УДК 004.421

ОБ ОСОБЕННОСТЯХ РЕАЛИЗАЦИИ ЦИФРОВЫХ ЛИНЕЙНЫХ ФИЛЬТРОВ НА ПЛАТФОРМАХ .NET FRAMEWORK И .NET CORE

© 2019. *Т.В. Шарий*

В статье производится исследование способов программной реализации цифровых линейных фильтров в среде .NET. Описаны алгоритмы и программный код фильтров с конечной и бесконечной импульсной характеристикой. Фильтрация осуществляется как в частотной, так и временной областях. Приведены результаты сравнительного анализа скорости работы фильтров на платформах .NET Framework 4.7 и .NET Core 2.1, а также рекомендации по выбору подходящей реализации.

Ключевые слова: .NET; C#; цифровой фильтр; импульсная характеристика; разностные уравнения.

Введение. На протяжении нескольких последних лет в сфере информационных технологий активно разрабатываются специализированные программные инструменты и библиотеки для научных исследований и практического применения в новаторских проектах. Основная часть таких инструментов направлена на исследования в области машинного обучения и цифровой обработки сигналов (звука, изображений, видео, сейсмических сигналов и т.д.) [1-4]. Это обусловлено тем, что подавляющее большинство современных передовых научно-технических разработок связано с данными направлениями. Наиболее популярными языками программирования для прототипирования и визуализации научных моделей являются Python и R, а с точки зрения практической реализации, самыми эффективными остаются языки C и C++ с возможным добавлением параллельных вычислений CUDA, т.к. эти технологии предоставляют самые большие возможности для оптимизации производительности.

В свою очередь, в последнее время активно развивается среда .NET. На данный момент в ней присутствует богатая экосистема, обладающая рядом достоинств, таких как: кросс-платформенность, надежность, достаточно высокая скорость, удобство создания графических интерфейсов настольных, мобильных и web-приложений, широкие возможности Azure для облачных вычислений. Тем не менее, на сегодняшний день существует не так много .NET-библиотек с отдельными функциями цифровой обработки сигналов, а универсальных решений, аналогичных, к примеру, пакету sciPy.signal или MATLAB Signal Processing Toolbox [5], практически нет. Функции, связанные с цифровой фильтрацией, предоставляют библиотеки Math.NET Filtering [6], Bonsai [7], DspSharp [8].

Таким образом, задача эффективной реализации цифровых линейных фильтров на современных версиях платформ .NET является актуальной. Анализ источников выявил, что подробные исследования по данной проблеме еще не проводились. Следует также отметить, что .NET-решение гарантированно не превзойдет по эффективности решение с CUDA-распараллеливанием, но вопрос уменьшения времени выполнения цифровой фильтрации средствами .NET, тем не менее, остается важным.

Постановка задачи. Целью работы является программная реализация, анализ эффективности и ускорение цифровой фильтрации сигналов на платформах .NET Framework и .NET Core. В статье исследуется наиболее используемый и теоретически проработанный класс фильтров: линейные инвариантные к сдвигу цифровые фильтры (Linear Time-Invariant filters, LTI filters). Фильтрация производится как во временной, так и частотной областях.

Линейные цифровые фильтры. Линейными называются фильтры, для которых соблюдается принцип суперпозиции: линейная комбинация входных цифровых сигналов преобразуется в идентичную линейную комбинацию выходных сигналов. Также добавляется свойство инвариантности к сдвигу: задержка входного сигнала во времени приводит к аналогичной задержке выходного сигнала [2]. Линейные фильтры характеризуются уникальной передаточной функцией

$$H(z)=\frac{\sum_{k=0}^{N} b_k z^{-k}}{1+\sum_{m=1}^{M} a_m z^{-m}}=\frac{b_0+b_1 z^{-1}+\ldots+b_N z^{-N}}{1+a_1 z^{-1}+\ldots+a_M z^{-M}},$$

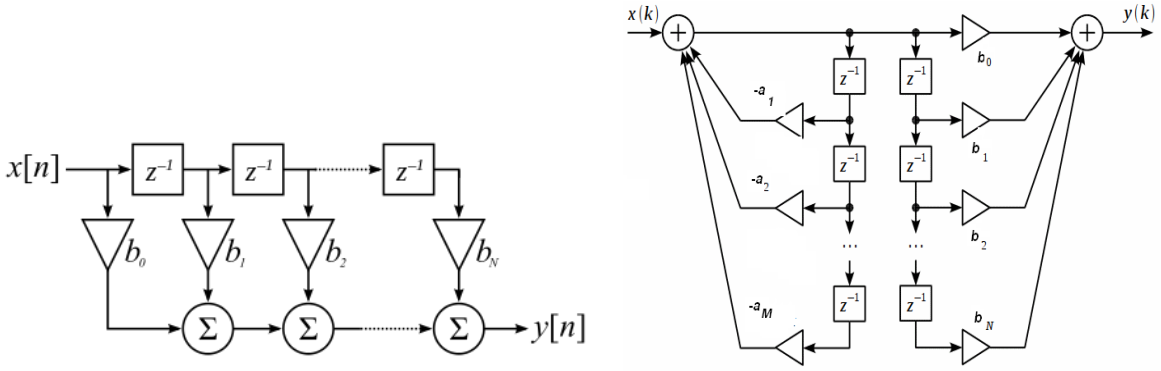
где *N+1* – количество нерекурсивных коэффициентов фильтра; *M+1* – количество коэффициентов обратной связи (рекурсивной части) фильтра.

Во временной области линейный фильтр описывается разностным уравнением

$$y[n]=\sum_{k=0}^{N} b_k x[n-k]-\sum_{m=1}^{M} a_m y[n-m], \qquad (1)$$

где *n* – индекс текущего отсчета сигнала; *x* и *y* – входной и выходной сигналы, соответственно.

В общем случае число *M* не равно 0, в связи с чем, импульсная характеристика фильтра является бесконечной (БИХ-фильтр). Нерекурсивные фильтры (без обратной связи в уравнении (1)) имеют конечную импульсную характеристику (КИХ-фильтры). На рис.1 показаны схемы реализации КИХ- и БИХ-фильтров.



а) б)

Рисунок 1 – Схема реализации линейного цифрового фильтра:
а) КИХ-фильтр; б) БИХ-фильтр

В частотной области линейная фильтрация производится с помощью секционной свертки на основе быстрого преобразования Фурье (БПФ) [3]. Для данной процедуры применяются два традиционных алгоритма: перекрытие с суммированием и перекрытие с сохранением. В работе реализован второй алгоритм, несколько выигрывающий по скорости у первого, т.к. он не требует дополнительных арифметических операций. Последовательность шагов следующая: 1) выбирается размер секции *N*. Это число должно быть не меньше размера ядра КИХ-фильтра *L* (обычно *N>4L*) и степенью

двойки; 2) вычисляется N-точечное комплексное БПФ ядра свертки, дополненного нулями справа; 3) входной сигнал разбивается на перекрывающиеся фрагменты (секции) размером N отсчетов с величиной сдвига в $N–L+1$ отсчетов; 4) выполняется N-точечное комплексное БПФ каждой секции, при этом каждая секция предваряется $L–1$ отсчетами, сохраненными в конце предыдущей секции; 5) результат БПФ ядра комплексно умножается на результат БПФ секции; 6) вычисляется обратное БПФ полученного произведения; 7) полученный блок добавляется к уже отфильтрованному сигналу; последние L–1 отсчетов сохраняются для следующей секции.

Технически секционную свертку можно применять и к БИХ-фильтрам, для чего необходимо обрезать импульсную характеристику до конечного размера N, но результат фильтрации, очевидно, будет отличаться от конечно-разностной схемы.

Алгоритмы и программная реализация фильтров. Существует два возможных способа обработки сигналов с помощью конечно-разностных уравнений вида (1). Первый способ предполагает, что входной сигнал доступен полностью и размещен в непрерывной области памяти в виде массива. В этом случае фильтрация производится в режиме «оффлайн» и программно реализуется напрямую с помощью цикла по всем отсчетам входного сигнала, без дополнительных структур данных. Однако в большинстве случаев данные поступают на обработку последовательно в режиме реального времени, и каждый выходной отсчет вычисляется при поступлении очередного входного отсчета. Такой процесс еще называется онлайн-фильтрацией. В этом случае для корректной реализации уравнения (1) входные данные накапливаются в линии задержки, которая обычно программируется с помощью циклической очереди. Ниже приводится код типового решения на языке C# для КИХ-фильтра:

```
float Process(float sample)
{
    float output = 0;
    delayLine[offset] = sample;
    int pos = 0;
    for (int k = offset; k < size; k++)
    {
        output += b[pos++] * delayLine[k];
    }
    for (int k = 0; k < offset; k++)
    {
        output += b[pos++] * delayLine[k];
    }
    if (--offset < 0) offset = size - 1;
    return output;
}
```

Листинг 1 – Код реализации работы КИХ-фильтра (версия 1)

В листинге 1 используются переменные size (размер ядра КИХ-фильтра, т.е. количество коэффициентов b), delayLine (массив линии задержки размера size) и offset (позиция в линии задержки для текущего отсчета; при обработке очередного нового отсчета входного сигнала offset смещается на 1 влево; при достижении нулевого индекса offset устанавливается в конец очереди). Эти переменные являются полями класса FirFilter, одним из методов которого является и функция Process.

Временная сложность функции Process – O(K), где K – размер ядра фильтра. Соответственно, временная сложность алгоритма обработки N отсчетов – O(NK). В коде листинга 1 проход по циклическому буферу линии задержки начинается с отсчета по индексу offset до конца массива и затем с нулевого отсчета до индекса offset – 1. Это

в определенной степени замедляет фильтрацию, в силу особенностей работы кэш-памяти процессора. Поэтому в работе для сравнения была реализована вторая версия метода Process. В этой версии вопрос непрерывности прохода по массиву решается путем дублирования коэффициентов *b*:

```
// В конструкторе класса сначала создается «удвоенный» массив b:
// [b0, b1, … bN]   ->   [b0, b1, b2, … bN, b0, b1, b2, … bN]

float Process(float sample)
{
    delayLine[offset] = sample;
    float output = 0;
    for (int i = 0, j = size - offset; i < size; i++, j++)
    {
          output += delayLine[i] * b[j];
    }
    if (--offset < 0) offset = size - 1;
    return output;
}
```

Листинг 2 – Код реализации работы КИХ-фильтра (версия 2)

Версия листинга 2 требует вдвое больше памяти для коэффициентов фильтра, имеет ту же временную сложность O(*NK*), что и первая версия, однако учитывает нюансы архитектуры современных процессоров.

Для БИХ-фильтров были запрограммированы также две версии с аналогичными отличиями (класс IirFilter).

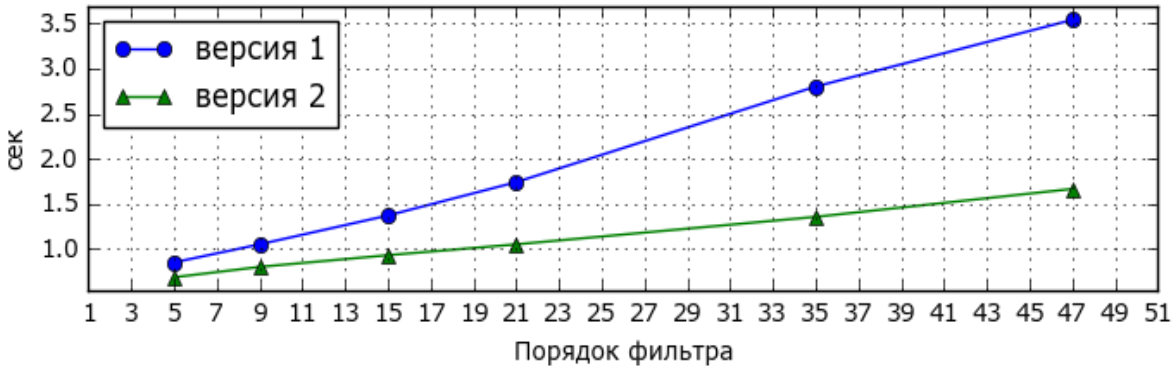
Кроме того, к экспериментам были подготовлены версии кода, работающие с вещественными числами двойной точности (double, 64 бит) вместо одинарной точности (float, 32 бит). Современные компиляторы и процессоры уже практически не имеют проблем со скоростью операций над 64-битными вещественными числами (что и подтвердили дальнейшие эксперименты), поэтому зачастую выбор определяется из соображений более эффективного использования оперативной памяти или прямой необходимости в двойной точности вычислений.

Среда .NET. Перед проведением сравнительного анализа скорости программных реализаций фильтров необходимо иметь представление о спецификации .NET Standard. Данная спецификация определяет базовый набор программных интерфейсов, который должна реализовать любая .NET платформа, без конкретных деталей реализации и без привязки к операционным системам. Этот базовый набор называется BCL (Base Class Library, библиотека базовых классов) [9].

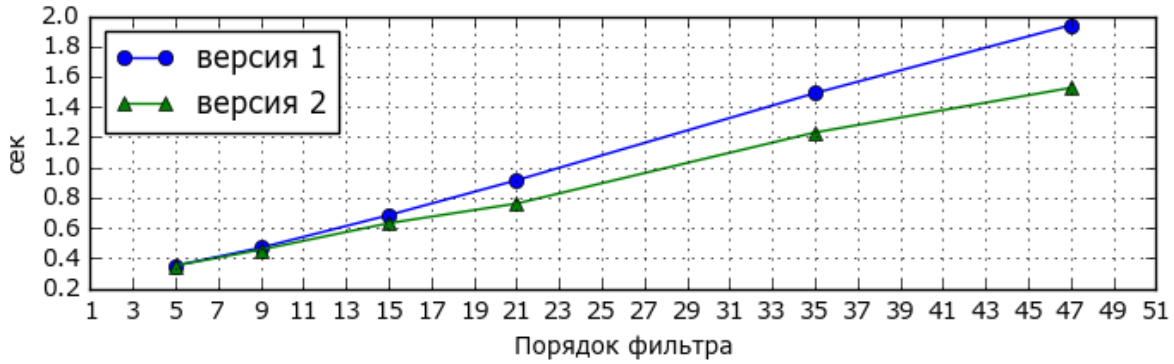
Исторически первой и по-прежнему очень популярной является платформа .NET Framework (текущая версия имеет номер 4.7). Ее ключевым элементом является общеязыковая среда выполнения (Common Language Runtime, CLR) [9]. Встроенный в виртуальную машину CLR JIT-компилятор (just-in-time compiler) «на лету» преобразует промежуточный байт-код в машинные коды процессора. Виртуальная машина CLR также обеспечивает базовую безопасность, управление памятью, организацию системы генерации и перехвата исключений.

Платформа .NET Core основана на .NET Framework и появилась относительно недавно. В ней используется отдельная среда выполнения CoreCLR, включающая новый JIT-компилятор RyuJIT, встроенный сборщик мусора и другие компоненты. В связи с этим, можно ожидать, что скорости выполнения одного и того же кода в .NET Framework и .NET Core будут отличаться.

Описание эксперимента и анализ результатов. Эксперимент проводился с аудиоданными в формате PCM WAV со следующими характеристиками: частота дискретизации – 22050 Гц, количество бит на отсчет – 16, количество каналов – 1 (моно сигнал). Замеры времени производились несколько раз и в различном порядке, затем полученные значения усреднялись. Размеры ядра фильтра варьировались в пределах от 5 до 47 отсчетов. Фильтры проектировались низкочастотными, с частотой среза 20% от частоты Найквиста (впрочем, конкретный вид ядра фильтра не играет роли в этих экспериментах). На рис.2 приведены результаты фильтрации сигнала длительностью 20 минут (около 26,5 миллионов отсчетов).



а)



б)

Рисунок 2 – Результаты сравнения эффективности реализаций КИХ-фильтра:
а) .NET Framework 4.7; б) .NET Core 2.1

Эксперименты производились на компьютере со следующими характеристиками: процессор Intel Core i3 с тактовой частотой 3,5 ГГц; объем оперативной памяти 12 Гб.

Как видно из графиков, реализация .NET Core работает значительно быстрее .NET Framework, особенно для малых размеров ядра фильтра. Кроме того, в .NET Core не так ощутим разброс между версиями кода фильтрации. Максимальный прирост 1.2x-1.3x достигается на размерах ядра более 35 коэффициентов. В случае с .NET Framework вторая версия гораздо в большей степени выигрывает по сравнению с первой. Прирост варьируется в пределах от 1.2x до 2.2x с ростом размера ядра фильтра. Эксперименты с БИХ-фильтрами также выявили повышение эффективности для второй версии кода, но прирост оказался ощутимо меньше в обеих платформах .NET. В .NET Core 2.1 разница между версиями почти незаметна. Также, вполне ожидаемо, при размере ядра фильтра

более 60 коэффициентов, конечно-разностная схема реализации фильтрации начинает стремительно уступать схеме, основанной на секционной свертке.

Выводы. Проведенные исследования позволили сравнить показатели скорости цифровой фильтрации различными способами на новейших платформах .NET. Исходя из результатов, можно выработать следующие рекомендации для КИХ-фильтров. Если размер фильтра превышает 55-60 отсчетов, целесообразно применять секционную свертку на любой платформе (стандартное решение). Для меньших порядков фильтров на платформе .NET Framework более предпочтительна вторая версия кода. На платформе .NET Core выбор между двумя версиями кода не так критичен, и на первый план могут выйти вопросы памяти: вторая версия немного быстрее, но требует в два раза больше памяти. В случае с БИХ-фильтрами можно остановиться на первой версии кода, т.к. прирост скорости начинает ощущаться при большом числе коэффициентов, не свойственном для БИХ-фильтров, применяемых на практике.

СПИСОК ЛИТЕРАТУРЫ

[1] Lyons R. Understanding Digital Signal Processing. 3rd Edition / R.Lyons. – Prentice Hall, 2014. – 984 p.
[2] Downey A. Think DSP. Digital Signal Processing in Python / A.Downey. – O'Reilly Media, 2016. – 176 p.
[3] The Scientist and Engineer's Guide to Digital Signal Processing by S.W. Smith [Электронный ресурс]. – Режим доступа: http://www.dspguide.com / 20.08.2019.
[4] Orfanidis S. Introduction to signal processing / S.Orfanidis. – Prentice Hall, 2010. – 783 p.
[5] Signal Processing Toolbox [Электронный ресурс]. – Режим доступа: https://www.mathworks.com/products/signal.html / 20.08.2019.
[6] Math.NET Filtering [Электронный ресурс]. – Режим доступа: https://filtering.mathdotnet.com / 20.08.2019.
[7] Bonsai. A visual programming language for reactive systems built on top of Rx for .NET [Электронный ресурс]. – Режим доступа: https://bitbucket.org/horizongir/bonsai / 20.08.2019.
[8] DspSharp. API for Digital Signal Processing in C# [Электронный ресурс]. – Режим доступа: https://github.com/Jonarw/DspSharp / 20.08.2019.
[9] Common Language Runtime (CLR) Overview [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/en-us/dotnet/standard/clr / 20.08.2019.

Поступила в редакцию 01.09.2019 г.

ON SPECIFICS OF DIGITAL LINEAR FILTERS REALIZATION ON .NET FRAMEWORK AND .NET CORE PLATFORMS

T.V. Sharii

The article provides research of program implementation of digital linear filters in .NET environment. The algorithms and program code of finite and infinite impulse response filters are described. Filtering is carried out both in time and frequency domain. The results of comparative analysis of filtering performance on .NET Framework 4.7 and .NET Core 2.1 platforms are given, as well as recommendations concerning the choice of appropriate realization.

Keywords: *.NET; C#; digital filter; impulse response; difference equations.*