программирования Nyquist, предполагает межпрограммную настройку, что существенно сокращает время на сбор статистики и расширяет возможности для проведения математических манипуляций (рис. 2).

Язык программирования Nyquist имеет собственную автономную интегрируемую среду разработки микропрограмм NyquistIDE. Nyquist позиционируется как специальный (профильный) язык для аудиосинтеза и композиций. Следует отметить, что Nyquist также хорошо реализован в виде интегрированной оболочки, реализуемой через программный интерфейс в редакторе Audacity®. Адаптация данного языка программирования под программную среду Audacity® позволяет управлять процессами обработки звука в аудиоредакторе на программном уровне, минуя интерфейсно-ориентированную систему. Важным фактором по качественному управлению обработкой звука, а также по его синтезу и составлению новых музыкальных аудио партий, выступает довольно обширный функционал языка Nyquist, который базируется на интеллектуальных возможностях LISP[9]. Прямая взаимосвязь с LISP позволяет распределять и редактировать аудиоданные в соответствии с современными представлениями об искусственном интеллекте.

[1] **Прим. автора.** (По определению автора — Таран В.В., 2022) Финализация аудиоматериала — это технологический процесс, задействующий определённый арсенал программно-технических средств, направленных на улучшение звучания предварительно сведённых аудиосесиий (имеющих отличающиеся между собой технико-акустические характеристики) в единую звуковую форму, с последующим её воспроизведением на акустическом оборудовании, использующем различные системы возбуждения аудиосигнала (колонки, сабвуферы, репродукторы и т.п.).

[2] **Прим. автора.** По тексту статьи название библиотеки используется в кавычках для сохранения официального англоязычного названия, поскольку некоторые программные элементы NyquistIDE (линейная алгебра, векторная математика и т.д.) могут быть ассоциированы с расширением «statistics.lsp» либо в определённых местах быть частью этого расширения. Допустимо также использовать и русскоязычное наименование — «Статистика».

[3] **Прим. автора.** Речь идёт о создании Nyquist-плагинов и в некоторых случаях периферийных Nyquist-программ, которые будут ресурсно-зависимыми от среды исполнения (Audacity®) но в то же время будут иметь возможность автономного функционирования в NyquistIDE, что позволит оператору (программисту) тщательно отслеживать изменения в событиях программы и в реальном времени вносить важные коррективы в программный код с последующим автоматическим перезапуском плагина. Таким образом, к примеру, обычная задача по общей компрессии звукового образца может быть на ходу перестроена и это касается, прежде всего, изменения типа компрессии, лимитирования пороговых значений компрессии и т.п. В добавлении ко всему ранее перечисленному, задача может приобрести межаспектные черты, когда компрессия и, к примеру, экспондирование звуковой дорожки, могут быть объединены в один плагин с заранее предустановленными техническими характеристиками.

Примечание: автор логотипа — Таран Василий Васильевич. При использовании логотипа в периодической печати, в электронных документах, а также для нужд идентификации данной библиотеки — ссылка на автора логотипа обязательна. Логотип характеризует фирменный шрифт с применением летеринга и частичной транслитерации «Statistics» — Nyquist Statistica.

Рис. 1. Предлагаемый логотип для идентификации библиотеки «Statistics»

```
(send statistics-class :answer :point '(x)
```

Код 1

Язык Nyquist в сочетании с аудиоредактором Audacity® открывает большие возможности по обработке, созданию и реставрации аудиоматериала, позволяет гибко и точно настраивать все необходимые при обработке аудиоматериала штатные модули[1] Audacity®. Одним из таких модулей, имеющих прямое отношение к статистическому анализу звуковой формы, является «Частотный анализ».

Он спроектирован в целях осуществления контроля над аудиочастотами, которые коррелируются с децибелами. Если говорить о прикладном аспекте обработки аудиоматериала, то эта функция очень важна, например, при реставрации аудиоматериала, когда полезный аудиоматериал насыщен нежелательными акустическими артефактами. Такие артефакты часто могут встречаться при оцифровке аудиоматериала со старых аналоговых носителей и при некорректном аудиомонтаже. В качестве примера подобных артефактов могут служить «пики»[2], нарушающие целостность аудиоматериала. Пики проявляются как кратковременные акустические импульсы и отражаются на представлении аудиоформы. Обычно, это резкое уплотнение амплитуды аудиосигнала с последующим интервалом, выражающееся в отклонении текущего проигрываемого фрагмента от общей аудиоформы (статистически это выглядит так: частота 1312 Гц при средней амплитуде аудиосигнала 36 дБ., пиком будет считаться значение, превышающее 36 дБ — 42,48,54 дБ). Разумеется, в таких ситуациях сбор статистических данных становится первопричиной. От его качества зависит, насколько точно будет реставрирован, оцифрован или форматно преобразован аудиоматериал.

Библиотека «Statistics» позволяет задействовать скрытый потенциал традиционного частотного анализа и увеличить количество функций, необходимых при обработке сложных структур аудиоданных.

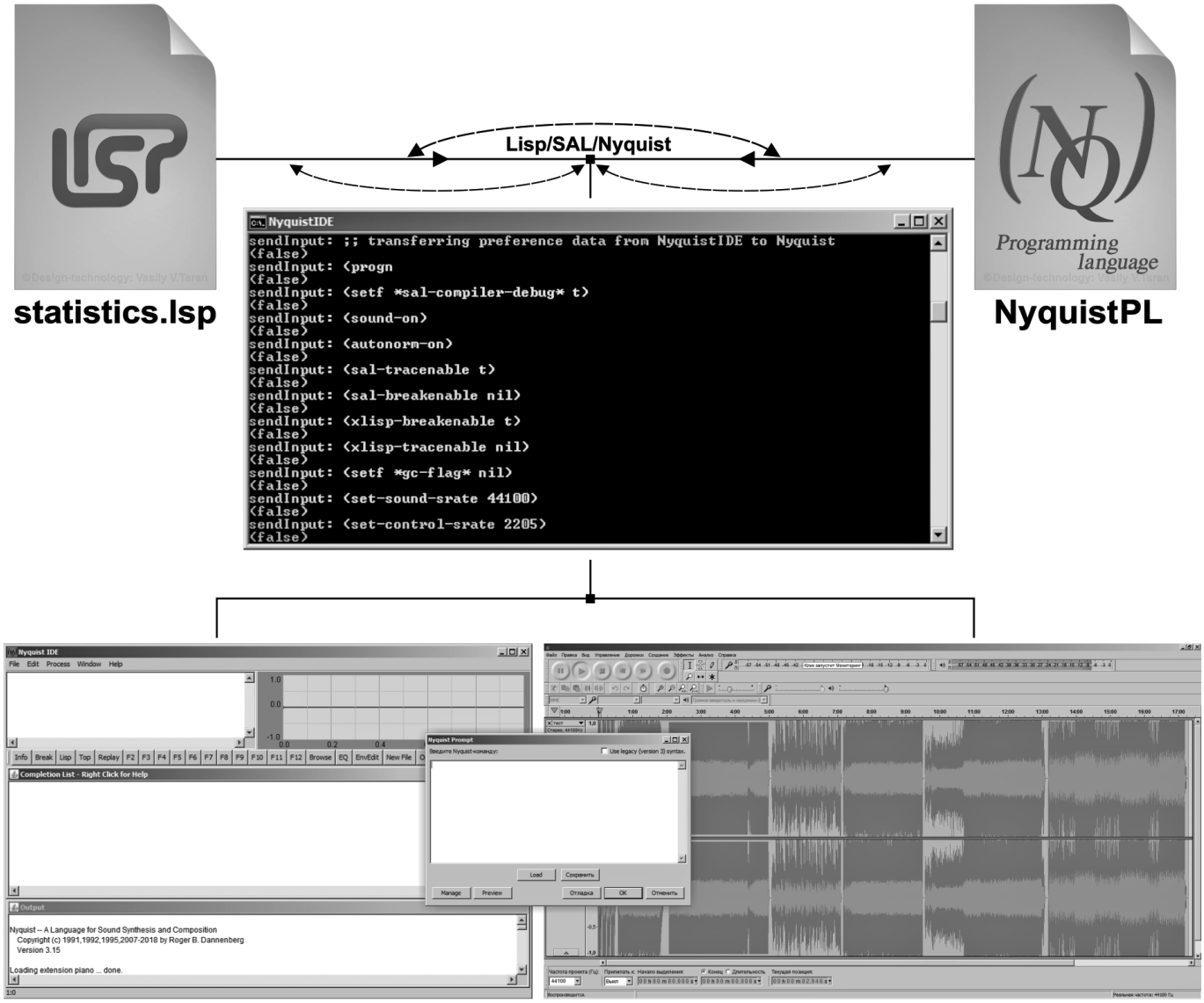
Библиотека «Statistics» в среде NyquistIDE имеет расширение statistics.lsp, это объясняется тем, что язык Nyquist и его одноимённая интегрируемая среда разработки NyquistIDE корректно воспринимают синтаксис двух языков LISP[3]/SAL. Библиотека работает с массивами данных и представляет их как объекты. К примеру, строчка, приведённая ниже, выполняет отправку «класса» статистки и получает ответ от точки x (код 1).

Для того чтобы подробнее разобраться в работе программного кода анализируемой библиотеки, приступим

[1] **Прим. автора.** Имеются ввиду плагины, входящие в комплект дистрибутива Audacity®.

[2] **Прим. автора.** Пики — один самых распространённых акустических артефактов. Бывают разных видов (в зависимости от происхождения). Видовые характеристики определяются положением пика в структуре аудиоматериала, его акустическими свойствами и импульсивностью. Когда речь идёт о нарушении целостности аудиоматериала, то чаще всего предполагается наличие дуплексных пиков, возникающих при некорректном горизонтальном сведении аудиоматериала либо включении в аудиодорожку дополнительных источников звука (микрофоны и т.д.), способных создать кратковременную электро-магнитную помеху, которая на спектрограмме будет отпечатана также как пиковое значение. Фрагмент аудиоматериала, который будет подвержен склейке, может иметь более высокую плотность, чем тот материал, с которым его необходимо склеить. В результате на узле стыковки двух аудиофрагментов появится пик.

[3] **Прим. автора.** Язык программирования LISP в статье пишется заглавными буквами, поскольку имеется ввиду оригинальный «LISP (1955 год — LISP 1, 1.5)» от которого следуют ветки типа XLISP и т.д. Поскольку XLISP наиболее родственная ветка для Nyquist, мы употребляем его в историческом смысле, с даты его официальной публикации (1983 год). Во всех остальных случаях удобнее пользоваться современным эквивалентом — Lisp.



Примечание: на представленном читателю рисунке* иллюстрируется возможность совместного использования библиотеки «Statistics» двумя программами — Audacity® и NyquistIDE. Такой подход значительно упрощает статистический анализ звукового файла и повышает эффективность отладки фрагментов кода библиотеки для нужд оператора. Поскольку Nyquist является LISP-подобным языком, а его структурная основа опирается на зависимость от алгоритмического языка SAL, то оператор вправе выбирать, в какой синтаксической форме ему удобнее писать код, вводить команды и определять выражения. Командная строка в данном случае будет зависеть от программной микросреды NyquistIDE, поэтому регистрация действий и событий будет производиться из окна NyquistIDE. В некоторых особых случаях, если оператор имеет глубокие знания программирования и составления лексических конструкций на языке LISP, а также владеет навыками компиляции исходных строчек кода программы Audacity®, возможно производить параллельную сессию регистрации событий с коррекцией выражений через NyquistPrompt.

* Автор рисунка — Таран Василий Васильевич. При использовании рисунка в периодической печати, в электронных документах — ссылка на автора и библиографические данные статьи — обязательна.

Рис. 2. Мультипрограммное использование библиотеки «Statistics» с регистрацией действий в приглашении командной строки

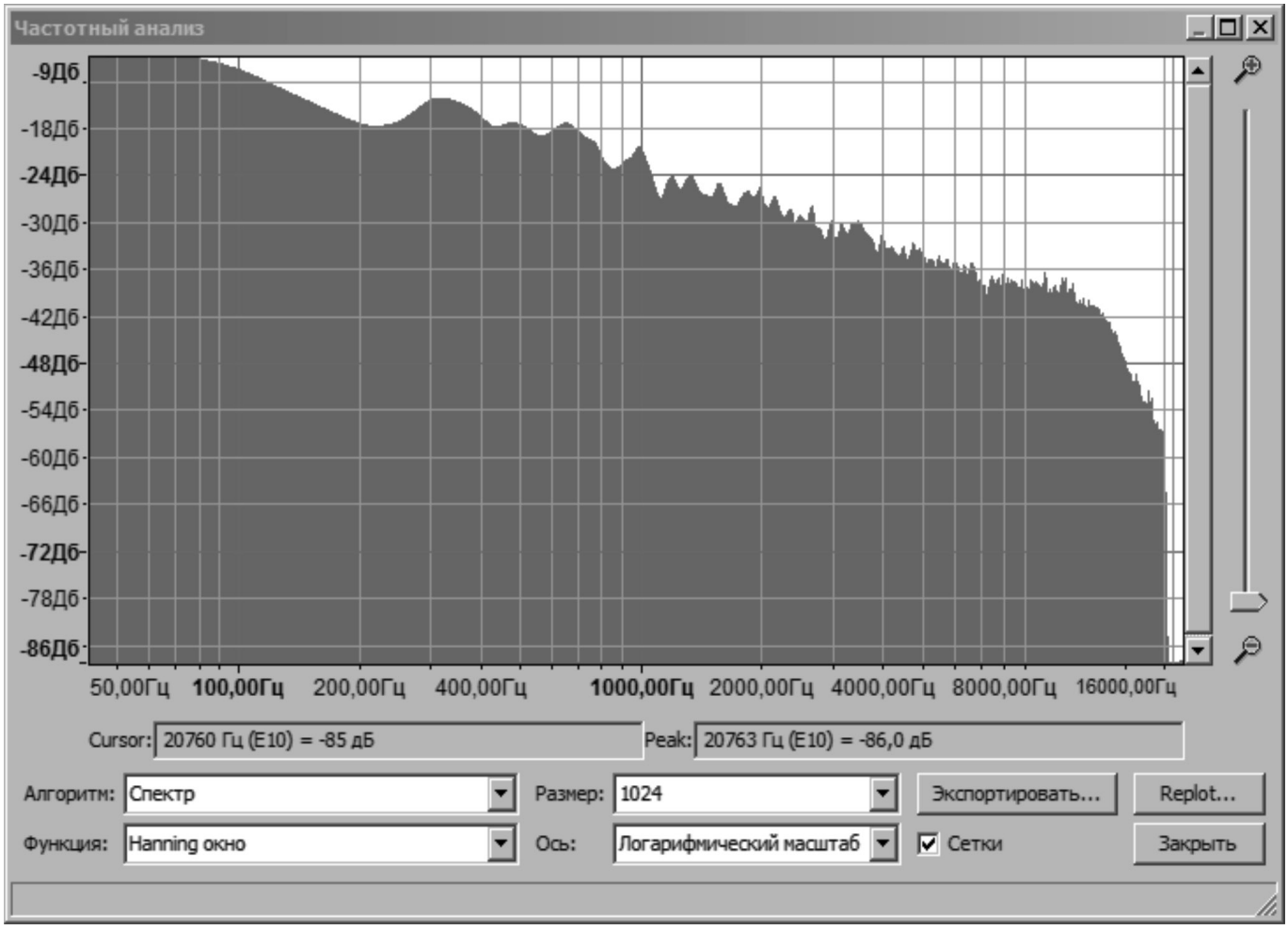


Рис. 3. Модуль обработки аудиосигнала — «Частотный анализ». На модуле отображён аудиосигнал высокой степени плотности при стандартных интерфейсно-опциональных возможностях программы Audacity®

```
;; отправка печати ключевых значений
;;     (require-from 'statistics "statistics.lsp")
(defun statistics () (print "See statistics.lsp"))
```

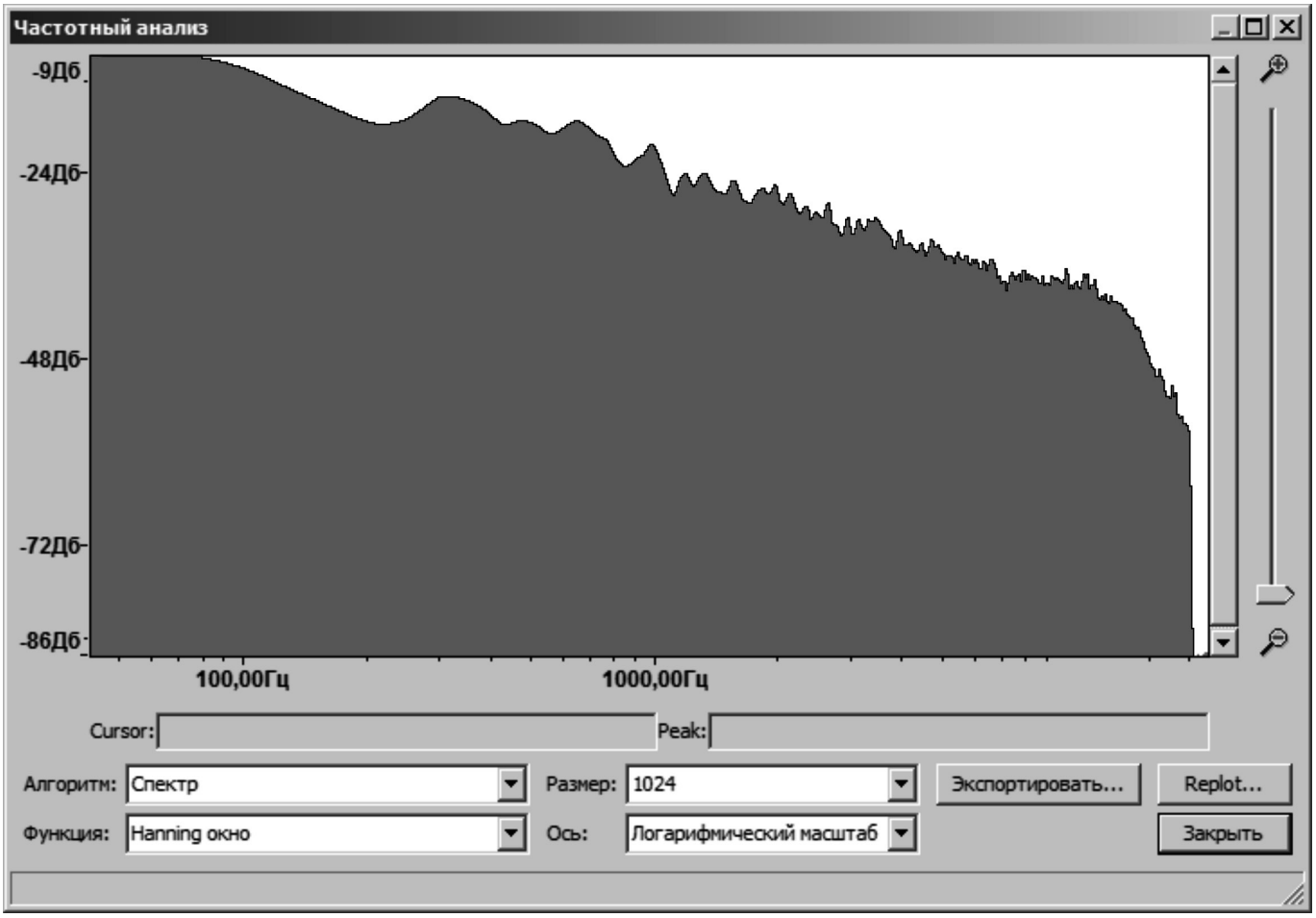
Код 2

к его техническому анализу. Данный анализ поможет нам уточнить некоторые технические нюансы при проведении манипуляций с аудиоматериалом, к тому же понимание некоторых выражений и синтаксических конструкций поможет в построении предметно-ориентированных массивов кода[1].

Итак, библиотека «Statistics» выполняет стандартные функции статистики, работает с гистограммами, обрабатывает числа, делает корреляционный анализ, выполняет листинг статистических значений аудиоформы и т.д. Как и в программном коде XLISP, который является веткой LISP[2], комментарии к коду записываются после ин-

[1] **Прим. автора.** Под предметно-ориентированными массивами кода понимается массив программного кода, отвечающий за конкретные процедуры при проведении манипуляций с аудиосигналом и с его мультидорожечными сессиями.

[2] **Прим. автора.** Стоит помнить, что LISP классификационно относится к языкам функционального программирования и характеризуется определением функций, функции также обеспечивают вызов других функций — отличается рекурсивностью (см. математическая теория рекурсивных функций), поэтому Nyquist в данном случае



Примечание: благодаря межпрограммному использованию библиотеки «Statistics», автор в качестве наглядного примера перекомпилировал модуль под свои нужды, изменив стандартное отображение осей и добавив обводную плавающую кривую для пиковых значений*.

* Прим. автора. Компиляция модуля выполнена согласно условиям лицензии GNU GENERAL PUBLIC LICENSE Version 2, June 1991.

Рис. 4. Модифицированный модуль обработки аудиосигнала — «Частотный анализ»

тервального значения (;;)[1]. Начальная функция требует от текущей статистики и её одноимённого расширения выполнить печать ключевых значений в следующий абзац (код 2).

Благодаря пользовательской функции require-from производится тестирование на возможность использования обращения к статистике повторно. Если повторное чтение невозможно, тогда каталог текущего файла используется в качестве основного пути[2]. Defun (символьная функция), служащая для определения функции, в данном случае определяет статистику для печати с соответствующим расширением.

Далее выполняется обращение к массиву векторов для обработки абстрактных векторных функций (код 3).

Здесь setf — это символьная функция, устанавливающая значение поля для идентификации впослед-

сохраняет преемственность декларативности стиля, направленного на функциональность.

[1] **Прим. автора.** В некоторых случаях отдаётся дань Common Lisp и при необходимости комментарии могут принимать следующий вид:

1) [;] — Комментарий, находящийся в конце строки.
2) [;;] — Вложенный комментарий.
3) [;;;] — Внешний комментарий.

[2] **Прим. автора.** Повторное обращение к статистике бывает необходимо при задачах многопоточной печати в абзац. Такая процедура обеспечивает перенос дополнительных данных в абзац, при этом, не сужая диапазон текущих значений.

```
(require-from vector-from-array "vectors.lsp")

(setf statistics-class (send class :new '(count sum sum-sqr max min retain data)))

(send statistics-class :answer :isnew '(ret) '((send self :init ret)))

(send statistics-class :answer :init '(ret) '(
    (setf count 0 sum 0 sum-sqr 0 data nil
          max nil min nil retain ret data nil)))

(send statistics-class :answer :point '(x) '(
    (incf count)
    (setf sum (+ sum x))
    (setf sum-sqr (+ sum-sqr (* x x)))
    (setf max (if max (max max x) x))
    (setf min (if min (min min x) x))
    (if retain (push x data))))

(send statistics-class :answer :get-count '() '(count))
(send statistics-class :answer :get-data '() '(data))
(send statistics-class :answer :get-min '() '(min))
(send statistics-class :answer :get-max '() '(max))

(send statistics-class :answer :get-mean '() '(
    (if (> count 0) (/ (float sum) count)
                    nil)))
```

Код 3

ствии спецификатора. Send — выполняет функцию отправки объекта к примеру (send statistics-class), означает отправку класса статистики для последующей её обработки. Init — является частью конструкции выполнения цикла и используется для инициализации начального значения символа. Например для установки цикла можно использовать следующие LISP-выражения:

(do* (binding…) (texpr[1] rexpr[2]…) expr[3]…)
(do (binding…) (texpr rexpr…) expr…)

Примечательно, что binding является привязкой переменных, которые определяются двумя параметрами. Параметр первый — символ, который инициализируется к значению nil. Параметр второй — список формы, который определяется как sym init [step], где *sym* — является символом связи, *init* — определяет начальное значение символа, *step* — является выражением шага. Isnew — определяет класс «класса» (т.е. класс всех классов объекта, включая себя), класс оперирует сообщениями Isnew (часто встречается в связке с: answer*), запускает инициализацию нового класса и может иметь следующие подстановочные значения:

1 | *ivars* — Список символов переменной экземпляра.
2 | *cvars* — Список символов переменной класса.
3 | *super* — Является суперклассом (по умолчанию может иметь значение object/объект).
4 | returns — Создаёт новый объект класса.

**: answer — также как и isnew входит в состав класс «класса» и оперирует следующими подстановочными значениями для добавления сообщения к классу:*

1 | *msg* — (Сообщение) символ сообщения.
2 | *fargs* — Формальный список параметров (список лямбды).
3 | *code* — Список исполняемых выражений.
4 | returns — (Объект) возврат объекта.
5 | self — Текущий объект в пределах контекста метода[4].

[1] **Прим. автора.** Texpr — тестовое выражение завершения. Тестирует, в каком состоянии находится процесс завершения. В случае если процесс не завершён, отправляет оператор returns — значение которого является последним выражением результата. Чтобы задать основную форму выполнения цикла, можно воспользоваться связкой (loop expr…).

[2] **Прим. автора.** Rexpr — выражение результата (по умолчанию имеет значение nil).

[3] **Прим. автора.** Expr — тело цикла (обрабатываемое как не явная программа).

[4] **Прим. автора.** Метод — это код, который реализует сообщение.

```
if x < 0 then x = -x ; x gets its absolute value
if x > upper-bound then
  begin
    print "x too big, setting to", upper-bound
    x = upper-bound
  end
else
  if x < lower-bound then
    begin
      print "x too small, setting to", lower-bound
      x = lower-bound
    end
```

Код 4

```
begin
  with db = 12.0,
       linear = db-to-linear(db)
  print db, "dB represents a factor of", linear
  set scale-factor = linear
end
```

Код 5

Таким образом, можно дать определения трём объектам, доставшимся языку Nyquist от языка LISP. Селектор (selector) — символ, который выбирает соответствующий метод, сообщение (message) — селектор и список фактических параметров, метод (method) — код который реализует сообщение. Send — выполняет функцию отправки для определения класса объекта, при получении результата пытается найти метод, соответствующий селектору сообщения в наборе имеющихся сообщений, определяемых для предыдущего класса. Бывают такие ситуации, когда сообщение не находится в классе объекта и классу присвоен «суперкласс», тогда поиск продолжится с опорой на сообщение, определённое для «суперкласса». Процесс поиска (в режиме чтения) будет продолжаться от одного суперкласса — до следующего, пока метод для сообщения не будет найден. Если ни один из методов не может быть найден — выдаётся ошибка. В том случае, когда метод все-таки найден, средства анализа связывают объект получения с символом и оценивают метод, используя оставшиеся элементы исходного списка как параметры метода. Параметры обычно оцениваются до того момента, при котором будет связь с их соответствующими «формальными параметрами». В этом случае результат оценки метода становится результатом выражения.

Sqr — выполняет роль арифметической функции, которая вычисляет квадратный корень числа. Имеет регулярное выражение (sqrt *expr*), где *expr — число с плавающей запятой (имеет вложенные функции).* returns — используется для возврата квадратного корня числа. Для того чтобы выполнять арифметические операции, встроенные подоператоры могут иметь опорные характеристики перечисленные ниже:

1 | (< *n1* n2**…) тестирует на меньше чем
2 | (<= *n1 n2*…) тестирует на меньше чем или равно
3 | (= *n1 n2*…) тестирует на равенство
4 | (/= *n1 n2*…) тестирует на неравенство
5 | (>= *n1 n2*…) тестирует на больше чем, или равно
6 | (> *n1 n2*…) тестирует на больше чем
*n1 — первое число, которое подлежит сравнению
*n2 — второе число, которое сравнивается
returns — t, если результаты сравнения n1 с n2, n2 с n3, и т.д., являются всей истиной.

If (если) — LISP-выражение в языке Nyquist обозначает условия выбора. Например, если оператор тестирует выражения, если это истина — она оценивается как истина stmt оператора. В том случае если при выполнении тестирования выражение оценивается как ложь, оно будет оценено как ложь stmt оператора. Можно использовать оператор begin-end для оценки больше, чем одного оператора при условии выбора. Рассмотрим пример работы оператора:

begin [оператор ⟶ инструкция] {оператор} + end

Фактически, begin — это последовательность инструкций, окружающих его операторов (операторов

```
make-sum(make-cycle({1 2 3})
make-cycle({4 5 6})) is (5 7 9)
```

Код 6

```
(send statistics-class :answer :get-stddev '() '(
  (if (> count 1) (sqrt (send self :get-variance)) nil)))

(send statistics-class :answer :get-variance '() '(
  (if (> count 1)
      (/ (- sum-sqr
            (/ (* sum sum) (float count)))
         (1- count))
      nil)))

(send statistics-class :answer :print-stats '() '(
    (format t "Number of points: ~A~%Max: ~A~%Min: ~A~%" count max min)
    (if retain
        (format t "Median: ~A~%" (send self :get-median)))
    (format t "Mean: ~A~%Std.Dev.: ~A~%"
              (send self :get-mean) (send self :get-stddev))
    ))

(send statistics-class :answer :get-data '() '(data))

(send statistics-class :answer :get-median '() '(
  (let (i)
    (cond ((not retain) nil) ;; no data retained to examine
          ((< count 1) nil) ;; no data to compute from
          (t
           (setf data (bigsort data '<))
           (cond ((oddp count)
                  (nth (/ count 2) data))
                 (t
                  (setf i (/ count 2))
                  (* 0.5 (+ (nth i data) (nth (1- i) data)))))))))
```

Код 7

связки). Инструкция находится во взаимодействии с оператором при выполнении близлежащей автономной функции и в процессе исполнения алгоритма описывается ключевыми словами begin ⟶ end[1].

Пример: код 5.

Следует обратить внимание, что в этом примере else всегда связывается с самым близким предыдущим значением при условии, что у близкого значения не имеется выражение else.

Sum — класс суммы формирует сумму чисел по одному от каждого из двух образцов.

Пример: код 6.

Выходная длинна периода (по умолчанию) — это продолжительность входного периода первого параметра. Поэтому первый параметр здесь выступает «образцом», а второй параметр может быть образцом либо числом. Следующий фрагмент кода библиотеки осуществляет отправку класса статистики с целью обработки данных с плавающей запятой (код 7).

[1] **Прим. автора.** Такая форма компоновки кода используется для функциональных определений и унаследована от языка программирования SAL. Обычно, после then/else, где синтаксис требуется для одного оператора, и при этом мы хотим выполнить более одного действия. Также при объявлении переменных может быть задействован дополнительный оператор — with, в иерархии строк, идущий сразу же за begin.

```
(send statistics-class :answer :get-kurtosis '() '(
  (let ((x4 0) x2
        (n (float count)) ; "n" is just a new name for count
        (mean (send self :get-mean))
        (variance (send self :get-variance)))
    (dolist (x data)
      (setf x2 (* (- x mean) (- x mean)))
      (setf x4 (+ x4 (* x2 x2))))
    (display "kurtosis" x4 (* variance variance) n)
    (if (> n 3)
        (- (* (/ (* (1+ n) n)
                 (* (1- n) (- n 2) (- n 3)))
              (/ x4 (* variance variance)))
           (/ (* 3 (1- n) (1- n))
              (* (- n 2) (- n 3))))
        nil))))
```

Код 8

```
;; доля значений в диапазоне
;;
(send statistics-class :answer :fraction-in-range '(low high) '(
  (let ((n 0))
    (dolist (d data)
      (if (and (<= low d) (< d high)) (setf n (1+ n))))
    (/ (float n) count))))
```

Код 9

Здесь вводится контрольная конструкция cond (conditionally), которая позволяет условно оценивать выражения, имеющие парные значения, при обработке списков. Конструкция досталась языку Nyquist в наследство от LISP. В LISP она выполняла функцию условной оценки. Обычно работает в паре с (pred expr…), где pred — является выражением предиката, а expr — выполняет оценку, при условии, что предикат не имеет значения nil. Функция returns, которая дополняет эту конструкцию, в зависимости от обстоятельств возвращает оценённое значение первого выражения, предикат которого не равен nil. Let — является также контрольной конструкцией и организует привязку к переменным, каждая из которых может являться символом либо списком. Следующая часть программного кода библиотеки демонстрирует работу субалгоритма, который должен производить обычную оценку совокупности. Для того чтобы заставить работать подобный алгоритм, необходимо чтобы статистический объект инициализировался для сохранения данных (код 8).

Основными управляющими элементами фрагмента программного кода являются контрольная, циклическая и символьная функции, а также один оператор — унаследованный от языка SAL (let, setf, dolist+display). Let — осуществляет последовательную привязку к ближайшим переменным, для необходимых вычислений с последующим преобразованием полученных данных в число с плавающей запятой через арифметическую функцию (float), далее dolist — организует цикл посредством списка, setf — производит установку поля для результативных математических значений. Variance — определяет дисперсию[1] близких статистическому объекту значений. Далее отдельным входным блоком описывается процедура установления пропорций в выделенном диапазоне (код 9).

Здесь первая строчка кода инициализирует отправку класса статистики с последующим промежуточным ответом от части диапазона в нижних и высоких пределах. Четвёртая строчка — начинается с условной оценки выражения и осуществляет преобразование целого числа в символ, после операторов сравнения низкого и высокого разделов диапазона (<=,<) setf — присваивает число полевому спецификатору. Далее идёт разделение

[1] **Прим. автора.** Дисперсия — среднеквадратическое отклонение. В теории вероятностей и математической статистике — ожидание квадратического отклонения случайной величины от её среднего значения. Применительно к ряду может быть коэффициентом корреляции. В теории вероятностей и математической статистике коэффициент вариации (также известный как относительное стандартное отклонение), является стандартизированной мерой дисперсии распределения вероятностей или частотного распределения. Часто выражается в процентах и определяется как отношение стандартного отклонения (сигма) к среднему значению ряда ню.

```
Low  - 0.1 0.0 1.0
          ðÿä-1
High - 1.0 0.0 0.1
          ðÿä-1
```

Код 10

```
(setf my-histogram (send histogram-class :new))
setf my-histogram ⟶ histogram-class
```

Код 11

```
(send my-histogram :point x)
```

Код 12

```
(aref an-array 3) <= x < (aref an-array 4)
```

Код 13

```
(send my-histogram :make-hist)
```

Код 14)

```
(send my-histogram :gnu-plot filename xlabel ylabel title [categories])
```

Код 15

числа списка, арифметическая функция float — преобразует целое число в число с плавающей запятой для проброса потока и последующего расчёта числа потока. Значения для low и high могут быть в приближенном диапазоне (код 10).

Класс гистограмм составляет гистограмму по данным и зависит от файла vectors.lsp, поэтому прежде, чем перейти к просчёту гистограммы, необходимо загрузить векторы. Прежде, чем использовать класс гистограмм, алгоритм предусматривает создание обрабатываемого экземпляра (код 11).

После описанной процедуры добавляются точки к гистограмме, для каждой точки x (код 12).

Происходит отправление my-histogram → точке x. На данном этапе можно создать гистограмму по умолчанию, осуществив вызов (send my-histogram: configure-bins), таким образом, создастся квадратный корень из N ячеек, где N — число точек, распределённых равномерно по всему диапазону данных. Кроме того, имеется возможность указать свои собственные (штатные) пороговые значения для определения ячеек, вызвав send my-histogram: set-thresholds an-array. Каждый элемент an-array — представляет нижнюю границу для элементов в этой ячейке, то есть, если x является точкой, он входит в ячейку 3 (код 13).

Здесь стоит обратить внимание на то, что никакие данные не входят в ячейку L-1, где L — длина an-array. Для функционального вычисления гистограммы вызываем (код 14).

и осуществляем её печать и построение с помощью send my-histogram: print-hist или send my-histogram: plot-hist. При использовании sal-load “gnuplot”, можно также осуществлять вывод данных в графическом виде

```
[экземпляр]
(setf histogram-class (send class :new '(stats counts thresholds)))

(send histogram-class :answer :isnew '() '((send self :init)))

(send histogram-class :answer :init '() '(
    (setf counts nil thresholds nil)
```

Код 16

```
    (setf stats (send statistics-class :new t))))

    (send histogram-class :answer :point '(x) '(
        (send stats :point x)))

    (send histogram-class :answer :configure-bins '() '(
        (let* ((nbins (round (sqrt (float (send stats :get-count)))))
               (minthreshold (send stats :get-min))
               (step (/ (- (send stats :get-max) (send stats :get-min))
                        (float nbins))))
          (setf thresholds (make-array (1+ nbins)))
          (dotimes (i (1+ nbins))
            (setf (aref thresholds i) (+ minthreshold (* i step))))
```

Код 17

```
  (setf (aref thresholds nbins) (* (aref thresholds nbins) 1.000001))
  (display "configure-bins" minthreshold (send stats :get-max) thresholds)
  thresholds)))

  (send histogram-class :answer :set-thresholds '(array) '(
  (setf counts nil)
  (setf thresholds array)))
```

Код 18

(код 15), где категории по умолчанию верны и указывают, должны ли отображаемые полосы гистограммы быть центрированы по низкому значению порогового диапазона или от установленного диапазона от низкого значения — к высокому. Если каждая ячейка представляет одно из целых чисел, тогда используется — t. Если же это гистограмма непрерывных реальных значений, которые попадают в заданный алгоритмом, используется диапазон — nil (false). Можно также изменять установленные пороговые значения с помощью: set-thresholds и: configure-bins, минуя повторную вставку (установку) всех точек. Чтобы осуществить заново все проделанные ранее процедуры, по такому же алгоритму можно воспользоваться командой send my-histogram: init. Но это не является лучшим выходом из ситуации, когда необходимо обнулять все имеющиеся значения. Для этого лучше создать новый экземпляр (код 16).

После создания экземпляра, алгоритм создаёт объект статистики с последующим сохранением точки значения (код 17).

Последняя ячейка будет немного объёмнее, чем остальные, для того чтобы уместить максимальное значение[1](код 18).

[1] **Прим. автора.** (от автора алгоритма — Роджера Данненберга (Roger B. Dannenberg)). Первоначально установленный порог был +1e-6, но за тем, когда значение равнялось 1e6, наступило переполнение и операция не сработала.

```
(send histogram-class :answer :make-hist '(&key (verbose t)) '(
    (let* ((data (send stats :get-data))
           (counter 0) (data-position 0))
      (if (null thresholds)
          (send self :configure-bins))
      (cond ((null counts)
             (setf counts (make-array (1- (length thresholds))))
             (dotimes (i (length counts))
                      (setf (aref counts i) 0))))
      (dolist (x data)
        (cond ((and verbose (> counter 100000))
               (format t "make-hist ~A% done\n"
                       (* 100
                          (/ data-position (float (send stats :get-count)))))
               (setf counter 0)))
```

Код 19

```
              (dotimes (i (length counts))
                (incf counter)
                (cond ((and (< x (aref thresholds (1+ i)))
                            (>= x (aref thresholds i)))
                       (incf (aref counts i))
                       (return))))
              (incf data-position)) )))
```

Код 20

```
      (send histogram-class :answer :print-hist '() '(
          (if (null counts) (send self :make-hist))
          (dotimes (i (length counts))
                   (format t "~A to ~A: ~A~%"
                           (aref thresholds i) (aref thresholds (1+ i))
                           (aref counts i)))))
      (send histogram-class :answer :plot-hist '(&optional (offset 0)) '(
        (let (args time cnt)
          (if (null counts) (send self :make-hist))
          (setf time (float (aref thresholds 0)))
```

Код 21

Здесь stats[1] позволяет определить, сколько полных байтов было выделено таблицам. Make-array — является функцией массива и создаёт новый массив. Dotimes[2] — является конструкцией цикла, выполняет цикл от 0 до n1 (код 19).

Увеличение правой ячейки позволяет использовать разные размеры ячеек и здесь можно использовать двоичный поиск необходимой ячейки (код 20).

Dolist[3] — также циклическая конструкция, она выполняет цикл через список. Incf — относится к полезным функциям и является инкрементным символом. Может также быть в виде макроса, либо ещё какой-либо ин-

[1] **Прим. автора.** Определяет состояние использования памяти. Позволяет узнать, сколько памяти используется экземплярами осциллятора поиска по таблице.

[2] **Прим. автора.** Dotimes использует следующие вспомогательные операторы:
1) sym — Символ, связывающий каждое значение от 0 до n-1.
2) expr — Число раз, определяет хронометраж цикла.
3) rexpr — Выражение результата (значение по умолчанию nil).
4) expr — Тело цикла (обработанное как неявная программа).

[3] **Прим. автора.** Dolist использует следующие вспомогательные операторы:
1) sym — Символ связи с каждым элементом списка.
2) expr — Выражение списка.
3) rexpr — Выражение результата (значение по умолчанию nil).
4) expr — Тело цикла (обработанное как неявная программа).

```
(setf args (list time))
    (setf args (cons 0.0 args))
    (dotimes (i (length counts))
      (setf cnt (float (aref counts i)))
      (setf args (cons time args))
      (setf args (cons cnt args))
      (setf time (float (aref thresholds (1+ i))))
      (setf args (cons time args))
      (setf args (cons cnt args)))
    (s-plot (pwl-list (reverse args)) time))))

(send histogram-class :answer :gnu-plot '(filename xlabel ylabel title
                                          &optional (categories t)) '(
  (let ((thresh-list (vector-from-array thresholds))
        (counts-list (vector-from-array counts))
        (low-wid (- (aref thresholds 1) (aref thresholds 0)))
        (high-wid (- (aref thresholds (- (length thresholds) 1))
                     (aref thresholds (- (length thresholds) 2))))
        xrange x)
```

Код 22

струкцией, установленной setf, но, как правило, бывает символом переменной, который в свою очередь может быть элементом массива (код 21).

Здесь let (args time cnt) производит вычисление аргументов по порядку, с последующим приведением аргументов в пользу списка pwl.

Format[1] — служит функцией формата и создаёт выходной формат. Строка формата содержит символы, которые должны быть скопированы непосредственно в директивы вывода и форматирования. Директивы форматирования определены следующие:

1 | [~A] — Печатает следующий параметр, используя (princ).
2 | [~S] — Печатает следующий параметр, используя (prin1).
3 | [~%] — Запускает новую строку.
4 | [~~] — Печатает символ тильды.

Прим. автора. Здесь подразумеваются две группы, соответствующие нормальному распределению по Гауссу*. Это наиболее распространенная колоколообразная кривая в статистике (см. рис. 5, данной статьи).

* Иоганн Карл Фридрих Гаусс (нем. Johann Carl Friedrich Gauß, англ. Johann Carl Friedrich Gauss) — немецкий математик и физик, внесший значительный вклад в развитие многих областей математики и естествознания. Иногда его называют Princeps mathematicorum (по-латыни «выдающийся из математиков») и «величайший математик со времен античности». Гаусс оказал исключительное влияние во многих областях математики и естественной науки и входит в число самых влиятельных математиков в истории. Гаусс доказал *метод наименьших квадратов* (method of least squares) (процедура, используемая во всех науках по сей день, чтобы свести к минимуму влияние ошибки измерения) в предположении о *нормально распределенных ошибках* (теорема Гаусса-Маркова** (Gauss-Markov theorem)); см. также по Гауссу (Gaussian — список научных названий, в честь Гаусса). Метод был описан Адрианом-Мари Лежандром (Adrien-Marie Legendre) в 1805 году, но Гаусс утверждал, что использовал его с 1794 или 1795 года. В истории статистики это разногласие называется «спор о приоритете открытия метода наименьших квадратов».

5 | [newline>] — Игнорирует новую строку и пробел на следующей строке до первого неразрывного символа или новой строки. Это позволяет строкам соединяться через многократные строки (код22).

S-plot[2] — функция, выводящая звук в графическое изображение. Функция была разработана для запуска программы построения графиков на рабочей станции Unix, но теперь в основном используется с NyquistIDE, ко-

[1] **Прим. автора.** stream — поток на выходе:
1. fmt — Строка формата (строка определяющая положение формата).
2. arg — Параметры формата.
3. returns — Строка выхода, если поток — ноль.

[2] **Прим. автора.** Некоторые вспомогательные (сопутствующие) функции, служащие вспомогательной базой при визуализации статистических данных аудиосигнала: spec-plot, *spec-plot-res*, *spec-plot-bw*,*spec-plot-db*, s-print-tree.

```
(setf xrange (list (- (apply 'min thresh-list) (/ low-wid 2.0))
                              (+ (apply 'max thresh-list) (/ high-wid 2.0)))))
          (t
          (setf xrange (list (apply 'min thresh-list)
                              (apply 'max thresh-list)))))
    (gp-init filename :xlabel xlabel :ylabel ylabel :title title
             :style :histogram
             :xrange xrange
             :yrange (list 0 (apply 'max counts-list)))
    (gp-newcurve)
    (dotimes (i (length counts))
      (setf x (pop thresh-list))
      (if (not categories) (setf x (* 0.5 (+ x (car thresh-list)))))
      (gp-point x (pop counts-list)))
    (gp-endcurve)
    (gp-endplot))))

(send histogram-class :answer :get-min '() '(
    (send stats :get-min)))

(send histogram-class :answer :get-max '() '(
    (send stats :get-max)))

(send histogram-class :answer :get-count '() '(
    (send stats :get-count)))

(send histogram-class :answer :get-counts '() '(
    counts))

(send histogram-class :answer :get-thresholds '() '(
    thresholds))

(send histogram-class :answer :get-stats '() '(
    stats))
```

Код 23

торый имеет автономное построение графиков. Обычно пары время/значение в ascii записываются в points.dat и, зависящий от системы код или программа NyquistIDE, берут его оттуда. Если sound продолжительнее, чем необязательный dur (по умолчанию 2 секунды), отображаются только первые секунды dur. Если для построения графика требуется более n выборок, сигнал интерполируется, чтобы перед построением графика было n выборок. Используемый файл данных*default-plot-file*[1].

Определяем, что области гистограммы сверху и снизу — это величина расстояния до следующей или предыдущей ячейки, но с центром в точке данных, поэтому здесь необходимо произвести настройку x-range для размещения объёмно-закрашенных[2] областей (код 23).

Здесь стоит обратить внимание на ветку (setf cnt (float (aref counts i)), где печатается временной интервал с функцией списка conc для создания очередного узла списка. Если звук многоканальный (более одного канала — стерео, квадро и т.д.), то время учёта обновлений выдачи данных по списку на гистограмме будет изменяться в соответствии с параметрами (a), то есть шаг об-

[1] **Прим. автора.** Файл, содержащий точки данных по умолчанию — имеет значение «points.dat».

[2] **Прим. автора.** Имеются в виду те области, которые закрашены чёрным цветом с элементами «жирности». Жирная окраска позволяет производить идентификацию и размежевание пороговых значений.

$$(a)\begin{vmatrix} \mathtt{setf=args/}\underset{0\leftarrow 1}{\mathtt{time}} \\ \underset{1\rightarrow 0}{} \\ \mathtt{setf=time/args} \end{vmatrix} = \mathtt{time}\begin{bmatrix} 000\rightarrow 1 & 00\rightarrow 1 & 0\rightarrow 1 \\ 000 & 000 & 000 \\ 000 & 000 & 000 \end{bmatrix}$$

$$(b)\begin{vmatrix} \mathtt{setf=args/}\underset{1\leftarrow 0}{\mathtt{time}} \\ \underset{0\rightarrow 1}{} \\ \mathtt{setf=time/args} \end{vmatrix} = \mathtt{time}\begin{bmatrix} 0\rightarrow 1 & 00\rightarrow 1 & 000\rightarrow 1 \\ 000 & 000 & 000 \\ 000 & 000 & 000 \end{bmatrix}$$

Код 24

```
;(setf upearson-class (send class :new '(sumxy sumx sumy sumxx sumyy n)))
;
;(send upearson-class :answer :isnew '() '((send self :init)))
;(send upearson-class :answer :init '() '(
;      (setf sumxy 0 sumx 0 sumy 0 sumxx 0 sumyy 0 n 0)))
;(send upearson-class :answer :points '(x y) '(
;      (setf sumxy (+ sumxy (* x y)))
;      (setf sumx (+ sumx x))
;      (setf sumy (+ sumy y))
;      (setf sumxx (+ sumxx (* x x)))
;      (setf sumyy (+ sumyy (* y y)))
;      (setf n (+ n 1))))
;(send upearson-class :answer :correlation '() '(
;      (/ (- (* n sumxy) (* sumx sumy))
;         (* (sqrt (- (* n sumxx) (* sumx sumx)))
;            (sqrt (- (* n sumyy) (* sumy sumy)))))))
```

Код 25

новления числовых показателей будет более плотным, в то время как одноканальный звук будет иметь менее плотную степень обновления числовых показателей (б). Это связано с тем, что при визуализации звукового спектра монодорожка имеет более упрощённую структуру и при её масштабировании, значения, отображающие узкие частоты, будут иметь погрешность (код24).

После необходимых манипуляций с состояниями совокупностей и обработки пропорций библиотекой, подразумевается работа с использованием корреляции Пирсона[1]. Чуть ниже описывается его прямой нестабильный алгоритм, составленный исследователем в области компьютерных наук Роджером Данненбергом[2].

[1] **Прим. автора.** Корреляция Пирсона — операция обработки статистических данных с целью нахождения коэффициента корреляции, первоначально разработанного Огюстом Браве (Auguste Bravais) и Фрэнсисом Гальтоном (Francis Galton) и впоследствии доработанным Карлом Пирсоном* (Karl Pearson, при рождении Carl Pearson).

* Карл Пирсон — британский биостатистик и математик. Во многих научных трудах, посвящённых анализу статистических данных, упоминается как создатель математической статистики. Он основал первый в мире университетский статистический факультет в Университетском колледже Лондона в 1911 г. и внёс значительный вклад в область биометрии и метеорологии. Анализируя биографию Пирсона на английском языке, можно сделать вывод, что круг его научных интересов был крайне широк — от философии и филологии, до антропометрии, биометрии и математики. Но особой популярностью у научных исследователей различного толка пользуются наработки Пирсона в области статистики. Пирсону принадлежат заслуги в области совершенствования классических статистических методов, в частности коэффициент корреляции, метод моментов, система непрерывных кривых (система непрерывных одномерных распределений вероятностей), критерий хи-квадрата, метод «подгонки» линейного подпространства к многомерным данным путем минимизации расстояний хи и т.д.

[2] **Прим. автора.** Роджер Данненберг (Roger B. Dannenberg) — американский исследователь в области компьютерных наук школы компьютерных наук

```
(setf pearson-class (send class :new '(sum-sq-x sum-sq-y sum-coproduct
                                       mean-x mean-y n)))
(send pearson-class :answer :isnew '() '((send self :init)))
(send pearson-class :answer :init '() '(
    (setf n 0)
    (setf sum-sq-x 0 sum-sq-y 0 sum-coproduct 0)))

(send pearson-class :answer :points '(x y) '(
    (cond ((zerop n)
           (setf mean-x x mean-y y n 1))
          (t
           (setf n (1+ n))
           (let* ((sweep (/ (- n 1.0) n))
                  (delta-x (- x mean-x))
                  (delta-y (- y mean-y)))
             (setf sum-sq-x (+ sum-sq-x (* delta-x delta-x sweep)))
             (setf sum-sq-y (+ sum-sq-y (* delta-y delta-y sweep)))
             (setf sum-coproduct (+ sum-coproduct (* delta-x delta-y sweep)))
             (setf mean-x (+ mean-x (/ delta-x n)))
             (setf mean-y (+ mean-y (/ delta-y n)))))))))

(send pearson-class :answer :correlation '() '(
    (let* ((pop-sd-x (sqrt (/ sum-sq-x n)))
           (pop-sd-y (sqrt (/ sum-sq-y n)))
           (cov-x-y (/ sum-coproduct n)))
      (/ cov-x-y (* pop-sd-x pop-sd-y)))))
```

Код 26

Алгоритм создан для получения истинного ответа в процессе отладки, приведённой чуть ниже, более сложной версии. Все три алгоритма согласованы в рамках численного округления и pearson-class является улучшенной реализацией (код 25).

Корреляция Пирсона (код 26).

Здесь pop — отбрасывает первый элемент LISP-списка, который может быть в виде макроса, в LISP его эквивалент будет (setf lis (cdr lis)). Сам список возвращается — но не глава списка, которая была отброшена. Для получения главы списка можно использовать first или car. Алгоритм, приведённый ниже, предполагает прямое исполнение, хранит точки и создан в целях отладки[1](код 27).

Алгоритм из Википедии (код 28).

С этого абзаца начинается описание алгоритма теста Уэлча[2] для проверки нулевой гипотезы о том, что две со-

на правах факультета при университете Карнеги-Меллона. Является создателем и ведущим разработчиком языка Nyquist, осуществляет поддержку технических электронных регламентов в области Nyquist-программирования. Научные интересы: синтез звука (в том числе генерация звуков, тональностей и т.д.), музыкальное программирование, редактирование звука программными средствами. Подробнее см. статью, указанную в пункте третьем нашего списка литературы [3].

[1] **Прим. автора.** Оригинальное предписание по структуре программного кода помеченное Роджером Данненбергом.

[2] **Прим. автора.** В статистике тест Уэлча (t-критерий Уэлча) или t-критерий неравных отклонений, представляет собой тест на определение местоположения с двумя выборками, который используется для проверки гипотезы о том, что две популяции имеют равные средние значения. Он назван в честь своего создателя Бернарда Льюиса Уэлча* и является адаптацией t-критерия Стьюдента**. Он более надёжен, когда две выборки имеют неодинаковые отклонения и / или неодинаковые размеры выборки. Эти тесты часто называют t-тестами «непарных» или «независимых выборок», поскольку они обычно применяются, когда статистические единицы, лежащие в основе двух сравниваемых выборок, не пересекаются. Учитывая, что t-тест Уэлча был менее популярен, чем t-тест Стьюдента, и может быть менее знаком читателям, более информативным названием является «t-тест неравных отклонений Уэлча» — или «t-тест неравных отклонений» для краткости.
* Бернард Льюис Уэлч (Bernard Lewis Welch) — британский статистик и педагог. Он является создателем t-критерия Уэлча. Уэлч обучался в Лондонском университетском колледже по программе статистики. Пирсон и Фишер

```
;(setf npearson-class (send class :new '(pts)))
;(send npearson-class :answer :isnew '() '((send self :init)))
;(send npearson-class :answer :init '() '((setf pts nil)))
;(send npearson-class :answer :points '(x y) '(
;      (setf pts (cons (cons x y) pts))))
;(send npearson-class :answer :correlation '() '(
;  (setf pts (reverse pts))
;  (let ((sum-sq-x 0) (sum-sq-y 0) (sum-coproduct 0) (mean-x (caar pts))
;        (mean-y (cdar pts)) i (n (length pts)))
;    (dotimes (j (1- n))
;      (let* ((i (+ j 2))
;             (sweep (/ (- i 1.0) i))
;             (delta-x (- (car (nth (1- i) pts)) mean-x))
;             (delta-y (- (cdr (nth (1- i) pts)) mean-y)))
;        (setf sum-sq-x (+ sum-sq-x (* delta-x delta-x sweep)))
;        (setf sum-sq-y (+ sum-sq-y (* delta-y delta-y sweep)))
;        (setf sum-coproduct (+ sum-coproduct (* delta-x delta-y sweep)))
;        (setf mean-x (+ mean-x (/ delta-x i)))
;        (setf mean-y (+ mean-y (/ delta-y i)))))
;    (let ((pop-sd-x (sqrt (/ sum-sq-x n)))
;          (pop-sd-y (sqrt (/ sum-sq-y n)))
;          (cov-x-y (/ sum-coproduct n)))
;      (/ cov-x-y (* pop-sd-x pop-sd-y))))))
```

Код 27

```
;sum_sq_x = 0
;sum_sq_y = 0
;sum_coproduct = 0
;mean_x = x[1]
;mean_y = y[1]
;for i in 2 to N:
;    sweep = (i - 1.0) / i
;    delta_x = x[i] - mean_x
;    delta_y = y[i] - mean_y
;    sum_sq_x += delta_x * delta_x * sweep
;    sum_sq_y += delta_y * delta_y * sweep
;    sum_coproduct += delta_x * delta_y * sweep
;    mean_x += delta_x / i
;    mean_y += delta_y / i
;pop_sd_x = sqrt( sum_sq_x / N )
;pop_sd_y = sqrt( sum_sq_y / N )
;cov_x_y = sum_coproduct / N
;correlation = cov_x_y / (pop_sd_x * pop_sd_y)
```

Код 28

```
(defun welchs-t-test (mean1 stddev1 n1 mean2 stddev2 n2)
  (let* ((var1 (* stddev1 stddev1))
         (var2 (* stddev2 stddev2))
         (num (- mean1 mean2))
         (den (sqrt (+ (/ var1 n1)
                       (/ var2 n2))))
         (welchs-t (/ num den))
         (dof-a (+ (/ var1 n1) (/ var2 n2)))
         (dof-num (* dof-a dof-a))
         (dof-den (+ (/ (* var1 var1) (* n1 n1 (- n1 1)))
                     (/ (* var2 var2) (* n2 n2 (- n2 1)))))
         (dof (/ dof-num dof-den)))
  (list welchs-t dof)))
```

Код 29

вокупности равны в то время как дисперсии могут быть неравными. «t»-критерий Уэлча[1] для проверки нулевой гипотезы о том, что два средних значения популяции равны, когда дисперсии (отклонения) могут быть неравными (код 29).

Здесь излагается программное описание алгоритма критерия Левена[2] для оценки равенства дисперсий в различных выборках[3]. Данная реализация алгоритма предназначена для 2-х групп. Если две группы можно считать нормальными (гауссовскими)[4], тогда следует рассмо-

[1] **Прим. автора.** В отличие от t-критерия Стьюдента, который предполагает, что две сравниваемые популяции обычно распределены с равными отклонениями, t-критерий Уэлча предназначен для неравных различий в численности населения, но при этом сохраняется предположение о нормальности. «t» — критерий Уэлча является приближенным решением проблемы Беренса-Фишера*.

* В статистике проблема Беренса-Фишера (Behrens-Fisher problem), названная в честь Вальтера Беренса (Walter Behrens)** и Рональда Фишера (Ronald Fisher)***, представляет собой проблему оценки интервалов и проверки гипотез относительно разницы между средними значениями двух нормально распределенных популяций, когда отклонения двух популяций не предполагаются равными, на основе двух независимых выборок.

** Вальтер-Ульрих Беренс (Walter-Ulrich Behrens) немецкий химик и статистик, который совместно с Рональдом Фишером (Ronald Fisher) открыл проблему Беренса-Фишера (Behrens-Fisher problem) и связанное с ней распределение Беренса-Фишера (Behrens-Fisher distribution).

***Рональд Фишер (Ronald Aylmer Fisher, Sir Ronald Aylmer Fisher) — британский энциклопедист и биолог, который активно работал как математик, статистик, генетик и преподаватель. Член Лондонского королевского общества, обладатель титула Сэра. За свою работу в области статистики он был определен как «гений, который почти в одиночку создал основы современной статистической науки» и как «самая важная фигура в статистике 20-го века». В 1935 г. Фишер опубликовал статью о *фидуциальном выводе'* и применил результаты исследования к проблеме Беренса-Фишера (Behrens-Fisher problem), решением которой, предложенным сначала Вальтером Беренсом, а несколько лет спустя самим Фишером, является распределение Беренса-Фишера (Behrens-Fisher distribution).

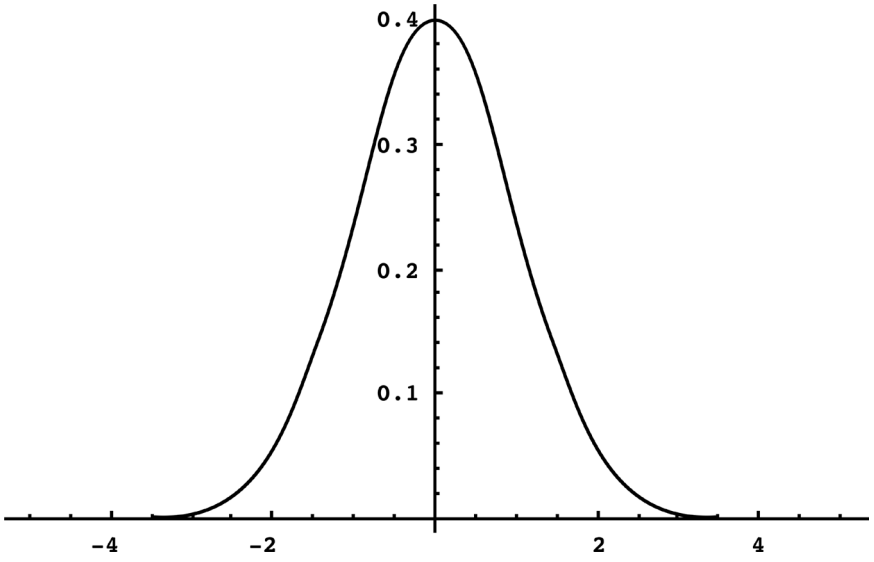
' Фидуциальный (*доверительный*) вывод — это один из множества различных типов статистических выводов. Это правила, предназначенные для общего применения, с помощью которых можно делать выводы из *выборок* данных. В современной статистической практике попытки работать с фидуциальным выводом вышли из моды, уступив *частотному* выводу, *байесовскому* выводу и теории принятия решений. Однако, фидуциальный вывод важен в истории статистики, поскольку его развитие привело к параллельному развитию концепций и инструментов в теоретической статистике, которые широко используются. Некоторые текущие исследования в области статистической методологии в различной степени связаны с фидуциальным выводом.

[2] **Прим. автора.** Говард Левин (Howard Levene, произносится как Лави'н или Лэви'н) —американский статистик в области биологии и генетик. Родился в городе Нью-Йорк. Окончил Нью-Йоркский университет. В 1941 г. перешёл в Колумбийский университет, где работал над проблемой контроля качества в условиях войны. В 1947 г. получил степень доктора философии (PhD) в Колумбийском университете и, вскоре после этого, стал работать на одном из его факультетов. По другим данным он получил степень PhD в 1953 году и тема его диссертации (консультант — Джейкоб Вулфовиц (Jacob Wolfowitz))* — «Вклад в теорию непараметрических критериев случайности». В Колумбийском университете он работал профессором математической статистики и генетики до 1982 года. Известен разработкой теста Левена (1960 год) — *модифицированной формы одностороннего дисперсионного анализа*. Занимал пост президента Американского общества натуралистов в 1976 году.

* Джейкоб Вулфовиц (Jacob Wolfowitz) — американский статистик польского происхождения, лауреат премии Шеннона в области теории информации. Он был отцом бывшего заместителя министра обороны Соединенных Штатов и президента Группы Всемирного банка Пола Вулфовица (Paul Wolfowitz).

[3] **Прим. автора.** Концепция теста основывается на открытых источниках, преимущественно описываемых в Википедии.

[4] ** В статистике теорема Гаусса-Маркова (или просто теорема Гаусса для некоторых авторов) гласит, что обычная оценка наименьших квадратов (**O**rdinary **l**east **s**quares, OLS) имеет наименьшую дисперсию выборки в классе линейных несмещенных оценок, если ошибки в модели линейной регрессии некоррелированы, имеют равные отклонения и математическое ожидание равное нулю. Ошибки не обязательно должны быть нормальными, и при этом они не должны быть независимыми и одинаково распределёнными (только некоррелированными со средним нулём и гомоскедастическими (то есть со свойством, означающим постоянство условной дисперсии вектора или последовательности случайных величин) с конечной дисперсией). Требование о том, чтобы оценщик (правило для вычисления оценки заданной величины на основе наблюдаемых данных) (estimator) был беспристрастным, не может быть отменено, поскольку существуют предвзятые оценщики



Примечание: составлено автором* — Таран В.В., по материалам Nyquist Reference Manual Version 2.36, 2007. В качестве уточнения (простейшая форма распределения по Гауссу) использовался электронный ресурс Zhang X. (2011) Gaussian Distribution. In: Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-30164-8_323 [электронный ресурс, дата обращения к электронному ресурсу: 03.03.2022].

* Прим. автора. При подготовке графика использовалось программное обеспечение компании Wolfram Research — Wolfram Mathematica 10.2 и, функционально соответствующий версии, язык программирования Wolfram.

Рис. 5. Распределение по Гауссу (колоколообразная форма кривой — xmu = 0, sigma = 1).

треть F-критерий. Вариацией критерия Левена является критерий Брауна-Форсайта[1], в котором вместо *средних* используются *медианы (вместо средних значений используются медианные значения)*. Необязательный (или дополнительный) параметр brown-forsythe можно установить на значение true, чтобы осуществить проверку по критерию Брауна-Форсайта вместо критерия Левена. Режим подробной информации по умолчанию имеет значение t и выводит некоторую полезную информацию (входные данные для levenes-test) — это пара списков образцов. Входные данные для levenes-test — это пара списков образцов. Возвращаемое значение — W^2. Критерий Левена для оценки равенства дисперсий в различных выборках — код сгенерирован на основе статьи в Википедии. Критерий выполняется для двух групп, если две группы можно считать нормальными (гауссовскими), то следует рассмотреть F-тест (F — критерий, или критерий Фишера).

Разновидностью критерия Левена является критерий Брауна-Форсайта, в котором вместо средних значений

с меньшей дисперсией. Смотрите, например, оценщик Джеймса-Стайна (James-Stein estimator) (который также снижает линейность), гребневую регрессию или просто любой редуцированный оценщик. Теорема была названа в честь Карла Фридриха Гаусса и Андрея Маркова, хотя работа Гаусса значительно предшествовала работе Маркова. Но в то время как Гаусс вывел результат в предположении независимости и нормальности, Марков привел допущения к форме, указанной выше. Дальнейшее обобщение с учётом несферических ошибок было дано Александром Айткеном (Alexander Aitken), одним из самых выдающихся математиков Новой Зеландии.

[1] **Прим. автора.** Критерий Брауна-Форсайта — это статистический критерий (тест) на равенство групповых дисперсий, основанный на выполнении дисперсионного анализа ANOVA (**AN**alysis **O**f **VA**riance) при преобразовании независимой переменной (результирующей переменной, переменной отклика). Критерий (тест) назван по имени Мортона Брауна (Morton Brown), родившегося в Канаде американо-израильского исследователя в области биоматематики и статистики и Алана Форсайта (Alan Forsythe), специалиста в области клинического анализа и биологической статистики. Является соавтором Мортона Брауна (Morton Brown) по многим научным статьям, касающимся вопросов дисперсионного анализа, а также применения устойчивого критерия равенства дисперсий.

[2] **Прим. автора.** Более подробное описание по тесту Уэлча изложено в англоязычной Википедии по адресу: https://en.wikipedia.org/wiki/Levene%27s_test [электронный ресурс, дата обращения к электронному ресурсу: 03.03.2022]. Очень хорошо и доступно (формульно) изложен данный тест в украиноязычной Википедии по адресу: https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82_%D0%9B%D0%B5%D0%B2%D0%B5%D0%BD%D0%B5 [электронный ресурс, дата обращения к электронному ресурсу: 03.03.2022]. Также исходный принцип теста опубликован здесь: https://www.itl.nist.gov/div898/handbook/eda/section3/eda35a.htm [электронный ресурс, дата обращения к электронному ресурсу: 03.03.2022].

```
(defun levenes-test (y1 y2 &optional brown-forsythe (verbose t))
  (let* ((n1 (float (length y1)))
         (n2 (float (length y2)))
         (n (+ n1 n2))
         m1 m2 z1 z2 z.. z1. z2. stat (den 0) w)
```

Код 30

```
(cond (brown-forsythe
           (setf m1 (vector-median y1))
           (setf m2 (vector-median y2)))
           (t
           (setf m1  (vector-mean y1))
           (setf  m2  (vector-mean  y2))))
```

Код 31

```
(dolist (y1j y1) (push (abs (- y1j m1)) z1))
   (dolist (y2j y2) (push (abs (- y2j m2)) z2))
```

Код 32

```
(setf z1. (vector-sum-elements z1))
(setf z2. (vector-sum-elements z2))
```

Код 33

```
(setf z.. (/ (+ z1. z2.) n))
```

Код 34

```
(setf z1. (/ z1. n1))
(setf z2. (/ z2. n2))
```

Код 35

используются медианные значения. Необязательный параметр Браун-Форсайт может быть установлен на «истину» для того, чтобы получить критерий Брауна-Форсайта вместо критерия Левена. Текстовая метка по умолчанию имеет значение t и выводит некоторую полезную информацию. Входные данные для критерия Левена представляют собой пару списков выборок. Возвращаемым значением является W (подробности см. в Википедии), код 30.

Вычисляем средние или медианные значения, код 31.

Вычисляем zij (lists z1 and z2), код 32.

Вычисляем zi. Sums, код 33.

Вычисляем z… Код 34.

Преобразуем zi. переменных из сумм в средние значения, код 35.

Вычисление большого компонента знаменателя, код 36.

```
            (dolist (z1j z1)
                  (let ((diff (- z1j z1.)))
                    (setf den (+ den (* diff diff)))))
                (dolist (z2j z2)
                  (let ((diff (- z2j z2.)))
                    (setf den (+ den (* diff diff)))))
```

Код 36

```
(setf w (* (- n 2) (/ (+ (* n1 (* (- z1. z..) (- z1. z..)))
                         (* n2 (* (- z2. z..) (- z2. z..))))
                      den)))
```

Код 37

```
        (cond (verbose
                (format t "Summary of ~A test results:
        Size of group 1: ~A, ~A: ~A
        Size of group 2: ~A, ~A: ~A
        W (result): ~A
```

Код 38

```
  (if brown-forsythe "Brown-Forsythe" "Levene's")
                    n1 (if brown-forsythe "Median" "Mean") m1
                    n2 (if brown-forsythe "Median" "Mean") m2
                    w
                    (- n 2) (- n 2))))
      w))
```

Код 39

```
(defun levenes-test-test ()
  (let (y1 y2 y3)
    (dotimes (i 50)
      (push (gaussian-dist 1.0 0.1) y1))
    (dotimes (i 75)
      (push (gaussian-dist 1.0 0.2) y2))
    (dotimes (i 75)
      (push (gaussian-dist 1.0 0.1) y3))
    (format t "\nTHE FOLLOWING HAVE UNEQUAL VARIANCE\n")
    (levenes-test y1 y2) ;; тест Левена
    (format t "\n")
    (levenes-test y1 y2 t) ;; тест Брауна-Форсайта
    (format t "\nTHE FOLLOWING HAVE EQUAL VARIANCE\n")
    (levenes-test y1 y3) ;; тест Левена
    (format t "\n")
    (levenes-test y1 y3 t) ;; brown-forsythe test
    (format t "\n")
    'done
  ))
```

Код 40

Вычисляем w… Код 37.

Распечатываем информацию, если она содержит много знаков, код 38.

Значимость W проверяется по отношению к F(альфа, 1, ~A), где альфа-уровень значимости (обычно 0,05 или 0.01), и ~A равно N-2.~%", код 39.

Простая проверка по levenes-test. Данная программа использует distributions.lsp, который должен быть обязательно загружен (явным образом).

Создаём данные с сигмой 0.1 и 0.2, код 40.

Здесь gaussian-dist возвращает значение FLONUM из распределения Гаусса или Гаусса-Лапласа[1] линейной функции нормального распределения. Он симметричен относительно среднего значения xmu со стандартным отклонением sigma, которое должно быть больше нуля. Параметры low и high показывают необязательные искусственные границы минимального и максимального выходных значений соответственно. Push — макрос эквивалентный написанию в LISP (setf lis (cons val lis)). То есть, используя val — нажмите на lis (список LISP). LISP = (push val lis), SAL = push(val, lis).

Рассмотрев ключевые функции библиотеки «Statistics», а также исследовав её программный код, можно сделать вывод следующего содержания: технические инструменты, предлагаемые данной библиотекой, обширны и представляют собой внушительный перечень опций, направленных на проведение качественного статистического анализа аудиоматериала в условиях его реставрации, мастеринга, сведения и синтеза.

Библиотека, являясь неотъемлемой частью среды NyquistIDE, служит важным технологическим элементом в области обработки звука и может быть посредником среди других периферийных библиотек, поддерживающих устойчивый баланс технико-технологических средств в отношениях NyquistIDE — Audacity®. Устойчивый баланс таких средств зачастую бывает, необходим, когда оператору технологического процесса (оператор обработки аудиоматериала) нужно выполнять обработку аудиоматериала программным способом с поддержанием визуализации сигналограммы через интерфейс аудиоредактора. На программном уровне это может быть NyquistIDE (библиотека «Statistics»), который для визуализации процессов обработки аудиоданных может задействовать оболочку интерфейса редактора Audacity® (NyquistPrompt). При проведении операций со статистическим анализом, такая связка может обеспечить интерфейсно-ориентированный учёт статистических данных, что в свою очередь поможет оператору наблюдать изменения линейной корреляции, внося изменения в событийный ход исполнения сценария, прописанного на языке Nyquist. Такой подход существенно повысит качество предобработки аудиоматериала, поскольку учитываемые показатели корреляции можно использовать в любых технических целях и производить с ними математические операции сколько угодно раз.

Несомненно, проведённый в статье технический анализ кода библиотеки «Statistics» органично дополнит перечень исследований, затрагивающих точную обработку аудиоматериала и привлечёт внимание других исследователей, в сфере которых лежат интересы научного анализа и обоснования применения программных средств, к проблемно-ориентированным областям редактирования аудиоматериала. Авторский анализ, проведённый в данной статье, также может послужить хорошей базой при проведении практических процедур, направленных на обработку и совершенствование аудиоматериала.

ЛИТЕРАТУРА

1. Компьютерная программа Nyquist IDE v.3.15 / Файл директории (statistics.lsp) // Полная реализация — Jesse Clark, David Hovard, David Movatt, David Deangelis, Roger B. Dannenberg.— 2002–2018. [электронный источник, компьютерная программа].
2. Таран В.В. Проектирование дизайна аудиопродукции в программной среде Audacity® с применением языка Nyquist // Современная наука: актуальные проблемы теории и практики. Серия Естественные и технические науки — 2019.— № 10.— С. 159–171. [ISSN2223–2966].

[1] **Прим.автора.** Пьер-Симон, маркиз де Лаплас (Pierre-Simon, marquis de Laplace) — французский учёный и энциклопедист, чья работа имела важное значение для развития инженерии, статистики, математики, физики, астрономии и философии. Он обобщил и расширил работу своих предшественников в пятитомной рукописи «Небесная механика», 1799–1825 (фр. Mécanique céleste, англ.Celestial Mechanics). Эта работа перевела геометрическое изучение классической механики в область, основанную на математическом исчислении, открыв более широкий круг проблем. В статистике байесовская интерпретация вероятности была разработана главным образом Лапласом.

3. Таран В.В. Язык программирования Nyquist: настоящее время и перспективы его развития в области компьютерной аудиоинженерии и аудиоинформатики // Современная наука актуальные проблемы теории и практики: Серия Естественные и технические науки — 2020. — № 4. — С. 135–153. [ISSN2223–2966]. (DOI 10.37882/2223–2966.2020.04.37).
4. Dannenberg R.B. Nyquist Reference Manual Version 3.16 // Carnegie Mellon University — School of Computer Science/ Pittsburgh, PA 15213, U.S.A. 2013–2020 WEB-version, URL: http://www.cs.cmu.edu/~rbd/doc/nyquist/ [электронный ресурс, дата обращения к электронному ресурсу: 03.03.2022].
5. Dannenberg R.B. Nyquist Reference Manual Version 3.15 // Carnegie Mellon University — School of Computer Science/ Pittsburgh, PA 15213, U.S.A. 11.08. 2018 p.276.
6. Dannenberg R.B. Nyquist Reference Manual Version 3.09 // Carnegie Mellon University — School of Computer Science/ Pittsburgh, PA 15213, U.S.A. 27.12. 2014 p.297.
7. Dannenberg R.B. Nyquist Reference Manual Version 2.36 // Carnegie Mellon University — School of Computer Science/ Pittsburgh, PA 15213, U.S.A. 05.03. 2007 p.205.
8. Taube H.K. SAL: A simple algorithmic language in common music, Paper presented at International Computer Music Conference, ICMC2007, Copenhagen, Denmark, 27.08.07–8/31/07 pp. 121–124.
9. Touretzky, David S. Common LISP: a gentle introduction to symbolic computation /Carnegie Mellon University///Copyright (c) 1990 by Symbolic Technology, Ltd.//// Published by The Benjamin/Cummings Publishing Company, Inc.p.587 (ISBN0–8053–0492–4).
10. Godøy R.I. (2009) Chunking Sound for Musical Analysis. In: Ystad S., Kronland-Martinet R., Jensen K. (eds) Computer Music Modeling and Retrieval. Genesis of Meaning in Sound and Music. CMMR2008. Lecture Notes in Computer Science, vol. 5493. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978–3–642–02518–1_4 [электронный ресурс, дата обращения к электронному ресурсу: 03.03.2022].

Российская Академия Наук