Лекция 0: Цифровая фильтрация и базовые понятия языка программирования Си

Д. А. Караваев

Санкт-Петербургский государственный университет телекоммуникаций им. проф. М. А. Бонч-Бруевича

Факультет РТС, Кафедра РОС

Факультатив «Программирование в ЦОС»

Осень 2019

07.10.2019 Санкт-Петербург

Необходимые/приобретаемые знания

- **Язык программирования Си**:
  Выразительный и, до сих пор, один из наиболее востребованных и популярных языков программирования (https://www.tiobe.com/tiobe-index/, http://pypl.github.io/PYPL.html);
- **Цифровая обработка сигналов (ЦОС)**:
  Акцент будет сделан на задачах, возникающих в приложениях ЦОС и радиотехнике.
- **Алгоритмы**:
  Будут представлены некоторые классические алгоритмы и структуры данных с примерами их приложений;
- **Операционные системы семейства GNU/Linux**:
  Наиболее приспособлены для разработки и сопровождения ПО, часто встречаются во встроенной разработке (чем мы будем заниматься во второй части курса) и к тому же бесплатные.

Материалы

- Исходные файлы проектов с заданиями и материалы: https://github.com/dkaravaev/bonch-dsp-faculty.
- Книга: «Язык программирования Си: лекции и упражнения», С. Прата;
- Книга: «Теория и применения цифровой обработки сигналов», Л. Рабинер, Б. Гоулд;
- Онлайн-курс «Программирование на языке C++»: https://stepik.org/course/7/promo;
- Онлайн-курс «Digital Signal Processing»: https://www.coursera.org/learn/dsp;
- Дистрибутив GNU/Linux семейства Debian: Ubuntu или Linux Mint.

КИХ-фильтр: алгоритм свёртки

Формулировка

Реализовать цифровой фильтр с конечной импульсной характеристикой (КИХ-фильтр), который задаётся формулой свёртки:

$$y[n] = h[n] * x[n] = \sum_{k=0}^{T-1} h[k]x[n-k], \tag{1}$$

где даны:

- входной сигнал (последовательность) $x[n]$ длины N: $\{x[0], x[1], \ldots, x[N-1]\}$;
- импульсная характеристика (ИХ) $h[n]$ длины T,

и надо найти выходной сигнал $y[n]$ длины N.

Пример

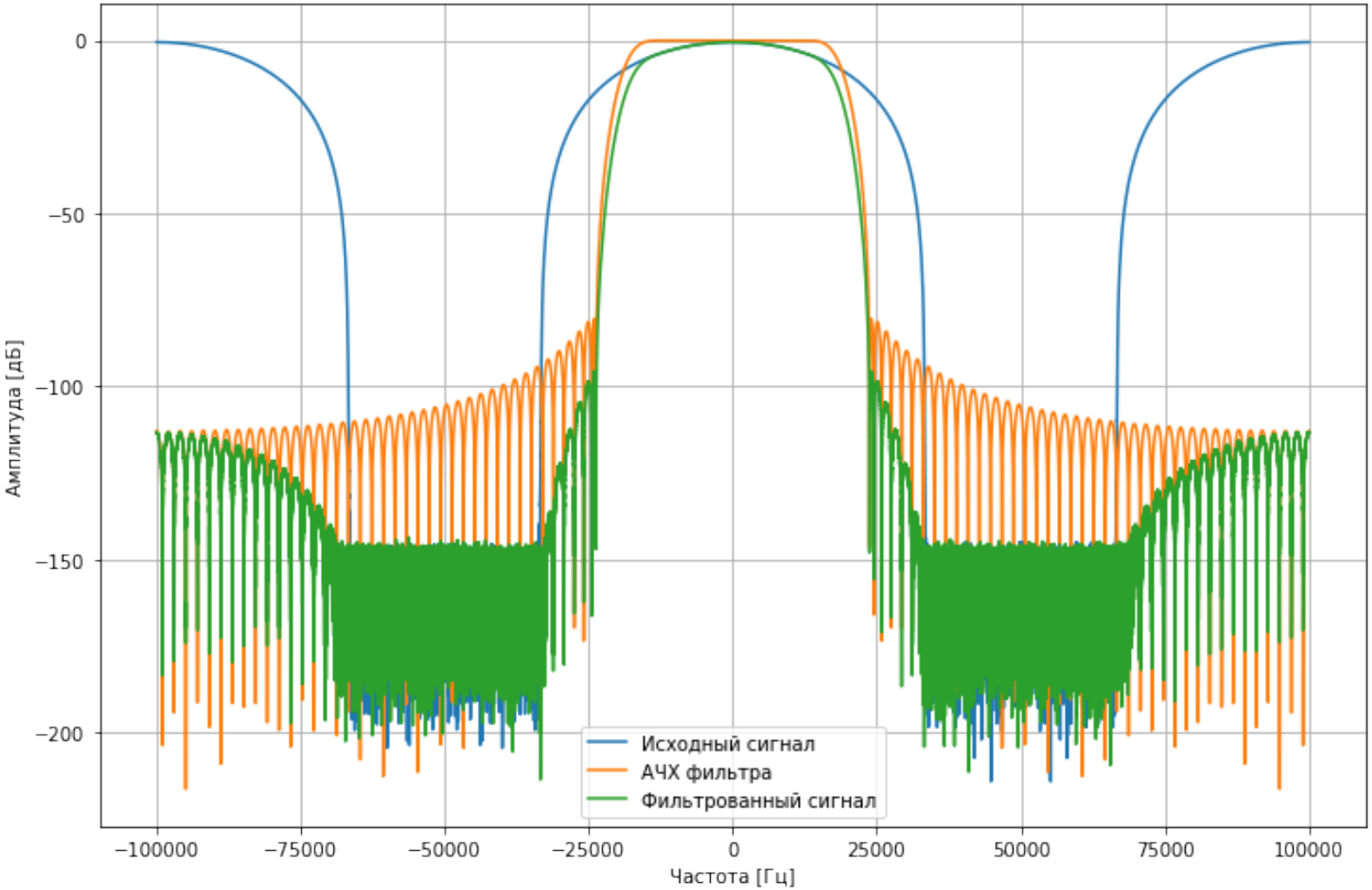

Рис.: Пример амплитудного спектра исходного и фильтрованного широкополосного сигнала

Дополнение нулями

Вычисление свёртки начинается с $x[0] \implies$ необходимо определить $x[n]$ для $n \in \{T-2, \ldots, -1\}$ (иначе сумма в (1) не имеет смысла при $n \in 0, \ldots, T-1$).
Положим: $x[n] = 0,\ n \in \{T-2, \ldots, -1\}$.
Операция называется **дополнением нулями** (*англ.* zero padding), результатом которой будет последовательность длинны $N+T-1$:

$$\hat{x}[n] = \{\hat{x}[0] = 0, \ldots, \hat{x}[T-1] = x[0], \ldots, \hat{x}[N-T-2] = x[N-1]\}.$$

Тогда формула свёртки (1) будет иметь вид:

$$y[n] = h[n] * x[n] = \sum_{k=0}^{T-1} h[k]\hat{x}[n + (T-1) - k], \tag{2}$$

которую можно разложить в два этапа (2.1) и (2.2) (см. далее).

Визуализация дополнения нулями

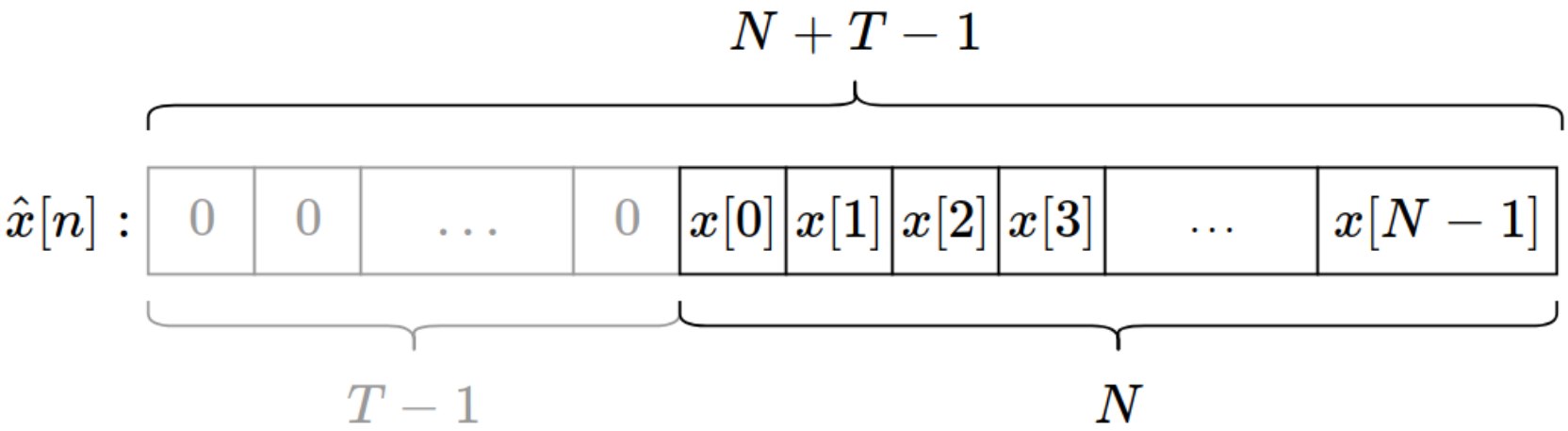

Рис.: Результат дополнения нулями

Визуализация алгоритма свёртки

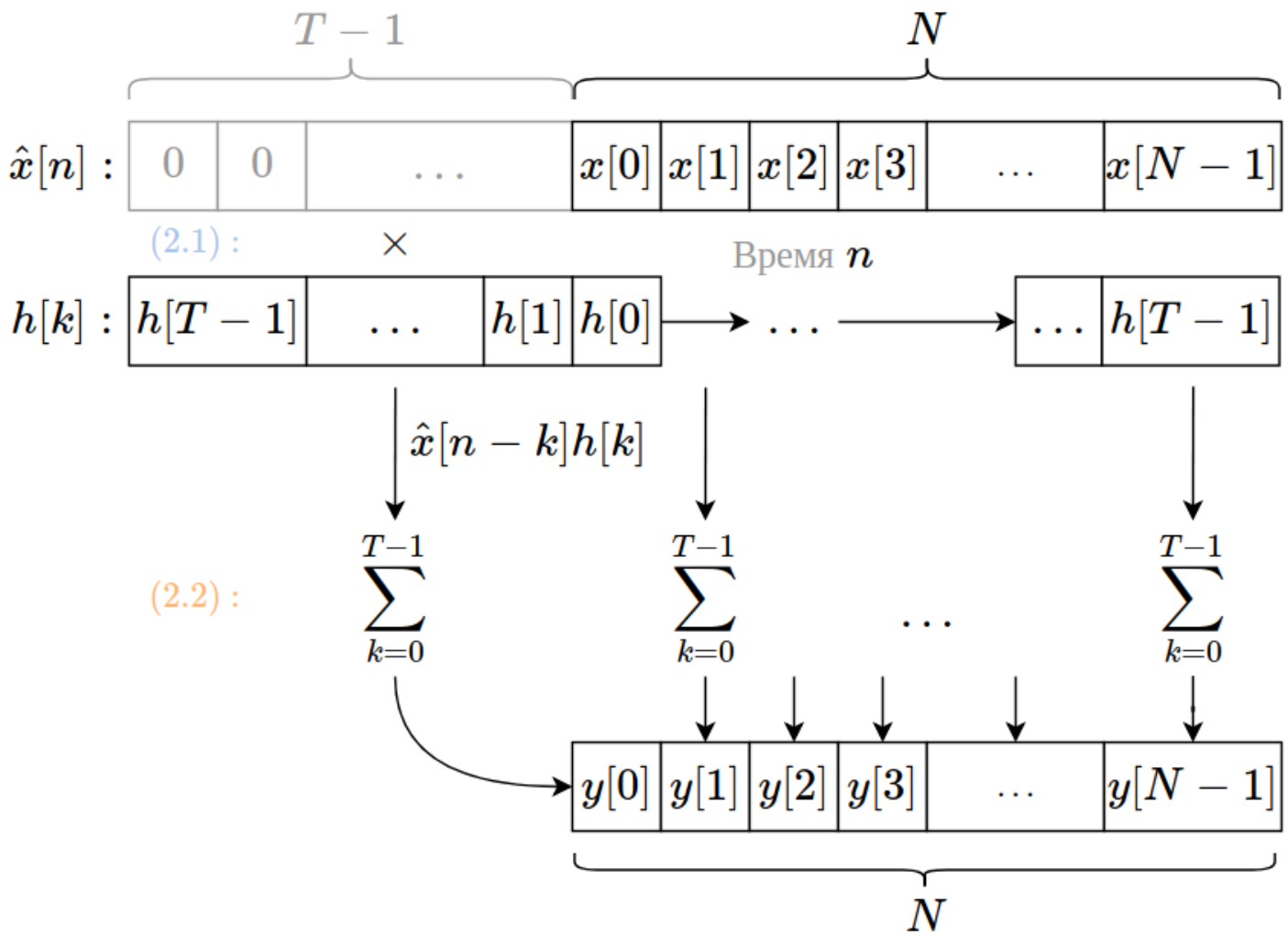

Рис.: Схема алгоритма свёртки

Псевдокод алгоритма свёртки

Вход : $\hat{x}[n]$ - вх. сигнал длины $N+T-1$, $h[n]$ - ИХ длины T.
Выход: $y[n]$ - вых. сигнал длины N.

for $n \leftarrow 0$ **to** $N-1$ **do** // Цикл по времени
  $y[n] \leftarrow 0$ // Инициализация n-ого вых. отсчёта
  $m \leftarrow n + (T-1)$ // Смещение по доб. нулями массиву
  for $k \leftarrow 0$ **to** $T-1$ **do** // Цикл вычисления n-ого вых. отсчёта
    $y[n] \leftarrow y[n] + \hat{x}[m-k]h[k]$ // Умножение с накоплением (MAC)
  end
end

Вывод

Необходимо узнать как в языке Си: *создавать переменные и последовательности, задавать цикл со счётчиком, производить арифметические операции и операции сравнения.*

Типы и переменные в языке Си

```c
/* Это комментарий! */
T variable; /* Объявление переменной с некоторым (псевдо)типом T. */
/* Нас интересуют числовые типы: int, float и т.д. */
float a;    /* Объявление переменной типа float. */
/* Переменным можно (и нужно) присваивать значения (=): */
a = 3.14;   /* Это называется инициализация - первичное
             * присвоение значения. */
int x = 1;  /* Объявление с инициализацией переменной типа int. */
/* Можно (и иногда нужно) использовать модификаторы для типов:
 * unsigned (беззнаковый) и const (неизменяемое значение). */
const unsigned int CONSTANT = 20; /* Константа типа беззнаковый int. */
```

Вывод: Переменная - это область памяти компьютера, к которой можно обращаться для чтения или записи по заданному имени.

Арифметические операции и операции сравнения

```
/* В Си арифметические операции обозначаются стандартно:
 * (-, +, /, *) */
float x = 1.0;
float y = 2.0;
/* Пример для +. Аналогично для: (-, *, /). */
float z = x + y; /* z: 3.0. x и y - операнды (в контексте операции). */
/* Сделать операцию с самим собой в качестве операнда: */
z += x; /* <-> z = z + x, z: 4.0. */
/* Операции сравнения: (==, !=, <, >, <=, >=). */
/* Результат: true (1) или false (0). */
int less = z < x; /* less == false. */
int moreq = z >= y; /* moreq = true. */
/* Приоритет исполнения операций: */
int neq = (2 * z + 1) == (2 * (z + 1)); /* neq == true. */
```

Массивы в языке Си

Для реализации алгоритма свёртки нам нужны переменные, которые бы содержали последовательность значений одного типа (например `float`). Такие переменные называются **массивами**.

```
/* Нумерация (индексация) массивов в Си начинается с 0, а не с 1. */
float x[10]; /* Объявление массива типа float с двумя элементами (длины 2).
x[0] = 0.1; /* Присвоить значение 0-му элементу. */
int i = 9; /* Переменная-индекс. */
x[i] = 0.1 * i; /* Присвоить значение 9-му (последнему) элементу. */
x[i - 5] = x[i] + 3.14; /* Присвоить значение 4-му элементу. */
x[i + 4] = 0.4; /* !Ошибка!: В массиве нет 13-ого элемента. */
/* С элементами массива можно делать все тоже самое, что с
 * обычными переменными, то есть массив это переменная,
 * состоящая из пронумерованной последовательности
 * других переменных (сигнал <-> массив!). */
```

Указатели в языке Си

Подобные массивы не удобны, так как они имеют заранее заданную длину. В нашем случае, длины последовательностей заранее неизвестны. На помощь приходят **указатели**.

```
/* Указатель - это переменная, значение которой - адрес другой переменной
 * в памяти компьютера. */
float* ptr; /* Указатель на переменную типа float.
             * Не инициализирован (указывает в никуда). */
/* Оператор & - возвращает адрес своего аргумента (переменной). */
float y = 0.0;
ptr = &y; /* ptr указывает в адрес y (инициализирован). */
*ptr = 1.12; /* По указателю можно обращаться к переменной
              * с помощью оператора (*), примененного к нему. */
/* Разименованный указатель - синоним переменной, на которую он
 * указывает. После последней операции: y == 1.12. */
```

Указатели и массивы/Арифметика указателей

```
float x[10];
/* Инициализируем все значения x...*/
/* По указателю можно смещаться вправо (+) или влево (-) на n шагов. */
float* begin = &x[0]; /* Указывает на x[0] */
float* mid   = begin + 4; /* Указывает на x[4] */
int n = 3; /* Можно сделать так: */
float* second = mid - n; /* Указывает на x[1] */
/* Более того: */
begin[1] = 0.2; /* x[1] == 0.2 -> begin - это синоним имени массива! */
/* Имя массива - это указатель на его 0-вой элемент. */
```

Таким образом последовательности неизвестной длины мы будем задавать через указатели на их 0-вой элемент и, имея переменную с длиной этой последовательность, мы не будет выходить за её пределы.

Иллюстрация арифметики указателей

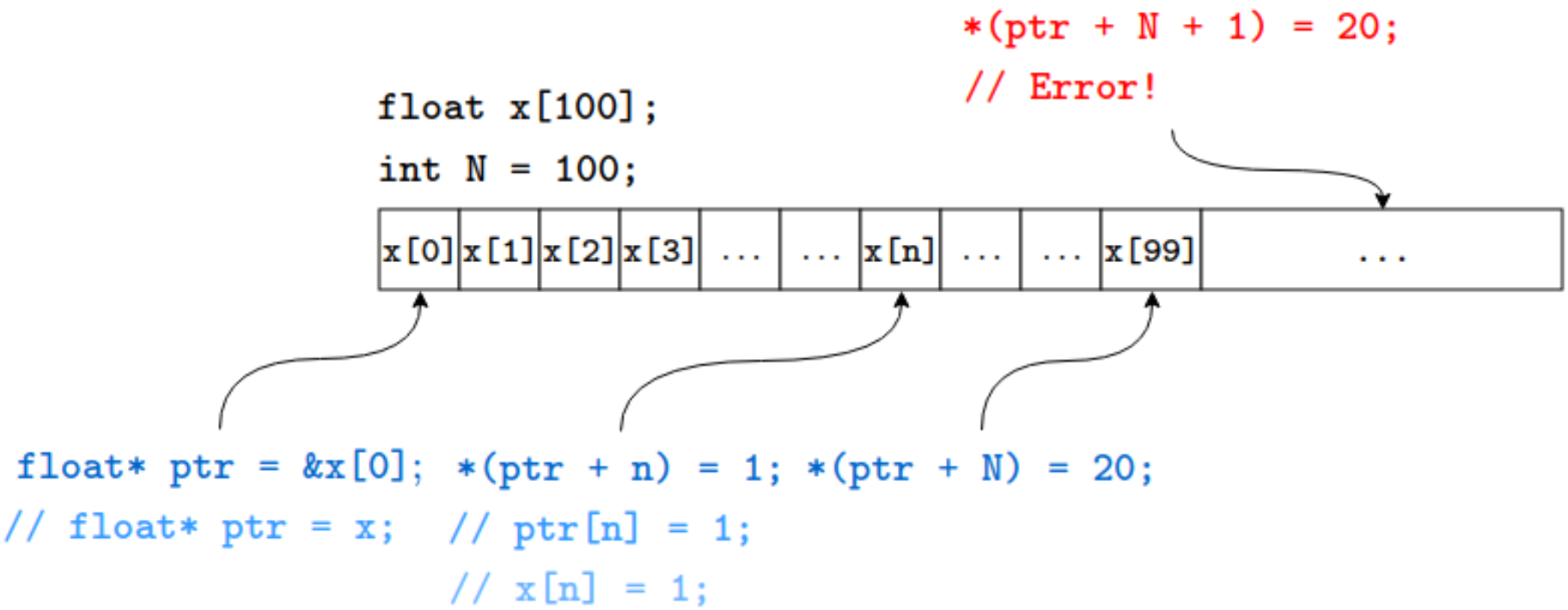

Рис.: Арифметика указателей. Более тусклые цвета в комментариях - синонимичные действия.

Цикл со счётчиком (for)

Для того чтобы вычислить значения $y[n]$ нам нужно использовать цикл со счётчиком, который в языке Си определяется так:

```
float array[10];
int N = 10;
for (int i = 0; i < N; i++)
{
    /* 1. int i = 0; Инициализация счётчика. */
    /* 2. i < N; Ограничение на величину счётчика. */
    /* 3. i++; Инкремент <-> i += 1 <-> i = i + 1. */
    array[i] = 0.1 * i;
}
```

Прототип функции

Алгоритм свёртки будет реализован в теле **функции**. К тонкостям функций в языке Си мы ещё вернёмся. На данном этапе определим *прототип* функции (файл `source/convolve.c`):

```
int DSP_convolve(const float* xh, float* y, size_t N,
                 const float* h, size_t T)
{
    /* N, T - длины последовательностей x[n] и ИХ фильтра соответственно,
     * xh - входная последовательность x[n] дополненная нулями,
     * y - выходная последовательность, h - ИХ фильтра. */
    /* Замечание: Тип size_t - беззнаковый целочисленный тип. */
    /* Замечание: Модификатор const у указателя говорит о том,
     * что значение по его адресу изменить нельзя. */
    /* Далее идёт код для вычисления... ваша задача:) */
    return 0; /* Возвращающие значение функции. Об этом позже. */
}
```

Сборка

После того как код функции был написан, необходимо **собрать** проект, в котором реализована данная функция, то есть преобразовать код на языке Си в файл, который можно будет исполнить. Для этого в папке проекта нужно исполнить следующие команды в терминале:

```
mkdir build # Создать папку для сборки build.
cd build # Войти в папку build.
cmake .. # Настроить сборку.
make # Осуществить сборку.
```

Замечание: В программе могут быть *синтаксические* ошибки, и компилятор (программа, которая осуществляет преобразование кода на языке Си в исполняемый файл) скажет вам, где они.
Замечание: Отсутствие *синтаксических* ошибок не означает отсутствие *семантических* :)!

Запуск и просмотр результата

После того, как проект был собран, то можно запустить полученный исполняемый файл:

```
# В папке build:
./convolve # Запускаем проект.
# Запускаем скрипт на языке Python для просмотра результата:
python3 ../scripts/plot_regular.py
```

Замечание: Мы использовали системы для сборки проектов на языке Си: `CMake` и `Make`. О них поговорим подробнее позже.
Замечание: Входной сигнал считывается из файла, а выходной записывается. На основе этих файлов, скрипт на языке `Python` отображает результат работы алгоритма.

Визуализация результата

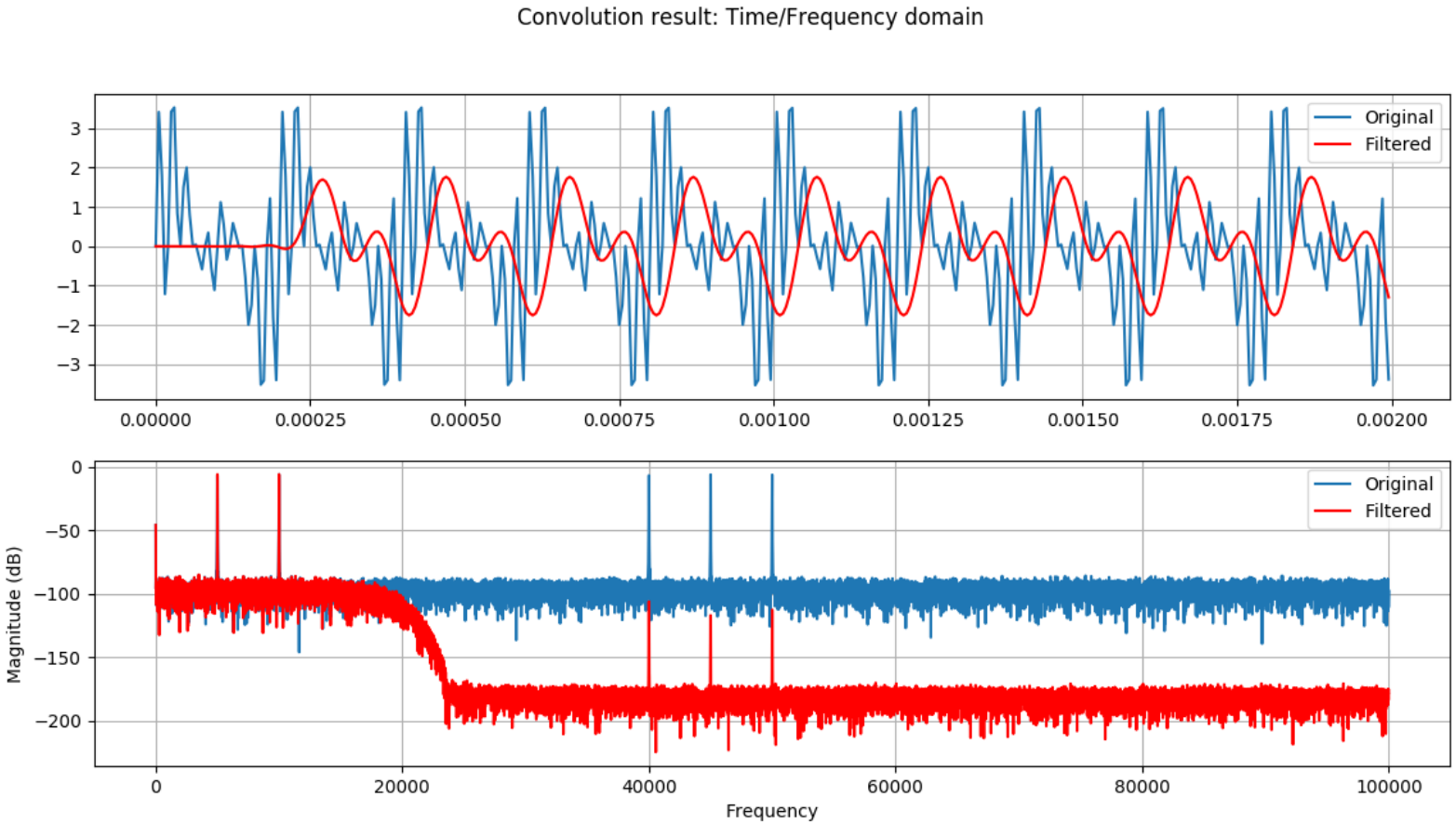

Рис.: Результат фильтрации входного сигнала $x[n]$ (линейная комбинация пяти синусоид)

Свёртка с децимацией*: формулировка

Из рисунка на предыдущем слайде видно, что подавленная фильтром часть полосы больше не представляет интереса $\Longrightarrow$ можно понизить частоту дискретизации $x[n] - F_s$ до перехода к полосе пропускания фильтра.
Для этого берутся только отсчёты $y[nD]$, где D - коэффициент децимации, то есть формула (2) будет иметь вид:

$$y[nD] = h[n] * x[nD] = \sum_{k=0}^{T-1} h[k]\hat{x}[nD - (T-1) - k] \tag{3}$$

Частота дискретизации для $y[nD]$ равна F_s/D!
Задача: *Реализовать алгоритм свёртки с децимацией для входной последовательности* $\hat{x}[n]$.
NB: Если размер входа $x[n]$ равен N, то у выхода $y[nD]$ - N/D!

Свёртка с децимацией*: прототип функции

```
/* Прототип функции для свёртки с децимацией: */
int DSP_convolve_decimate(const float* xh, float* y, size_t N,
                          const float* f, size_t T, size_t D)
{
    /* D - коэффициент децимации. */
    return 0;
}
```

Рис.: Схема свёртки с децимацией

Свёртка с децимацией*: результат

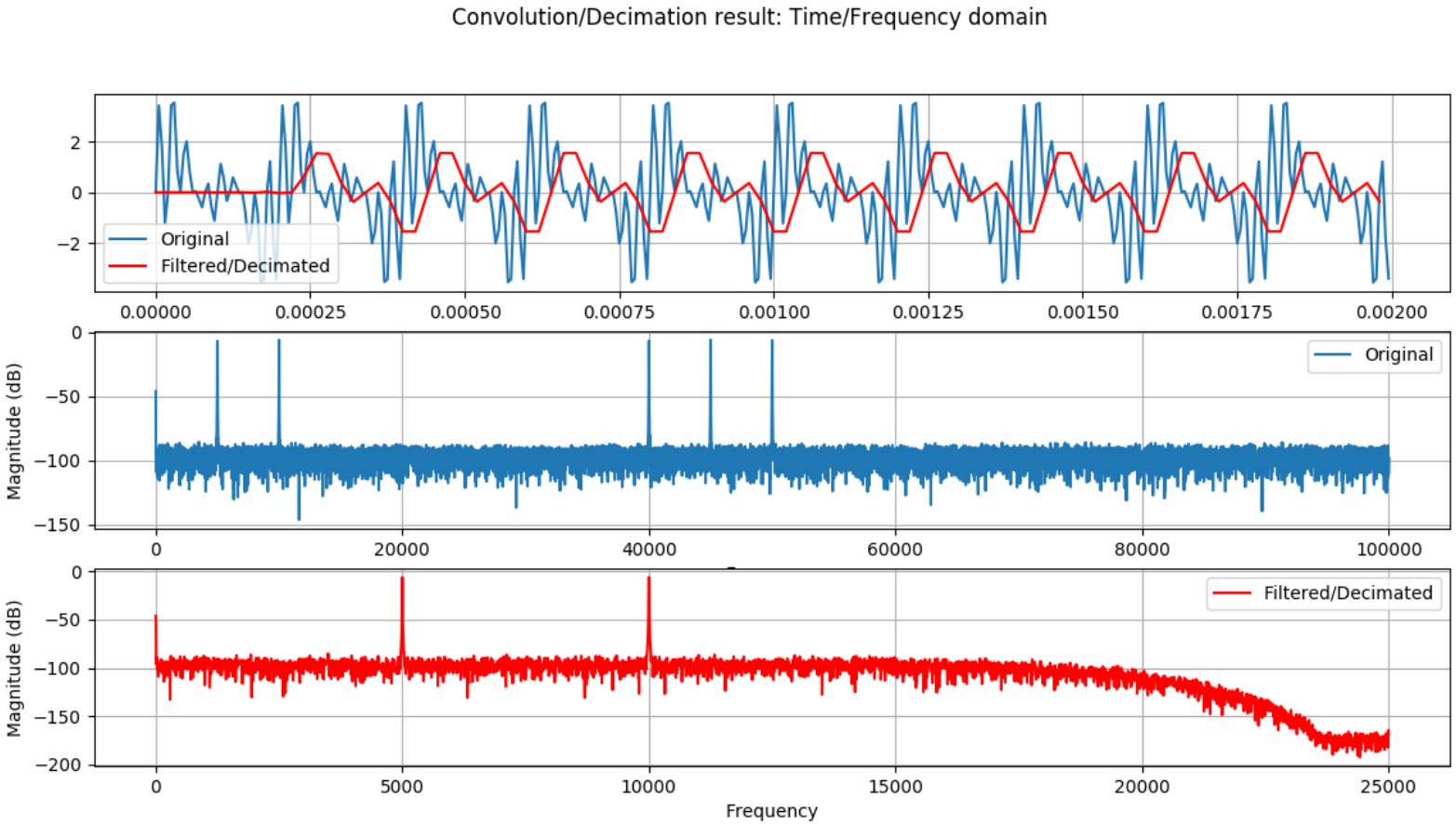

Рис.: Результат фильтрации с децимацией. Команда для отображения:
`python3 ../scripts/plot_decimated.py`

Спасибо за внимание!