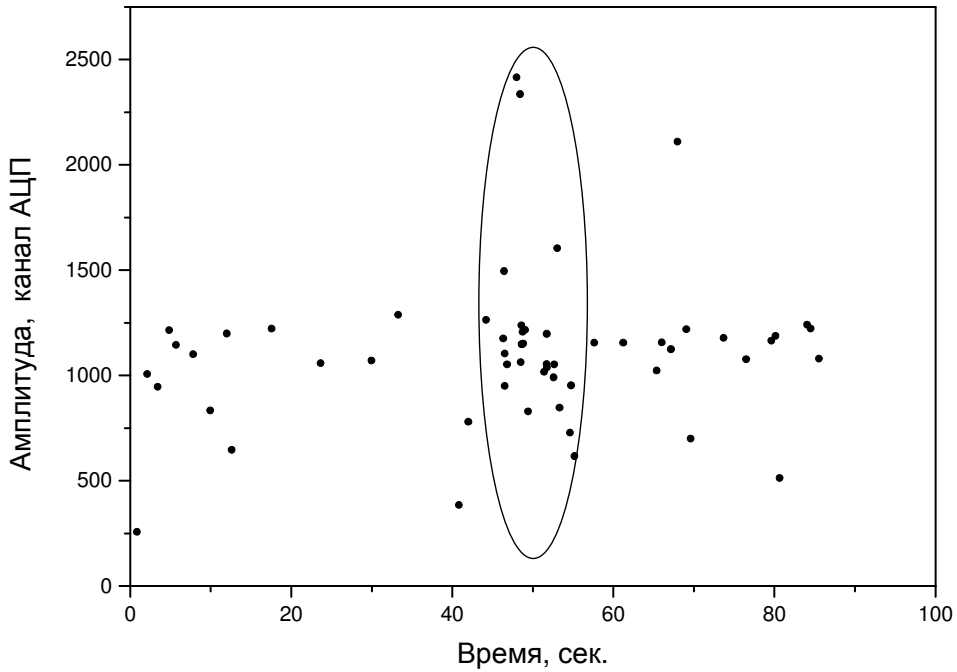
Поиск и отбор нерегулярного фона в событиях с постоянной скоростью счета.

Семинар ОЭФ

Александр Нозик

Постановка задачи

Задача отбора «пачек» возникла при обработке данных установки «Троицк ню-масс». Группы импульсов возникают при запирании электрона в спектрометре и дают нерегулярный вклад в фон.



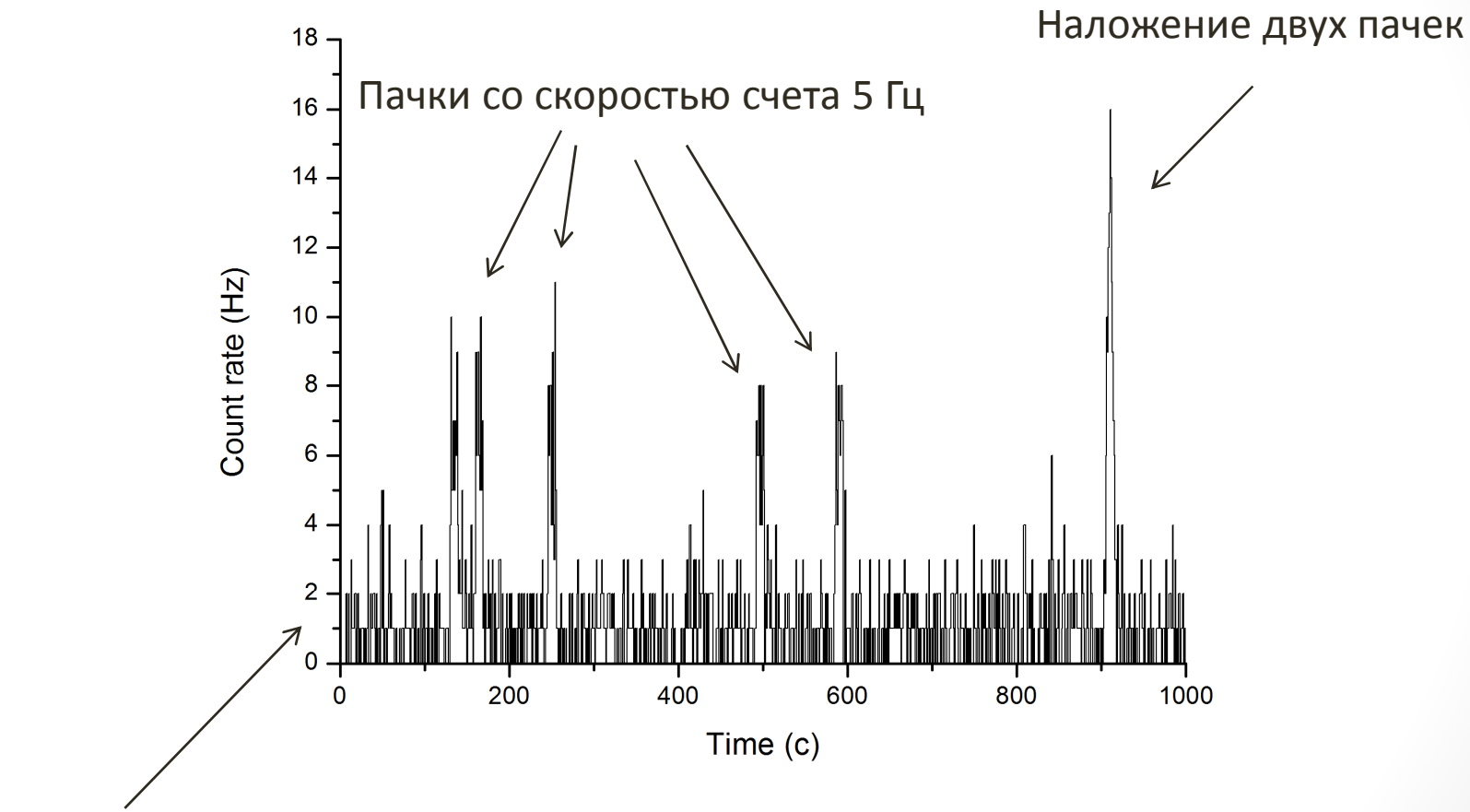
Из диссертации С. В. Задорожного

Постановка задачи

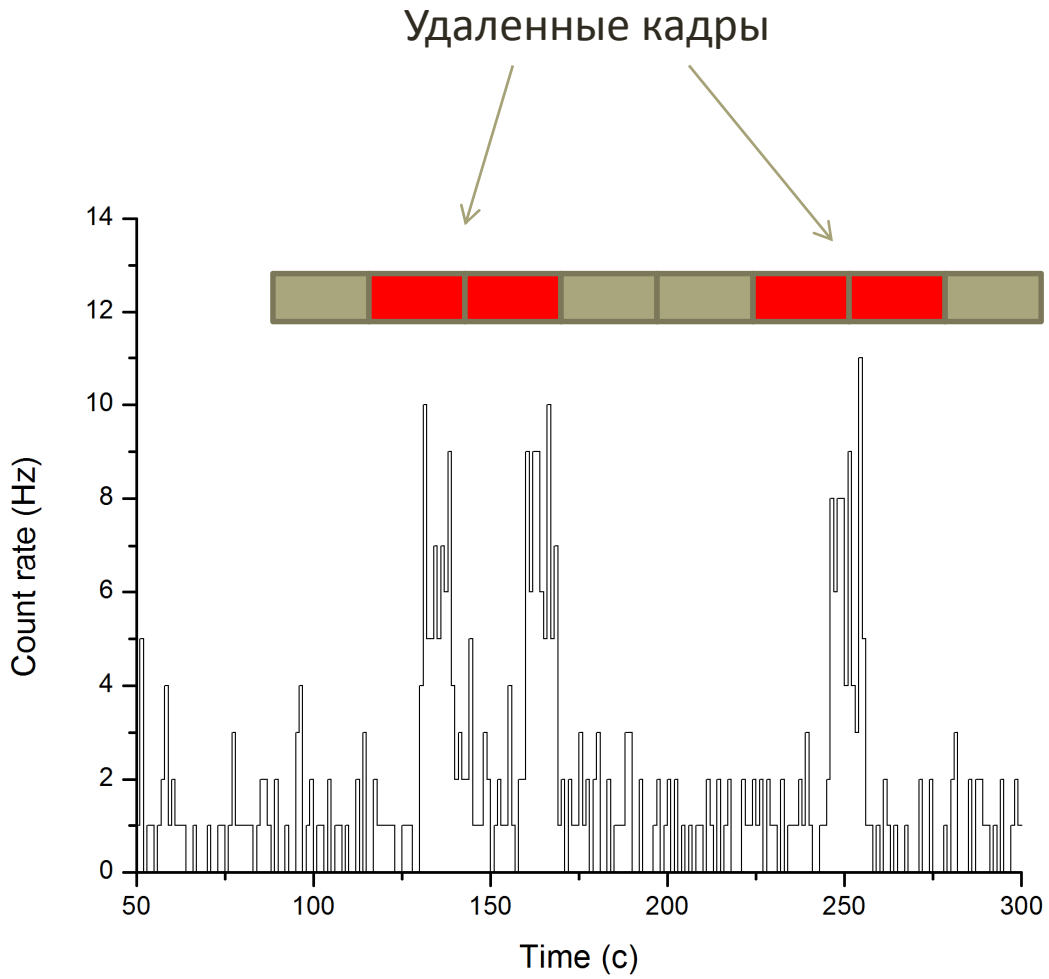
- Проблема нерегулярного фона особенно существенная при измерении спектра с неравными временами набора различных точек.
- Эффективность отбора пачек не оказывает существенного влияния на оценку массы активного нейтрино, но может влиять на определение параметра смешивания тяжелой компоненты.
- В течение долго времени при обработке данных «Троицк ню-масс» использовался алгоритм отбора пачек, изложенный в диссертации С. В. Задорожного. Этот алгоритм очевидно обладает высокой эффективностью, но эта эффективность никогда не была точно измерена.
- Алгоритмы по отбору нерегулярного фона очевидно могут быть использованы в более широких диапазонах входных параметров, чем это было сделано в эксперименте «Троицк ню-масс». Для этого требуется при помощи моделирования определить границы применимости этих алгоритмов.

Все дальнейшие рассуждения не связаны непосредственно с обработкой данных «Троицк ню-масс» и делаются относительно абстрактной модели.

Постановка задачи



Покадровый подход



Отбор кадров

Средняя скорость счета

$$P(n|\nu)=\frac{(\nu\Delta t)^n}{n!}e^{-\nu\Delta t}$$

Критерий отбора: $\sum_{n=\mu\Delta t}^{\infty} P(n|v)<\lambda$

Маленькая хитрость: если брать последовательные кадры, то есть шанс потерять пачку, которая стоит на разделе кадров. Чтобы решить эту проблему берем пересекающиеся кадры с шагом d<<Δt. Количества отсчетов в последовательных кадрах в этом случае является зависимыми, но в отсутствии пачек все равно распределены по Пуассону.

Простой покадровый алгоритм

- Выделяем кадр шириной Δt.
- Проверяем, превосходит ли скорость счета в кадре пороговое значение λ. Если превосходит, то вырезаем кадр.
- Сдвигаем границы кадра на величину шага *d.*
- Повторяем процедуру. Когда вырезаются два последовательных кадра, надо сделать так, чтобы один и тот же участок не вырезался повторно. Для этого можно во всех кадрах до конца пачки вырезать только разницу шириной *d* между текущим кадром и предыдущим.

Недостатки:

- Сильная зависимость эффективности от выбора длины кадра
- При узкой пачке вырезается много полезных событий

Адаптивный алгоритм

- Выделяем кадр шириной Δt.
- Проверяем, превосходит ли скорость счета в кадре пороговое значение для срабатывания λ. Если срабатывания нет, то сдвигаем кадр на *d* и повторяем все с начала.
- В случае срабатывания, двигаем кадр, не изменяя его длину шагами по *d*, пока скорость счета продолжает увеличиваться.
- Начинаем растягивать кадр, отодвигая его конец шагами по *d* до тех пор, пока скорость счета не станет меньше, чем скорость счета в исходном кадре.
- Если скорость счета в кадре превышает порог отбрасывания (вычисленный с учетом новой длины кадра), то вырезаем кадр. Порог отбрасывания в принципе может не совпадать с порогом срабатывания. В этом случае выгодно ставить порог срабатывания достаточно большим (например 10^{-3}) и заставлять программу цепляться за любые подозрительные места, но отбрасывать события только если вероятность меньше более строгого порога (10^{-5} – 10^{-6}).
- Сдвигаем начало нового кадра к концу вырезанного.
- Повторяем процедуру.

Test framework (программа для моделирования)

Эффективность алгоритмов проверялась по следующей схеме:

- Генерировались случайные события, расстояние по времени между которыми моделировалось экспоненциальным распределением с фиксированным параметром, соответствующим скорости счета «нормальных» событий.
- Генерировался набор «пачек», расстояние между которыми также определялось экспоненциальным распределением.
- Для каждой из этих пачек по экспоненциальному распределению с параметром, соответствующим скорости счета в пачке генерировались события и добавлялись к «нормальным» событиям.
- Полученный набор данных прогонялся через алгоритм отбора. Полученная в результате отбора скорость счета сравнивалась с заданной при моделировании. Рассчитывалась величина

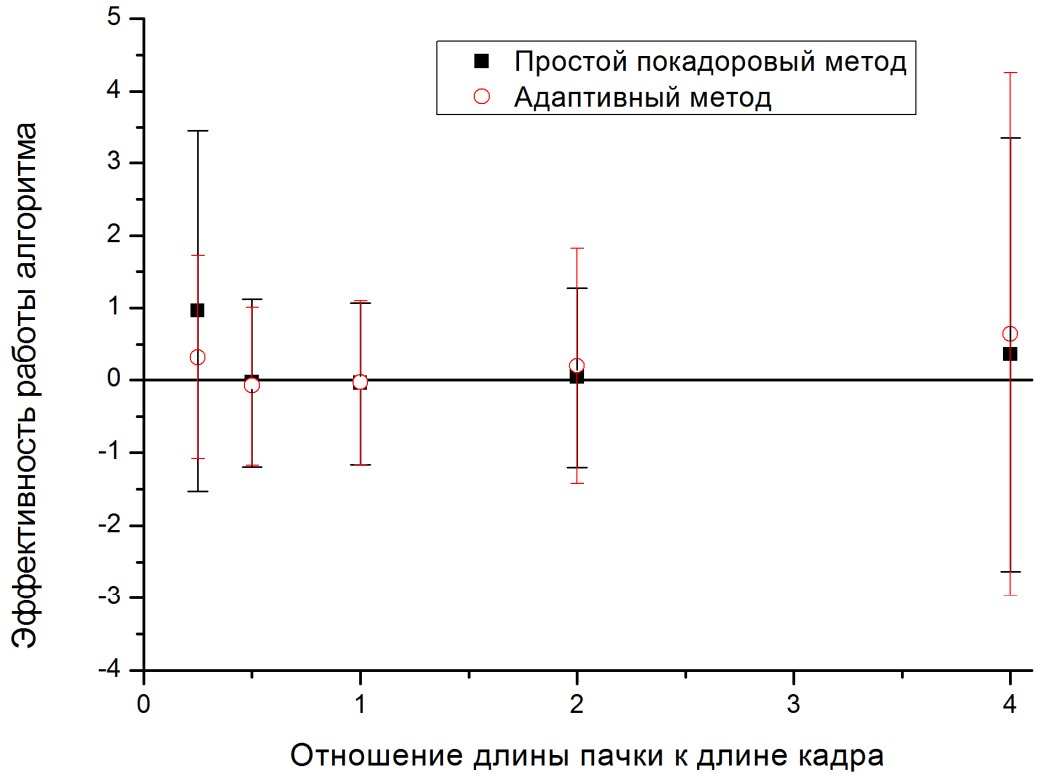
$$eff = (cr - realCr)\sqrt{\frac{T}{realCr}}$$

, где *cr* — скорость счета после очистки, а *realCr* — реальная скорость счета «нормальных» событий, которая закладывалась при моделировании. Эта величина соответствует разнице ожидаемой и полученной скорости счета, выраженной в стандартных отклонениях. Положительные значения эффективности означают то, что алгоритм не смог идентифицировать все фоновые события, а отрицательная — то, что программа вырезаны лишние события.

- Процедура повторяется 1500 раз (параллельные вычисления) для одних и тех же параметров генератора событий. Подсчитывается среднее значение эффективности и его стандартное отклонение. Тут нужно заметить, что при таком определении если бы не было пачек и процедуры их отбора, то в результате чисто статистических флуктуаций среднее значение эффективности должно быть равно нулю, а ее стандартное отклонение единице.

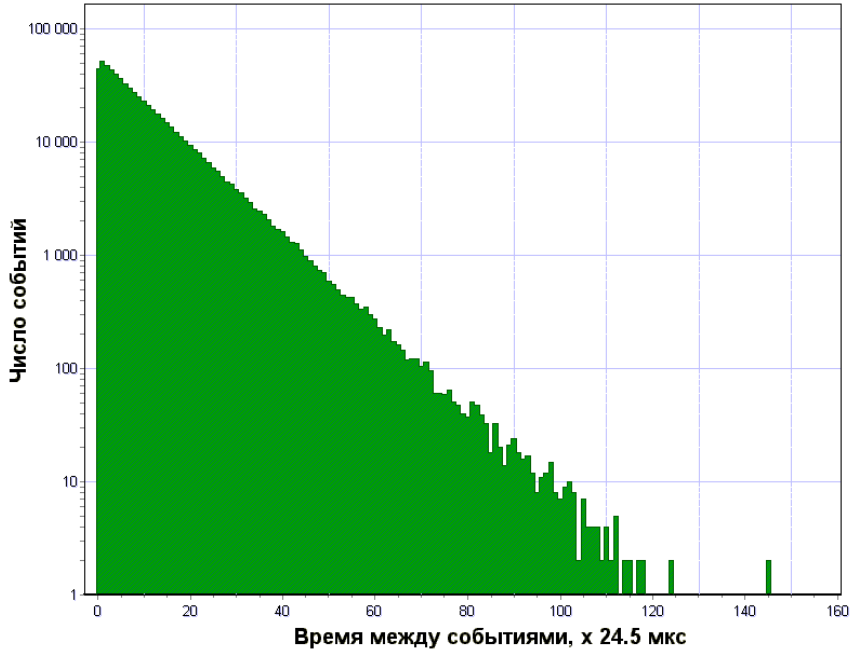
Результаты моделирования

Результаты моделирования



Пособытийный подход

Расстояние по времени между последовательными событиями в отсутствие пачек подчиняется экспоненциальному распределению.



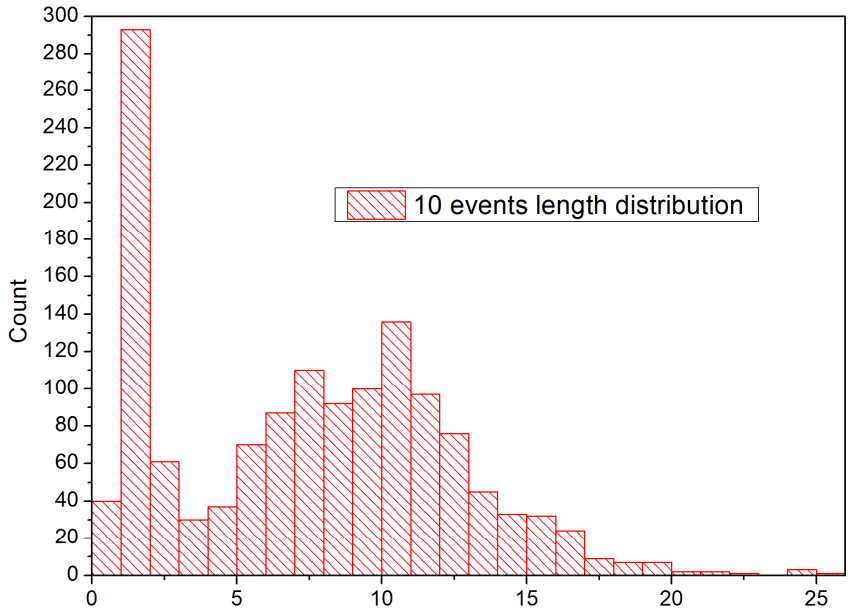
Из диссертации С. В. Задорожного

Пособытийный подход

Проблема: распределение имеет максимум в нуле; сложно выделить аномальную часть.

Решение: берем распределение времени нескольких событий вместо одного.

$$f(t) = \frac{\alpha}{n!}(\alpha t)^n e^{-\alpha t}$$



Такие картинки не позволяют осуществлять количественный отбор фона, но дают качественную информацию о его наличии.

Выводы

- Существует возможность высокоэффективного отбора нерегулярного фона при постоянной скорости счета событий.
- Используя результаты моделирования, можно легко определить ожидаемое смещение среднего и поправку к статистической ошибке.
- В результате сравнения простого покадрового и адаптивного алгоритма можно сделать вывод о том, что в большинстве случаев простой покадровый алгоритм дает достаточную эффективность.

БОНУС!

Быстрый способ решать неравенство

$$\sum_{n=x}^{\infty} \frac{(\mu)^n}{n!} e^{-\mu} < \alpha$$

```
static int getPoissonThreshold(double mean, double prob) {
double pdf = FastMath.exp(-mean); //Функция плотности вероятности на данном шаге
double cdf = pdf; //Функция распределения, вычисляемая как сумма плотностей вероятности на предыдущих шагах
/*
Находим точку "обнуления" распределения и значения кумулятивной плотности в этой точке. Константа POISSON_THRESHOLD отражает с какой максимальной точностью должен быть вычислен порог. Ее стоит брать заведомо маленькой, например 10^-40.
*/
int k = 0;
while (pdf > POISSON_THRESHOLD) {
k++;
pdf *= mean / k;
cdf += pdf;
}
/*
На этом этапе мы нашли наименьший из интересующих нас членов Пуассоновской последовательности. Начинаем считать кумулятивную плотность в обратном порядке.
*/
cdf = 1 - cdf; // Здесь учитывается, что полный интеграл распределения Пуассона равен единице.

while (cdf < prob) {
k--;
pdf *= k / mean;
cdf += pdf;
}
return k; //Решение неравенства
}
```