Цифровые фильтры.
Библиотека функций на языке «С» и вспомогательные утилиты.

АО Микрон 2026

Оглавление

Автор: Жораев Тимур Юлдашевич

Каждый параграф проиндексирован соответствующим номером для отзывов и предложений.

UUID данного документа 019e4e7e-d7cb-7c24-999a-347d1cfa94c0

1 Установка компиляторов и библиотек по умолчанию.

Для построения проекта будет использоваться набор инструментов xPack.

1. Перейти на сайт https://xpack.github.io/docs/getting-started/prerequisites/

Общая установка node, npm, xpm с использованием готового скрипта

Установка
mkdir -pv "${HOME}/Downloads/" wget --quiet --output-document="${HOME}/Downloads/install-nvm-node-npm-xpm.sh" https://raw.githubusercontent.com/xpack/assets/master/scripts/install-nvm-node-npm-xpm.sh

Обратите внимание на созданную папку Downloads, в неё загружается непосредственно сам скрипт. Просмотреть скрипт можно

```
cat "${HOME}/Downloads/install-nvm-node-npm-xpm.sh"
```

или Midnight Commander (устанавливается sudo apt-get install mc)

Далее выполнить установку. Права суперпользователя не требуются, установка производится в папку

```
bash "${HOME}/Downloads/install-nvm-node-npm-xpm.sh"
exit
```

После чего установить актуальную версию инструментов для сборки RISCV

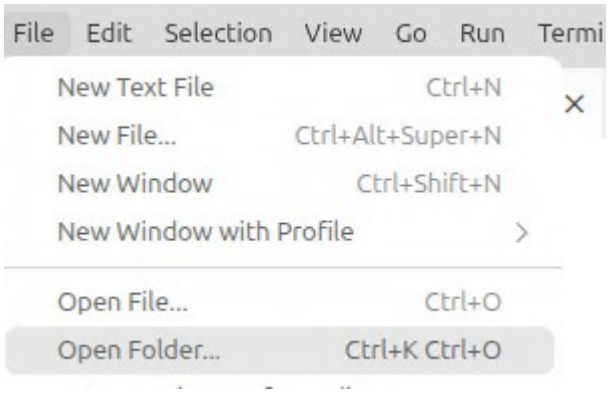
```
xpm install @xpack-dev-tools/riscv-none-elf-gcc@latest --verbose
```

Папка по умолчанию куда будет установлены инструменты:
~/.local/xPacks

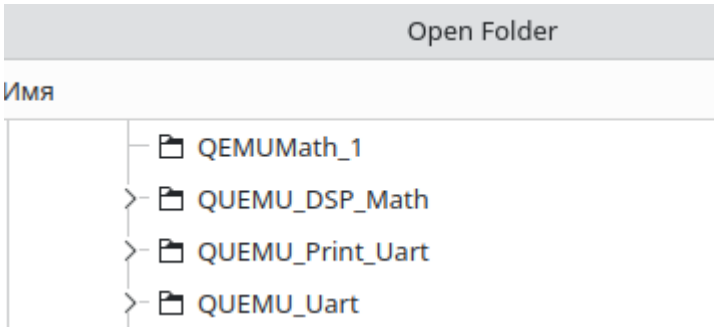
Следует отметить, что если пакет установлен и появился новый, будет создана папка с новой версией. Далее предлагается скрипт (см. приложение) который автоматически проверяет новую версию и устанавливает её в качестве необходимой по умолчанию и оповещает об этом окружение сборки cmake.

2 Использование симулятора QEMU для VS Code

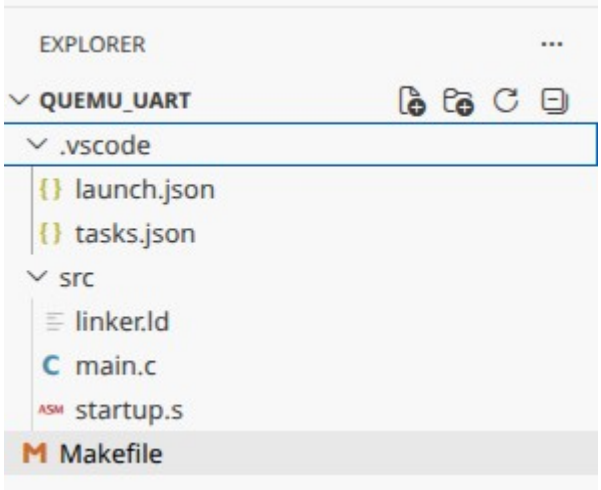
2. Далее необходимо открыть имеющиеся готовые проекты. Использовать открыть папку. Каждая папка — это отдельный проект.



Открыть имеющийся проект QUEMU_Uart (019d8ab8-274a-7886-b125-f040d4b76da7)



Проект состоит из следующих основных файлов и каталогов:

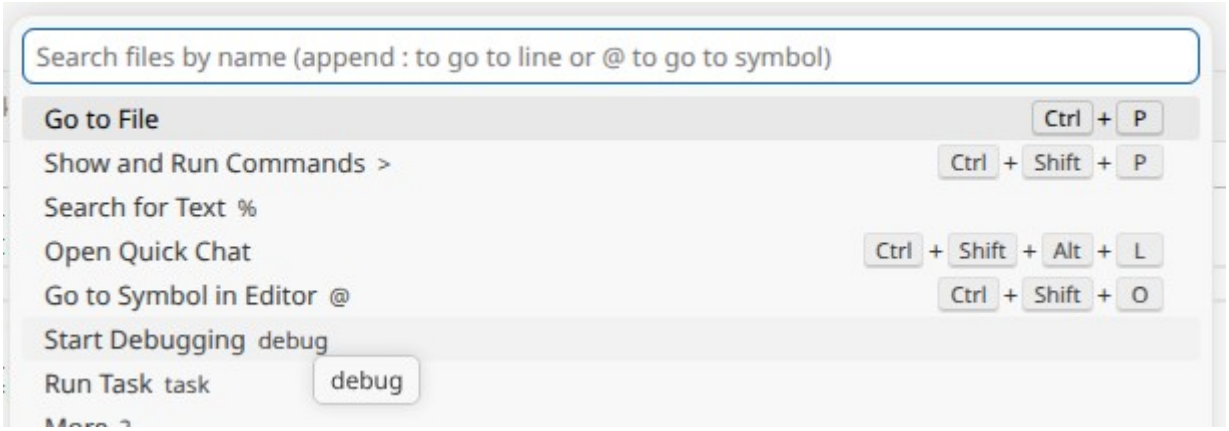


Открыть терминал Ctrl+Shift+`, осуществить сборку проекта командой make.

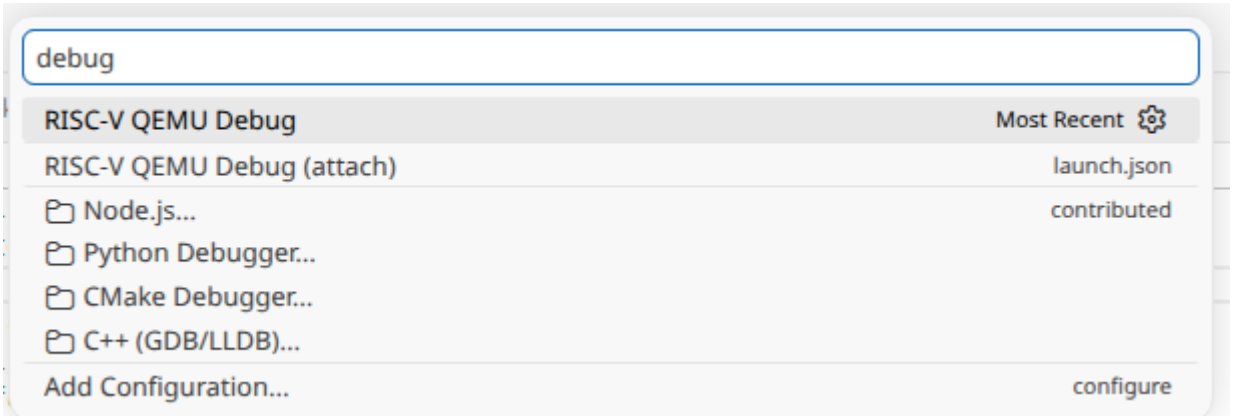
```
/home/timur/.local/xPacks/@xpack-dev-tools/riscv-none-elf-gcc/15.2.0-1.1/.content/bin/riscv-none-elf-gcc -march=rv32i -mabi=ilp32 -
dany -fvisibility=hidden -nostdlib -nostartfiles -g -O0 -Wall -Wextra -T src/linker.ld -Wl,-Map=app.map -o app.elf src/startup.o sr
/home/timur/.local/xPacks/@xpack-dev-tools/riscv-none-elf-gcc/15.2.0-1.1/.content/bin/../lib/gcc/riscv-none-elf/15.2.0/../../../../
/ld: warning: app.elf has a LOAD segment with RWX permissions
/home/timur/.local/xPacks/@xpack-dev-tools/riscv-none-elf-gcc/15.2.0-1.1/.content/bin/riscv-none-elf-objcopy -O binary app.elf app.
/home/timur/.local/xPacks/@xpack-dev-tools/riscv-none-elf-gcc/15.2.0-1.1/.content/bin/riscv-none-elf-objdump -D app.elf > app.dis
/home/timur/.local/xPacks/@xpack-dev-tools/riscv-none-elf-gcc/15.2.0-1.1/.content/bin/riscv-none-elf-size app.elf
   text    data     bss     dec     hex filename
    221       0    1024    1245     4dd app.elf
```

Обратить внимание на версию, скрипт построения Makefile автоматически получает новую.

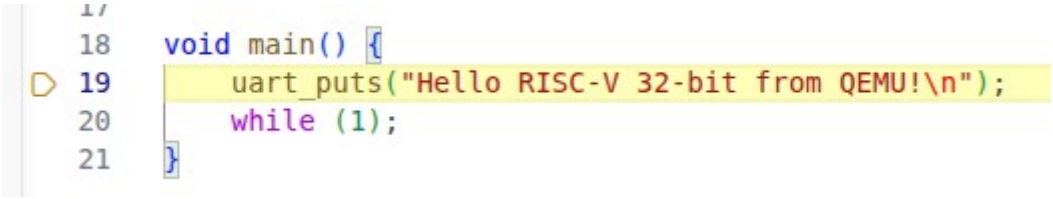
3. Перейти в верхнее окно пользовательских команд и выбрать отладку Debug



Выбрать RISC-V QEMU Debug



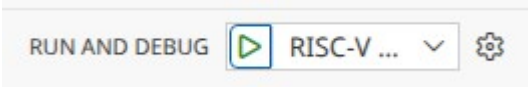
4. Далее можно устанавливать точки остановки, следовать по шагам, просматривать переменные и так далее как это принято в режиме отладки



Набор инструментов и соответствующие горячие клавиши

Окно отладки можно также включить отдельным значком слева

В нём можно произвести запуск в один клик



После запуска F5 программа выполнится

Далее необходимо закрыть терминал. Для упрощения отсутствует обработка выхода из виртуальной машины, которая делается отдельным сигналом. Это реализовано в последующих более сложных проектах. В данном случае осуществляется лишь ввод вывод.

5. Примеры файлов.

Функция main. Ввод-вывод осуществляется через последовательный порт виртуальной машины по адресу 0x10000000.

main. c

```c
//Простейший запуск RISC-V 32-bit на QEMU
//документ 019d8ab8-274a-7886-b125-f040d4b76da7
#define UART_ADDR 0x10000000
#define UART_REG(reg) ((volatile unsigned char *)(UART_ADDR + reg))
void uart_putc(char c) {
// Write character to the Transmitter Holding Register (THR)
*UART_REG(0) = c;
}
void uart_puts(const char *s) {
while (*s) {
uart_putc(*s++);
}
}
void main() {
uart_puts("Hello RISC-V 32-bit from QEMU!\n");
while (1);
}
```

6. Ассемблерный файл, который осуществляет инициализацию стека и переход на точку входа - функцию main()

startup.s

```
.section .text
.global _start
```

```
_start:
# Setup stack pointer (sp)
# The virt machine RAM starts at 0x80000000
la sp, _stack_top
# Jump to C main function
tail main

.section .bss
.align 4
stack:
.skip 1024 # 1KB stack space
_stack_top:
```

7. Файл команд компоновщику с указанием именованных секций расположения в памяти.

linker.ld

```
OUTPUT_ARCH("riscv")
ENTRY(_start)

SECTIONS
{
  /* QEMU RISC-V virt RAM starts here */
  . = 0x80000000;
  .text : {
    *(.text)
  }
  .data : {
    *(.data)
  }
  .bss : {
    *(.bss)
  }
  _end = .;
}
```

8. Данный простейший проект позволяет запустить QEMU и произвести базовую отладку.

Также обратить внимание на

launch.json

```
...
"MIMode": "gdb",
"miDebuggerPath": "${env:HOME}/.local/xPacks/@xpack-dev-tools/riscv-none-elf-gcc/14.2.0-3.1/.content/bin/riscv-none-elf-gdb",
"miDebuggerServerAddress": "localhost:1234",
"setupCommands": [
…
```

9. В этом варианте использован простой файл, содержащий внедрённый путь отладчика. Чтобы использовать динамический путь — см дальнейший пример.

tasks.json — пример файла конфигурации

```
...
"label": "Start QEMU GDB Server",
"type": "shell",
"command": "make",
"args": ["debug"],
"isBackground": true,
"problemMatcher": [
{
"pattern": [
{
"regexp": ".",
"file": 1,
"location": 2,
"message": 3
}
],
"background": {
"activeOnStart": true,
"beginsPattern": "Starting QEMU GDB server",
"endsPattern": "Starting QEMU GDB server"
}
}
]
...
```

Чтобы не было сообщения «Debug Anyway» необходимо установить необходимые шаблоны триггеров выполнения задачи из командной строки, что сервер стартует. Цель debug также обозначена в makefile:

Makefile

```
...
run: $(TARGET)
$(QEMU_PATH) -M virt -bios none -kernel $(TARGET) -nographic

debug: $(TARGET)
@echo "Starting QEMU GDB server on localhost:1234"
$(QEMU_PATH) -M virt -bios none -kernel $(TARGET) -nographic -s -S
...
```

Для отладки необходимо остановить виртуальный процессор ключом -S. Обратить внимание что необходимо закрывать каждый раз терминал, так как программа не посылает сигнал.

10. Далее следуют проекты, использующие расширенные возможности виртуальной машины — выход обратно в терминал по завершению main, использование специальные функции malloc-free без применения random, использующей затравку из таймера (отдельный регистр виртуальной машины), использование специального не библиотечного sprintf, реализованного с применением промпт-инженеринга по месту использования с заданными опциями. Проект получается полностью изолированный от внешних библиотек ввода-вывода, используются только библиотеки необходимые для работы с данными.

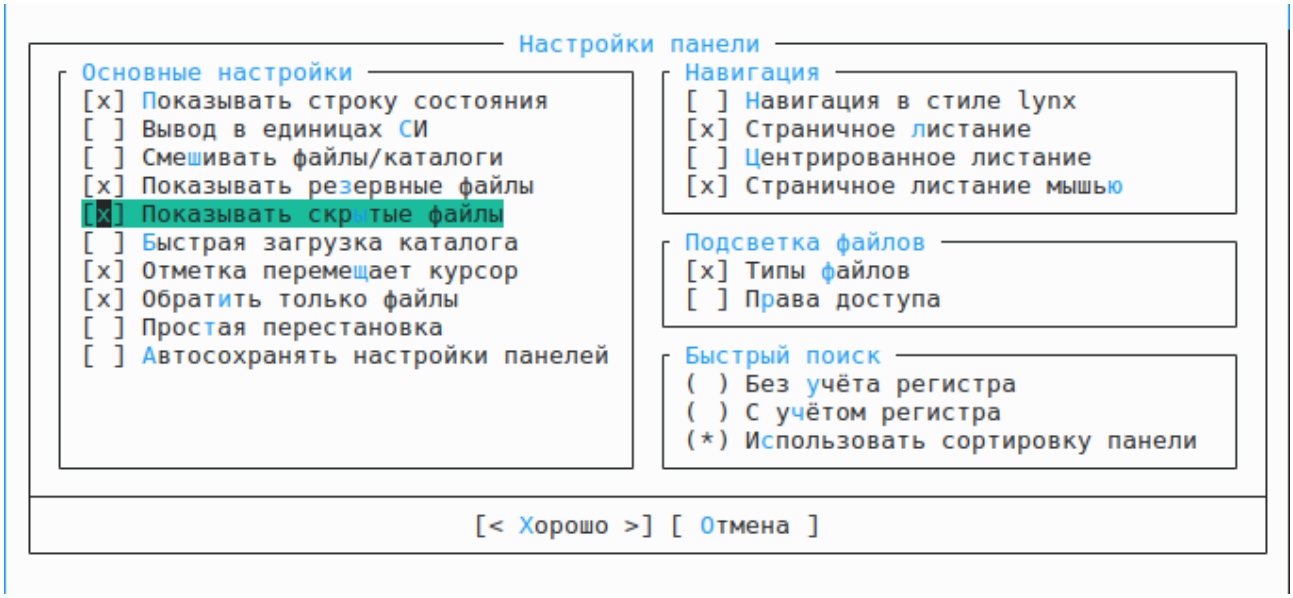
Проект, использующий сгенерированный ИИ printf (также который может быть использован для расширения функций) QUEMU_Print_Uart (019d8ae3-23e2-7ea5-9e46-d35f2e4dd208) он содержит специальный shell-скрипт find_xpack.sh, который позволяет корректно найти новую версию xPack. Он создаёт файл .xpack_cache.txt, который содержит пути к самой новой версии из всех установленных xPack. Также, файл launch.json осуществляет выполнение других инструментов с использованием «обёрток», например, gdb-wrapper.sh, который запускает самую свежую версию для отладки. Это необходимо для того, чтобы не использовать фиксированные («захардкоженные») пути в файле конфигураций launch.json, так как поле miDebuggerPath не поддерживает переменные окружения из envFile.

11. Следующий пример называется QUEMU_Print_Uart (019d8ae3-23e2-7ea5-9e46-d35f2e4dd208). Пример показывает умножение чисел в формате q26. Реализована функция qemu_exit_success, которая останавливает и завершает выполнение виртуальной машины, осуществляя выход в терминал. Также, в нём реализовано динамическое получение новой версии xPack с использованием обёртки для вызова gdb.

Ключевые скрипты и конфигурации VSCode для работы с динамически вызываемыми инструментами: gdb-wrapper.sh, find_xpack.sh. Дополнительно имеется файл .xpack_cache.txt

Более подробно — в Markdown-саммари в Приложении 1 код 5.

12. Скрипт find_xpack.sh, позволяет реализовать поиск новой версии среди установленных в xpack и сформировать файл конфигурации. Проверка установки новой версии производится раз в неделю. Информация о каталогах сохраняется в файл .xpack_cache.txt, присутствующий в корне проекта. Если в Linux файл не виден — включить режим отображения скрытых файлов, начинающихся с точки. В Midnight Commander это опция как показано ниже, вызовом меню F9 — настройки панели.



3 Дискретная передаточная функция и цифровой фильтр.

В данном разделе рассматривается преобразование от вида передаточной функции к виду цифрового фильтра и его эмуляция в Maxima а также реализация на C.

3.1 Нормализация передаточной функции

Пусть дана произвольная дискретная передаточная функция в общем виде (1) с произвольными коэффициентами, например, получаемая после дискретизации непрерывной (рассматривается в следующих разделах). Причём, порядок числителя и знаменателя может отличаться и быть произвольным.

$$\frac{u0+u1\cdot z+u2\cdot z^2}{w0+w1\cdot z+w2\cdot z^2} \tag{1}$$

Для преобразования к виду цифрового фильтра производится замена переменной (2). В коде Maxima приведённом в Приложении 1 это выглядит как $z=1/zm$ с введением новой переменной.

$$z \leftarrow z^{-1} \tag{2}$$

В этом случае коэффициенты «меняются местами» в обратном порядке относительно следования степеней полинома, в итоге получается передаточная функция (3) содержащая запаздывание z^{-1}:

$$\frac{u2+u1\cdot z^{-1}+u0\cdot z^{-2}}{w2+w1\cdot z^{-1}+w0\cdot z^{-2}} \tag{3}$$

Выполним нормировку (4) — разделим числитель и знаменатель (3) на $w2$.

$$\frac{\frac{u2}{w2}+\frac{u1}{w2}\cdot z^{-1}+\frac{u0}{w2}\cdot z^{-2}}{1+\frac{w1}{w2}\cdot z^{-1}+\frac{w0}{w2}\cdot z^{-2}} \tag{4}$$

Далее выполним замену переменных (5) в передаточной функции (4) для того чтобы получить упрощённое выражение.

$$\begin{aligned} b0&=\frac{u2}{w2} & a0&=1\\ b1&=\frac{u1}{w2} & a1&=\frac{w1}{w2}\\ b2&=\frac{u0}{w2} & a2&=\frac{w0}{w2}\\ &\dots & &\dots \end{aligned} \tag{5}$$

После подстановки получается (6), также следует отметить $a_0=w_2/w_2=1$.

$$\frac{b0+b1\cdot z^{-1}+b2\cdot z^{-2}}{1+a1\cdot z^{-1}+a2\cdot z^{-2}} \tag{6}$$

3.2 Переход к рекуррентному соотношению

Из полученной нормированной передаточной функции (6) цифрового фильтра можно легко перейти к рекуррентному соотношению для вычисления

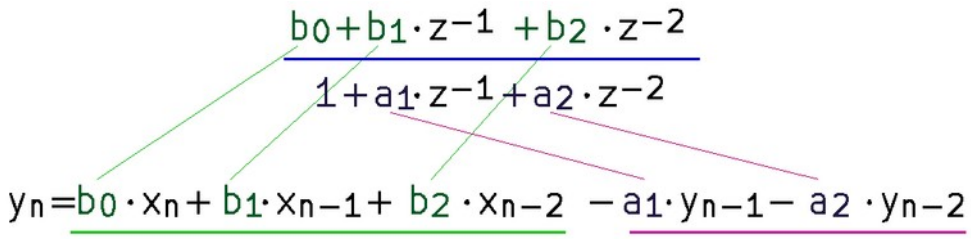


Рис. 1 Переход от формы записи передаточной функции цифрового фильтра к рекуррентному соотношению

На рис. 1 представлены следующие обозначения: y_n — текущий выходной отсчёт, x_n— текущий входной отсчёт, x_{n-1} — предыдущий входной, x_{n-2} — перед предыдущим входного, y_{n-1} — предыдущий выходного и так далее соответственно. Обратить внимание на знак минус перед выходными отсчётами умноженными на коэффициенты a_1, a_2. Данный знак обычно учитывается при расчёте коэффициентов. Таким образом, имеется массив входных отсчётов и их предыдущих значений а также массив предыдущих значений выходных отсчётов, текущий выходной отсчёт определяется как умножение с накоплением.

Умножение с накоплением — основная операция в цифровой обработке сигналов. Процессоры ЦОС содержат встроенные команды, включающие также операции пост- или прединкремента, декремента указателей, позволяя одновременно следовать по соответствующим массивам. В данном случае будем использовать аналог данной операции для упрощения расчётов. В процессорах ЦОС используется команда повторения следующей операции, например REP (repetition), которая и является умножением с накоплением MAC (Multiply and Accumulate).

3.3 Коэффициенты и отсчёты цифрового фильтра в виде массивов

Для формирования массивов используется следующий переход, представленный на рис. 2. Обратить внимание что в данной реализации цифрового фильтра коэффициент CY_0 равен нулю. Этот коэффициент является множителем при Y_0 - куда записывается текущий отсчёт. Данная форма записи соответствует прямому порядку следования коэффициентов. Для дальнейшей оптимизации возможно применение обратного, однако, в данном случае этот вариант не рассматривается.

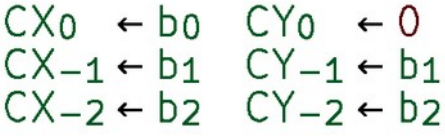


Рис. 2 Исследуемая схема для дискретизации ПФ первого порядка

Переход от рекуррентного соотношения к удобной форме для расчёта цифрового фильтра с умножением и накоплением представлен на рис. 3. Видно, что используется только сложение а знак минус «-» учитывается при расчёте коэффициентов и переходе от передаточной функции к этому виду.

$$y_n = b_0 \cdot x_n + b_1 \cdot x_{n-1} + b_2 \cdot x_{n-2} - a_1 \cdot y_{n-1} - a_2 \cdot y_{n-2}$$

$$Y_0 \leftarrow CX_0 \cdot X_0 + CX_1 \cdot X_{-1} + CX_2 \cdot X_{-2} + CY_1 \cdot Y_{-1} + CY_2 \cdot Y_{-2}$$

Рис. 3 Переход от рекуррентного соотношения к вычислению с использованием умножения с накоплением

Массивы входных и выходных отсчётов располагаются друг за другом, также как и массивы коэффициентов. В этом случае применяя умножение с накоплением можно перейти от накопления входных отсчётов к выходным без какой-либо инициализации заново. Данная структура представлена на рис. 4.

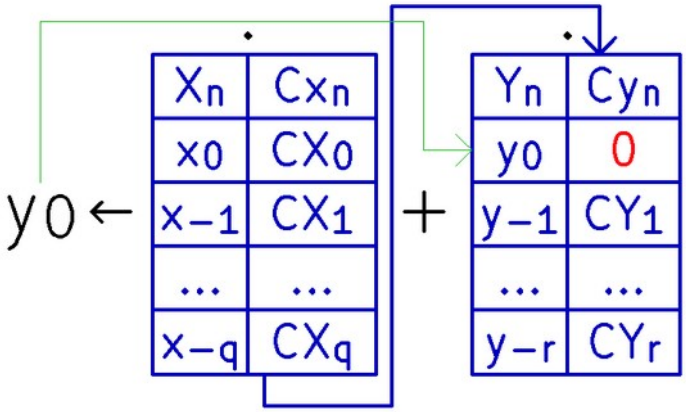


Рис. 4 Структурная схема вычислений цифрвого фильтра

DSP-процессоры имеют пост-инкремент, декремент указателя а также пересылку данных для обновления предыдущих отсчётов в рамках выполнения одной команды, также, загружаются регистры соответствующие массивам отсчётов и массивам коэффициентов. Для данной архитектуры нет такой оптимизации, поэтому, используется прямое расположение коэффициентов и отсчётов, более удобное для отладки, увеличение указателя осуществляется в цикле.

3.4 Реализация цифрового фильтра с использованием Maxima

Код Maxima представлен в Приложении 1, код 2. Данный код иллюстрирует вычисления с использованием входных массивов коэффициентов и отсчётов.

В качестве примера можно привести следующий переходной процесс, соответствующий коэффициентам выходных отсчтёв CX 1,0.19999999999999996,−0.24, коэффициентов выходных отсчётов CY 1,−0.6,0.33999999999999997. Нулевое значение CY[1] присваивается отдельно в функции инициализации. Пример переходного процесса с заданными коэффициентами представлен на рис. 5.

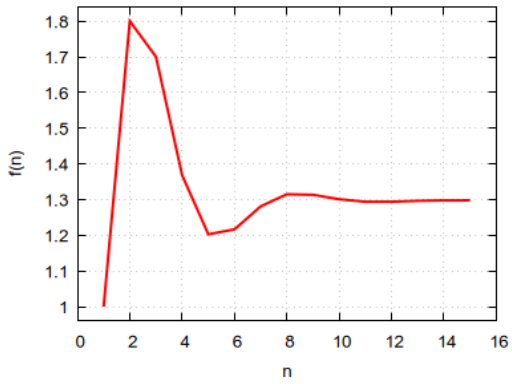


Рис. 5 Пример переходного процесса для цифрового фильтра.

Характерные особенности переходных процессов для заданного расположения нулей и полюсов исходной передаточной функции (не цифрового фильтра, нули и полюса для коэффициентов при z а не z^{-1}) представлены на рис. 6.

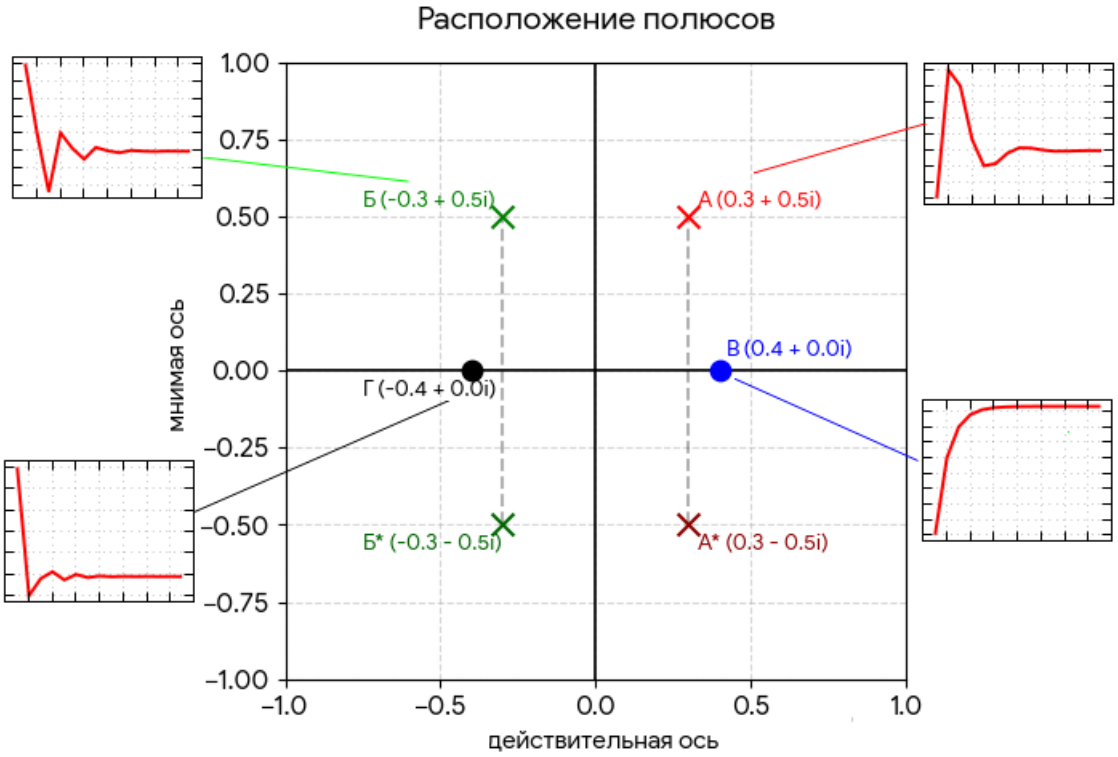


Рис. 6 Характерные особенности переходного процесса для заданного расположения нулей и полюсов дискретной ПФ

4 Реализация цифрового фильтра

Общая структура проекта:

Путь	Назначение
.vscode	Конфигурация VSCode. Содержит файлы JSON: tasks.json, launch.json.
Makefile	Сборка проекта с использованием утилиты make
gdb-wrapper.sh	Инициализация путей, задание переменных окружения, запуск GDB (запускается автоматически VSCode)
find_xpack.sh	Поиск актуальной версии xPack, актуализация файла с путями (запускается автоматически VSCode)
.xpack_cache.txt	Файл, содержащий пути к необходимым инструментам и используемый для задания переменных окружения
plot_tests.py	Вывод результатов тестирования
requirements.txt	Для установки библиотек pip Python
generate_requirements.sh	Автоматическая установка виртуального окружения, если оно присутствует, пропуск. Также, этим файлом запускается скрипт: ./generate_requirements.sh plot_tests.py. Виртуальное окружение инициализируется автоматически и запускается скрипт
convert_png_to_png.sh convert_png_to_jpg.sh	Сжатие произвольных png-изображений в jpg или png с ограничением размера. Изображения получаются в ходе работы скрипта plot_tests.py
Maxima	Maxima-скрипты содержащие примеры для расчётов цифровых фильтров
Markdown	Markdown-файлы с описанием необходимых модулей проекта. Могут быть использованы для проверки, модификации и генерацией ИИ новых файлов
src/dspmath.c src/dspmath.h src/dspmathtest.c	Основные функции реализации различного вида цифровых фильтров и проведение их тестирования.
src/io.c src/io.h src/printarr.c src/printarr.h	Функции ввода-вывода для виртуальной машины, функции печати символа и построения графиков в ASCII виде
src/main.c	Основная функция
src/linker.ld src/startup.s	Командный файл линкера и запуска для виртуальной машины
src/memory.c src/memory.h src/randomgenerator.c src/randomgenerator.h	Управление динамической памятью и генератор псевдослучайных чисел

Модель цифрового фильтра также приведена в DigFilterOverallExCoeff.wxmx. Методы дискретизации рассматриваются в следующем разделе, позволяющие получить коэффициенты для тестирования.

13. В этом разделе рассматриваются библиотечные функции цифровых фильтров а также их тесты.

Основной проект для локального тестирования — QUEMU_DSP_Math, ID документа 019e5dec-596d-7b6f-b6c4-fc051844b731 (main файл)

4.1 Виды цифровых фильтров в библиотеке

14. В библиотеке имеются следующие реализации цифровых фильтров, приведены прототипы функций.

Одноканальный фильтр первого порядка
Структура для работы с фильтром:

```
typedef struct IIR1 {
// --- Линия задержки (State variables) ---
long x; /**< @brief Текущий входной отсчет: x[n] */
long x_1; /**< @brief Предыдущий входной отсчет: x[n-1] (z^-1) */
long y; /**< @brief Текущий выходной отсчет: y[n] */
long y_1; /**< @brief Предыдущий выходной отсчет: y[n-1] (z^-1) */

// --- Коэффициенты фильтра (Coefficients) ---
long Cx; /**< @brief Коэффициент b0 (прямая связь / Feedforward) */
long Cx_1; /**< @brief Коэффициент b1 (прямая связь / Feedforward) */
long Cy; /**< @brief Коэффициент a0 (заглушка для выравнивания цикла) */
long Cy_1; /**< @brief Коэффициент -a1 (обратная связь / Feedback) */
} IIR1;
```

Прототипы функций:

```
// Function prototypes for dspmath.c (core compute and init functions only)
void IIR1_Reset(IIR1 *f);
void IIR1_SetCoeffs(IIR1 *f, long a1_q26);
long IIR1_Compute(IIR1 *f, long input_sample);
```

Одноканальный фильтр высшего порядка

Структура для работы с фильтром:

```
typedef struct IIRHigh {
int NumCx; // Число коэффициентов b (числитель, включая b0)
int NumCy; // Число коэффициентов a (знаменатель, начиная с a1)
long *Cx; // Массив коэффициентов b: [b0, b1, ... bN]
long *x; // Линия задержки входа (состояния): [x[n-1], x[n-2], ... x[n-N]]
long *Cy; // Массив коэффициентов a: [a1, a2, ... aM] (уже инвертированных для суммы)
long *y; // Линия задержки выхода (состояния): [y[n-1], y[n-2], ... y[n-M]]
} IIRHigh;
```

Прототипы функций:

```
long IIRHigh_Compute(IIRHigh *f, long input_sample);
void IIRHigh_Reset(IIRHigh *f);
void IIRHigh_SetCoeffs(IIRHigh *f, int num_x, int num_y,
long *cx_ptr, long *cy_ptr, long *x_ptr, long *y_ptr);
```

Двухканальный фильр первого порядка

Структура для работы с фильтром:

```
typedef struct IIR1_2 {
/**
* @brief Состояние каналов.
* Формат: [Канал][Переменная], где переменные: [0]:x, [1]:x_1, [2]:y, [3]:y_1
*/
long state[2][4];

/**
* @brief Коэффициенты фильтра.
* Формат: [0]:Cx (b0), [1]:Cx_1 (b1), [2]:Cy (a0=0), [3]:Cy_1 (-a1)
*/
long coeffs[4];
} IIR1_2;
```

Прототипы функций:

```
long IIR1_2_Compute(IIR1_2 *f, long in1, long in2);
void IIR1_2_Reset(IIR1_2 *f);
void IIR1_2_SetCoeffs(IIR1_2 *f, long a1_q26);
```

Трёхканальный фильтр первого порядка

Структура для работы с фильтром:

```
typedef struct IIR1_3 {
/**
* @brief Состояние каналов (линии задержки).
* Индексы: [0]:x, [1]:x_1, [2]:y, [3]:y_1
*/
long state[3][4];

/**
* @brief Общие коэффициенты: [Cx (b0), Cx_1 (b1), Cy (a0=0), Cy_1 (-a1)]
*/
long coeffs[4];
} IIR1_3;
```

Прототипы функций:

```
void IIR1_3_Compute(IIR1_3 *f, long in1, long in2, long in3);
void IIR1_3_Reset(IIR1_3 *f);
void IIR1_3_SetCoeffs(IIR1_3 *f, long a1_q26);
```

Двухканальный фильтр второго порядка

Структура для работы с фильтром:

```
typedef struct IIR2_2 {
/**
* @brief Состояние каналов (линии задержки).
* 2 канала, в каждом по 6 переменных: [x, x_1, x_2, y, y_1, y_2]
*/
long state[2][6];

/**
* @brief Общие коэффициенты для обоих каналов.
* Формат: [b0, b1, b2, a0(0), -a1, -a2]
*/
long coeffs[6];
} IIR2_2;
```

Прототипы функций:

```
void IIR2_2_Compute(IIR2_2 *f, long in0, long in1, long *out);
void IIR2_2_Reset(IIR2_2 *f);
// Note: IIR2_2_SetCoeffs_Random is not included here (has random init, goes to test file)
```

Трёхканальный фильтр второго порядка
Структура для работы с фильтром:

```
typedef struct IIR2_3 {
/**
* @brief Линии задержки для 3-х каналов.
* Каждая строка: [0]:x, [1]:x_1, [2]:x_2, [3]:y, [4]:y_1, [5]:y_2
*/
long state[3][6];

/**
* @brief Общие коэффициенты фильтра.
* Порядок: [0]:b0, [1]:b1, [2]:b2, [3]:a0(0), [4]:-a1, [5]:-a2
*/
long coeffs[6];
} IIR2_3;
```

Прототипы функций:

```
void IIR2_3_Compute(IIR2_3 *f, long in0, long in1, long in2, long *out);
void IIR2_3_Reset(IIR2_3 *f);
// Note: IIR2_3_SetCoeffs_Random is not included here (has random init, goes to test file)
```

15. Суффикс, используемый в наименовании функции отражает следующее:
void ..._Reset(...); - функция сброса переменных состояния фильтра
long ..._Compute(...); - функция вычисления состояния фильтра для одного отсчёта

Функции, содержащие суффикс ..._SetCoeffs присутствуют в тестировании но отсутствуют в секции памяти ROM Bootloader, они необходимы для задания коэффициентов фильтра, которые пользователь определяет исходя из вычислений. В качестве предоставляемых в ROM функций используются только сброс начальных условий и расчёт одной итерации фильтров

Для визуализации данных используется python-скрипт, код 1 которого приведён в Приложении 1 в виде Markdown, по которому ИИ-модель может сформировать необходимую программу.

4.2 Фильтры первого и второго порядка одноканальные

16. Данные фильтры представляют собой классическую реализацию с использованием умножения с накоплением для одного канала

При реализации ПИ-регулятора как цифрового фильтра требуется обеспечить ограничение по выходу или переменных состояния. Это можно сделать внешней функцией, например, после вычисления цифрового фильтра поставить заданные ограничения с использованием условных выражений.

4.3 Многоканальные фильтры первого и второго порядка

17. Для реализации многоканальной фильтрации используется также оптимальное расположение коэффициентов и отсчётов цифрового фильтра друг за другом. В этом случае при нулевом значении индекса производится замена входного отсчёта входным сигналом а аккумулятор сбрасывается. Тем самым, можно использовать как условие в цикле так и отдельный цикл для умножения с накоплением и предварительно заданными указателями.

5 Markdown файлы, описывающие цифровые фильтр и примеры работы

18. В настоящее время широко применяются системы ИИ для генерации кода, а также, для формирования описания алгоритмов, структур данных по исходному коду.

Пример работы — в файле dspmathtest.c. Чтобы выполнить тест необходимо ввести make run. Для очистки проекта — make clean, для сборки — make.

6 Описание методов используемых для формирования цифрового фильтра из непрерывной передаточной функции и непосредственно

В этом разделе описываются методы, позволяющие определить коэффициенты дискретных передаточных функций, цифровых фильтров, на основе физического объекта, представляющего собой электротехническую схему.

Следует отметить что приведённые большие формулы даны справочно и отражают вывод программы Maxima. Более подробно примеры рассматриваются в файле TransferFuncs1and2ord.wxmx , ID документа 019dae87-6157-7111-a91a-f8c5023a30b8 а также в файле DigFilterOverallExCoeff.wxmx, ID документа 019e6d15-2f56-77e1-b80e-092ed8d15992.

6.1 Дискретизация произвольной непрерывной передаточной функции с использованием подстановки Тастина

19. Дано: передаточные функции для апериодического звена и колебательного, например, заданные с использованием электрической схемы. Реализация представлена в документе 019dae87-6157-7111-a91a-f8c5023a30b8, файл TransferFuncs1and2ord.wxmx.

Пример для передаточной функции первого порядка.

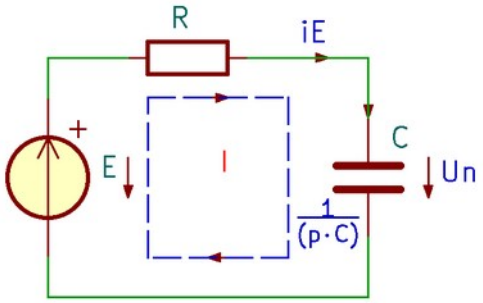


Рис. 7 Исследуемая схема для дискретизации ПФ первого порядка

Задаём исходную систему уравнений

$$\left[\frac{iE}{C\ p}+R\ iE-E, Un=\frac{iE}{C\ p}\right]$$

Решение системы уравнений относительно выбранных переменных состояния: выбираем необходимую Un

$$\frac{E}{C\ R\ p+1}$$

Формирование передаточной функции второго порядка

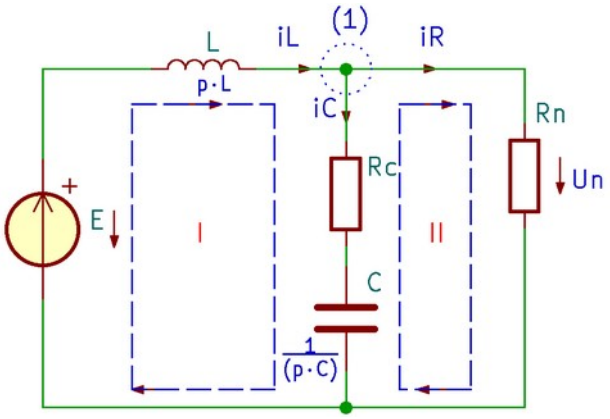


Рис. 8 Исследуемая схема для дискретизации ПФ второго порядка

задаём исходную систему уравнений

$$\left[L\ iL\ p+\frac{iC}{C\ p}+Rc\ iC-E\ ,-\left(\frac{iC}{C\ p}\right)+Rn\ iR-Rc\ iC\ ,-iR+iL-iC\ ,Un=Rn\ iR \right]$$

решение системы уравнений относительно выбранных переменных состояния

выбираем необходимую Un относительно выбранных переменных состояния [iL,iC,iR,Un]

$$\frac{C\ E\ Rc\ Rn\ p+E\ Rn}{\left(C\ L\ Rn+C\ L\ Rc\right)\ p^2+\left(C\ Rc\ Rn+L\right)\ p+Rn}$$

6.2 Дискретизация произвольной непрерывной передаточной функции с использованием подстановки Тастина

20. Рассматривается применение подстановки, являющейся первым элементом ряда билинейного преобразования:

Подстановка Тастина — первый элемент ряда Билинейного преобразования

$$p=\frac{2\ fd\ \left(z-1\right)}{z+1}$$

Для передаточной функции первого порядка:

$$\frac{E\ \left(z+1\right)}{2\ C\ R\ fd\ z+z-2\ C\ R\ fd+1}$$

Для передаточной функции второго порядка:

$$\frac{E\ Rn\ (z+1)\ (2\ C\ Rc\ fd\ z+z-2\ C\ Rc\ fd+1)}{4\ C\ L\ Rn\ fd^2\ z^2+4\ C\ L\ Rc\ fd^2\ z^2+2\ C\ Rc\ Rn\ fd\ z^2+2\ L\ fd\ z^2+Rn\ z^2-8\ C\ L\ Rn\ fd^2\ z-8\ C\ L\ Rc\ fd^2\ z+2\ Rn\ z+4\ C\ L\ Rn\ fd^2+4\ C\ L\ Rc\ fd^2-2\ C\ Rc\ Rn\ fd-2\ L\ fd+Rn}$$

Получается относительно громоздкое выражение, далее оно будет преобразовано к вектору коэффициентов при z в следующем разделе, где рассматривается дискретизация методом нулей и полюсов.

21. Сравнение частотных характеристик можно произвести с использованием подстановок, которые являются ключевыми для анализа комплексно-частотных характеристик, включая АЧХ и ФЧХ в операторной форме.

Важно!

переход к КЧХ для непрерывной ПФ

$p = 2\ \%i\ \pi\ f$

переход к КЧХ для дискретной ПФ

$z = \%e^{\frac{2\ \%i\ \pi\ f}{fd}}$

22. *%i*— мнимая единица, *%e* — экспонента (2.7…), %pi — число π, (3.14…). В Maxima %pi, %e или exp(1) является выражением с численным типом данных, i, e или π — символ, который не несёт в себе специального значения, поэтому необходимо внимательно использовать данные константы с указанием приставки процент %.

Для передаточной функции первого порядка.

23. КЧХ непрерывной ПФ, состоящая из действительной и мнимой частей - вектор двух действительных выражений

$$\left[\frac{E}{4\ \pi^2\ C^2\ R^2\ f^2+1}, -\left(\frac{2\ \pi\ C\ E\ R\ f}{4\ \pi^2\ C^2\ R^2\ f^2+1}\right)\right]$$

Для упрощения выражений, содержащих модуль от величин, имеющих заранее определённых знак можно использовать допущения

допущения о диапазонах переменных, чтобы исключить знак модуля при упрощении

$[E>0, f>0, fd>0, R>0, C>0, L>0]$

АЧХ непрерывной ПФ

$$\frac{E}{\sqrt{4\ \pi^2\ C^2\ R^2\ f^2+1}}$$

ФЧХ непрерывной ПФ

$-atan(2\ \pi\ C\ R\ f)$

КЧХ дискретной ПФ

$$\left[-\left(\frac{E\cos\left(\frac{2\,n\,f}{fd}\right)+E}{\left(4\,C^2\,R^2\cos\left(\frac{2\,n\,f}{fd}\right)-4\,C^2\,R^2\right)fd^2-\cos\left(\frac{2\,n\,f}{fd}\right)-1}\right),\frac{2\,C\,E\,R\sin\left(\frac{2\,n\,f}{fd}\right)fd}{\left(4\,C^2\,R^2\cos\left(\frac{2\,n\,f}{fd}\right)-4\,C^2\,R^2\right)fd^2-\cos\left(\frac{2\,n\,f}{fd}\right)-1}\right]$$

АЧХ дискретной ПФ

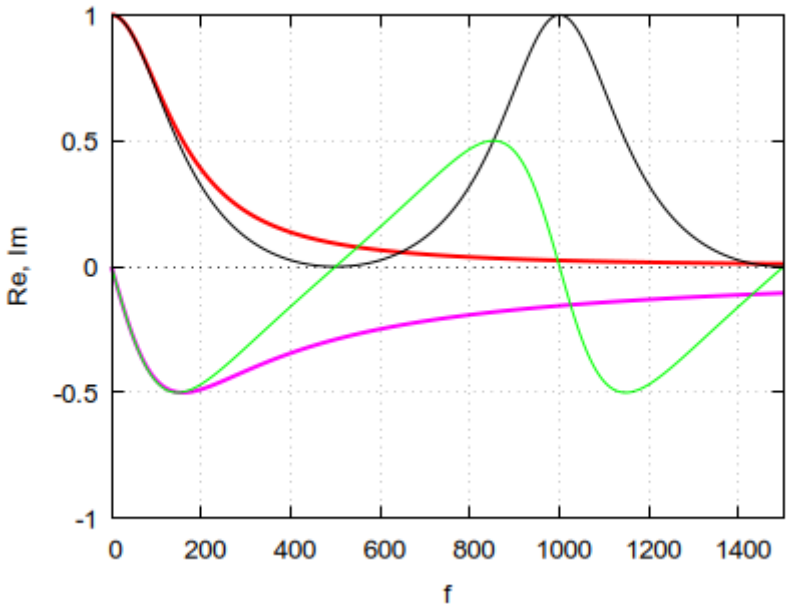
$$\frac{E\sqrt{2\cos\left(\frac{2\,\pi\,f}{fd}\right)+2}}{\sqrt{\left(8\,C^2\,R^2-8\,C^2\,R^2\cos\left(\frac{2\,\pi\,f}{fd}\right)\right)fd^2+2\cos\left(\frac{2\,\pi\,f}{fd}\right)+2}}$$

ФЧХ дискретной

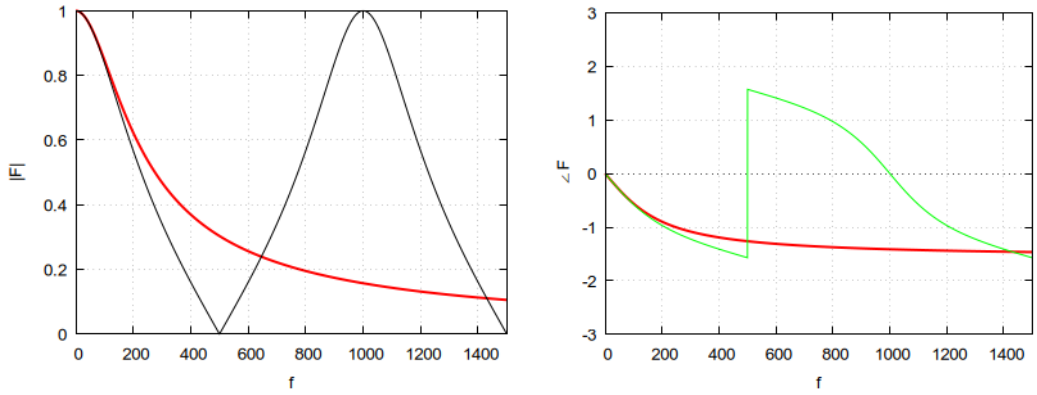
$$\operatorname{atan2}\left(-\left(2\,C\,R\sin\left(\frac{2\,n\,f}{fd}\right)fd\right)-\sin\left(\frac{2\,n\,f}{fd}\right),\left(2\,C\,R\cos\left(\frac{2\,n\,f}{fd}\right)-2\,C\,R\right)fd+\cos\left(\frac{2\,n\,f}{fd}\right)+1\right)+\operatorname{atan2}\left(\sin\left(\frac{2\,n\,f}{fd}\right),\cos\left(\frac{2\,n\,f}{fd}\right)+1\right)$$

24. Построение графиков частотных характеристик. Для непрерывной функции — красный и малиновый, для дискретной — чёрный и зелёный.

Комплексно-частотная



АЧХ и ФЧХ (красный - непрерывная)



25. Для передаточной функции второго порядка можно составить аналогично.

Получение КЧХ непрерывной ПФ

$$\frac{2\,\%i\,\pi\,C\,E\,Rc\,Rn\,f+E\,Rn}{-\left(4\,\pi^2\left(C\,L\,Rn+C\,L\,Rc\right)f^2\right)+2\,\%i\,\pi\left(C\,Rc\,Rn+L\right)f+Rn}$$

Получение КЧХ дискретизированной ПФ

$$\left(\left(\%e^{\frac{2\,\%i\,n\,f}{fd}}+1\right)E\,Rn\left(2\,\%e^{\frac{2\,\%i\,n\,f}{fd}}\,C\,Rc\,fd-2\,C\,Rc\,fd+\%e^{\frac{2\,\%i\,n\,f}{fd}}+1\right)\right)/\left(4\,\%e^{\frac{4\,\%i\,n\,f}{fd}}\,C\,L\,Rn\,fd^2-8\,\%e^{\frac{2\,\%i\,n\,f}{fd}}\,C\,L\,Rn\,fd^2+4\,C\,L\,Rn\,fd^2+4\,\%e^{\frac{4\,\%i\,n\,f}{fd}}\,C\,L\,Rc\,fd^2-8\,\%e^{\frac{2\,\%i\,n\,f}{fd}}\,C\,L\,Rc\,fd^2+4\,C\,L\,Rc\,fd^2+2\,\%e^{\frac{4\,\%i\,n\,f}{fd}}\,C\,Rc\,Rn\,fd-2\,C\,Rc\,Rn\,fd+2\,\%e^{\frac{4\,\%i\,n\,f}{fd}}\,L\,fd-2\,L\,fd+\%e^{\frac{4\,\%i\,n\,f}{fd}}\,Rn+2\,\%e^{\frac{2\,\%i\,n\,f}{fd}}\,Rn+Rn\right)$$

26. Использование формулы де Муавра для преобразования комплексной экспоненты в тригонометрические функции

$$\left(E\,Rn\left(\%i\sin\left(\frac{2\,n\,f}{fd}\right)+\cos\left(\frac{2\,n\,f}{fd}\right)+1\right)\left(2\,C\,Rc\left(\%i\sin\left(\frac{2\,n\,f}{fd}\right)+\cos\left(\frac{2\,n\,f}{fd}\right)\right)fd-2\,C\,Rc\,fd+\%i\sin\left(\frac{2\,n\,f}{fd}\right)+\cos\left(\frac{2\,n\,f}{fd}\right)+1\right)\right)/\left(4\,C\,L\,Rn\left(\%i\sin\left(\frac{4\,n\,f}{fd}\right)+\cos\left(\frac{4\,n\,f}{fd}\right)\right)fd^2+4\,C\,L\,Rc\left(\%i\sin\left(\frac{4\,n\,f}{fd}\right)+\cos\left(\frac{4\,n\,f}{fd}\right)\right)fd^2-8\,C\,L\,Rn\left(\%i\sin\left(\frac{2\,n\,f}{fd}\right)+\cos\left(\frac{2\,n\,f}{fd}\right)\right)fd^2-8\,C\,L\,Rc\left(\%i\sin\left(\frac{2\,n\,f}{fd}\right)+\cos\left(\frac{2\,n\,f}{fd}\right)\right)fd^2+4\,C\,L\,Rn\,fd^2+4\,C\,L\,Rc\,fd^2+2\,C\,Rc\,Rn\left(\%i\sin\left(\frac{4\,n\,f}{fd}\right)+\cos\left(\frac{4\,n\,f}{fd}\right)\right)fd+2\,L\left(\%i\sin\left(\frac{4\,n\,f}{fd}\right)+\cos\left(\frac{4\,n\,f}{fd}\right)\right)fd-2\,C\,Rc\,Rn\,fd-2\,L\,fd+Rn\left(\%i\sin\left(\frac{4\,n\,f}{fd}\right)+\cos\left(\frac{4\,n\,f}{fd}\right)\right)+2\,Rn\left(\%i\sin\left(\frac{2\,n\,f}{fd}\right)+\cos\left(\frac{2\,n\,f}{fd}\right)\right)+Rn\right)$$

КЧХ непрерывной ПФ, состоящая из действительной и мнимой частей - вектор двух действительных выражений

$$\left[\frac{E\,Rn\left(Rn-4\,\pi^2\left(C\,L\,Rn+C\,L\,Rc\right)f^2\right)+4\,\pi^2\,C\,E\,Rc\,Rn\left(C\,Rc\,Rn+L\right)f^2}{\left(Rn-4\,\pi^2\left(C\,L\,Rn+C\,L\,Rc\right)f^2\right)^2+4\,\pi^2\left(C\,Rc\,Rn+L\right)^2f^2},\frac{2\,\pi\,C\,E\,Rc\,Rn\,f\left(Rn-4\,\pi^2\left(C\,L\,Rn+C\,L\,Rc\right)f^2\right)-2\,\pi\,E\,Rn\left(C\,Rc\,Rn+L\right)f}{\left(Rn-4\,\pi^2\left(C\,L\,Rn+C\,L\,Rc\right)f^2\right)^2+4\,\pi^2\left(C\,Rc\,Rn+L\right)^2f^2}\right]$$

допущения о диапазонах переменных, чтобы исключить знак модуля при упрощении

assume(E>0,f>0,fd>0,Rn>0,Rc>0,C>0,L>0);

АЧХ непрерывной ПФ

$$\frac{\sqrt{4\,\pi^2\,C^2\,E^2\,Rc^2\,Rn^2\,f^2+E^2\,Rn^2}}{\sqrt{\left(16\,\pi^4\,C^2\,L^2\,Rn^2+32\,\pi^4\,C^2\,L^2\,Rc\,Rn+16\,\pi^4\,C^2\,L^2\,Rc^2\right)f^4+\left(\left(4\,\pi^2\,C^2\,Rc^2-8\,\pi^2\,C\,L\right)Rn^2+4\,\pi^2\,L^2\right)f^2+Rn^2}}$$

ФЧХ непрерывной ПФ

$$atan(2\ \pi\ C\ Rc\ f) - atan2\left((2\ \pi\ C\ Rc\ Rn + 2\ \pi\ L)\ f, \left(-(4\ \pi^2\ C\ L\ Rn) - 4\ \pi^2\ C\ L\ Rc\right) f^2 + Rn\right)$$

КЧХ дискретной ПФ

$$[-(E\ Rn^2\ (4\ C^2\ Rc^2\ \cos(\frac{4\ n\ f}{fd})\ fd^2 - 4\ C\ L\ \cos(\frac{4\ n\ f}{fd})\ fd^2 - 4\ C^2\ Rc^2\ fd^2 + 4\ C\ L\ fd^2 - \cos(\frac{4\ n\ f}{fd}) - 4\ \cos(\frac{2\ n\ f}{fd}) - 3))/(16\ C^2\ L^2\ Rn^2\ \cos(\frac{4\ n\ f}{fd})\ fd^4 + 32\ C^2\ L^2\ Rc\ Rn\ \cos(\frac{4\ n\ f}{fd})\ fd^4 + 16\ C^2\ L^2\ Rc^2\ \cos(\frac{4\ n\ f}{fd})\ fd^4 - 64\ C^2\ L^2\ Rn^2\ \cos(\frac{2\ n\ f}{fd})\ fd^4 - 128\ C^2\ L^2\ Rc\ Rn\ \cos(\frac{2\ n\ f}{fd})\ fd^4 - 64\ C^2\ L^2\ Rc^2\ \cos(\frac{2\ n\ f}{fd})\ fd^4 + 48\ C^2\ L^2\ Rn^2\ fd^4 + 96\ C^2\ L^2\ Rc\ Rn\ fd^4 + 48\ C^2\ L^2\ Rc^2\ fd^4 - 4\ C^2\ Rc^2\ Rn^2\ \cos(\frac{4\ n\ f}{fd})\ fd^2 + 8\ C\ L\ Rn^2\ \cos(\frac{4\ n\ f}{fd})\ fd^2 - 4\ L^2\ \cos(\frac{4\ n\ f}{fd})\ fd^2 + 4\ C^2\ Rc^2\ Rn^2\ fd^2 - 8\ C\ L\ Rn^2\ fd^2 + 4\ L^2\ fd^2 + Rn^2\ \cos(\frac{4\ n\ f}{fd}) + 4\ Rn^2\ \cos(\frac{2\ n\ f}{fd}) + 3\ Rn^2)),$$

$$(2\ E\ L\ Rn\ fd\ (4\ C^2\ Rc\ Rn\ \sin(\frac{4\ n\ f}{fd})\ fd^2 + 4\ C^2\ Rc^2\ \sin(\frac{4\ n\ f}{fd})\ fd^2 - 8\ C^2\ Rc\ Rn\ \sin(\frac{2\ n\ f}{fd})\ fd^2 - 8\ C^2\ Rc^2\ \sin(\frac{2\ n\ f}{fd})\ fd^2 - \sin(\frac{4\ n\ f}{fd}) - 2\ \sin(\frac{2\ n\ f}{fd})))/(16\ C^2\ L^2\ Rn^2\ \cos(\frac{4\ n\ f}{fd})\ fd^4 + 32\ C^2\ L^2\ Rc\ Rn\ \cos(\frac{4\ n\ f}{fd})\ fd^4 + 16\ C^2\ L^2\ Rc^2\ \cos(\frac{4\ n\ f}{fd})\ fd^4 - 64\ C^2\ L^2\ Rn^2\ \cos(\frac{2\ n\ f}{fd})\ fd^4 - 128\ C^2\ L^2\ Rc\ Rn\ \cos(\frac{2\ n\ f}{fd})\ fd^4 - 64\ C^2\ L^2\ Rc^2\ \cos(\frac{2\ n\ f}{fd})\ fd^4 + 48\ C^2\ L^2\ Rn^2\ fd^4 + 96\ C^2\ L^2\ Rc\ Rn\ fd^4 + 48\ C^2\ L^2\ Rc^2\ fd^4 - 4\ C^2\ Rc^2\ Rn^2\ \cos(\frac{4\ n\ f}{fd})\ fd^2 + 8\ C\ L\ Rn^2\ \cos(\frac{4\ n\ f}{fd})\ fd^2 - 4\ L^2\ \cos(\frac{4\ n\ f}{fd})\ fd^2 + 4\ C^2\ Rc^2\ Rn^2\ fd^2 - 8\ C\ L\ Rn^2\ fd^2 + 4\ L^2\ fd^2 + Rn^2\ \cos(\frac{4\ n\ f}{fd}) + 4\ Rn^2\ \cos(\frac{2\ n\ f}{fd}) + 3\ Rn^2))]$$

АЧХ дискретной ПФ

$$(\sqrt{2}\ E\ Rn\ \sqrt{\cos(\frac{2\ n\ f}{fd}) + 1}\ \sqrt{-((4\ C^2\ Rc^2\ \cos(\frac{2\ n\ f}{fd}) - 4\ C^2\ Rc^2)\ fd^2) + \cos(\frac{2\ n\ f}{fd}) + 1})/\text{sqrt}($$

$$((16\ C^2\ L^2\ Rn^2 + 32\ C^2\ L^2\ Rc\ Rn + 16\ C^2\ L^2\ Rc^2)\ \cos(\frac{4\ n\ f}{fd}) + (-(64\ C^2\ L^2\ Rn^2) - 128\ C^2\ L^2\ Rc\ Rn - 64\ C^2\ L^2\ Rc^2)\ \cos(\frac{2\ n\ f}{fd}) + 48\ C^2\ L^2\ Rn^2 + 96\ C^2\ L^2\ Rc\ Rn + 48\ C^2\ L^2\ Rc^2)\ fd^4 +$$

$$(((8\ C\ L - 4\ C^2\ Rc^2)\ Rn^2 - 4\ L^2)\ \cos(\frac{4\ n\ f}{fd}) + (4\ C^2\ Rc^2 - 8\ C\ L)\ Rn^2 + 4\ L^2)\ fd^2 + Rn^2\ \cos(\frac{4\ n\ f}{fd}) + 4\ Rn^2\ \cos(\frac{2\ n\ f}{fd}) + 3\ Rn^2)$$

ФЧХ дискретной

$$\text{atan2}(-((((4\ C\ L\ Rn + 4\ C\ L\ Rc)\ \sin(\frac{4\ n\ f}{fd}) + (-(8\ C\ L\ Rn) - 8\ C\ L\ Rc)\ \sin(\frac{2\ n\ f}{fd}))\ fd^2) - (2\ C\ Rc\ Rn + 2\ L)\ \sin(\frac{4\ n\ f}{fd})\ fd - Rn\ \sin(\frac{4\ n\ f}{fd}) - 2\ Rn\ \sin(\frac{2\ n\ f}{fd}),$$

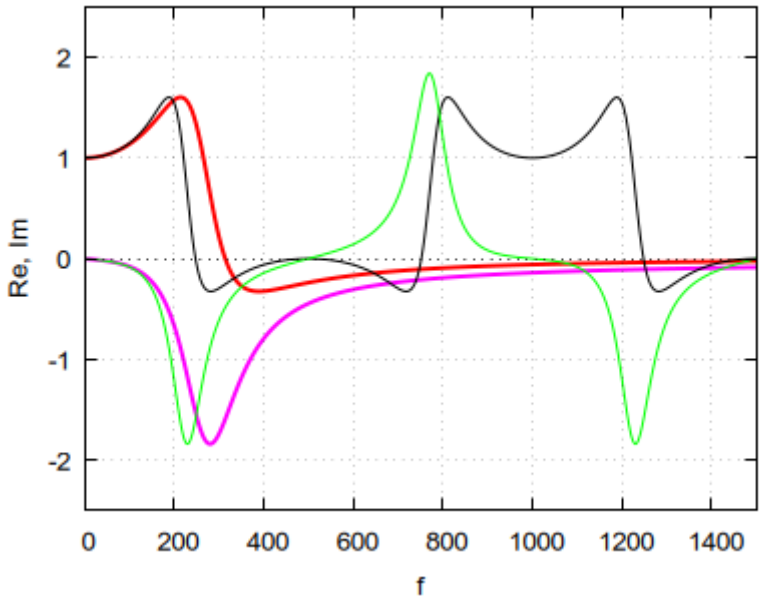
$$((4\ C\ L\ Rn + 4\ C\ L\ Rc)\ \cos(\frac{4\ n\ f}{fd}) + (-(8\ C\ L\ Rn) - 8\ C\ L\ Rc)\ \cos(\frac{2\ n\ f}{fd}) + 4\ C\ L\ Rn + 4\ C\ L\ Rc)\ fd^2 + ((2\ C\ Rc\ Rn + 2\ L)\ \cos(\frac{4\ n\ f}{fd}) - 2\ C\ Rc\ Rn - 2\ L)\ fd + Rn\ \cos(\frac{4\ n\ f}{fd}) + 2\ Rn\ \cos(\frac{2\ n\ f}{fd}) + Rn) +$$

$$\text{atan2}(\sin(\frac{2\ n\ f}{fd})\ (2\ C\ Rc\ fd + 1), (2\ C\ Rc\ \cos(\frac{2\ n\ f}{fd}) - 2\ C\ Rc)\ fd + \cos(\frac{2\ n\ f}{fd}) + 1) + \text{atan2}(\sin(\frac{2\ n\ f}{fd}), \cos(\frac{2\ n\ f}{fd}) + 1)$$

Задание численных параметров

$$[C = 5.1\ 10^{-4}, L = 6.3\ 10^{-4}, Rc = 0.5, Rn = 10, fd = 1000, E = 1]$$

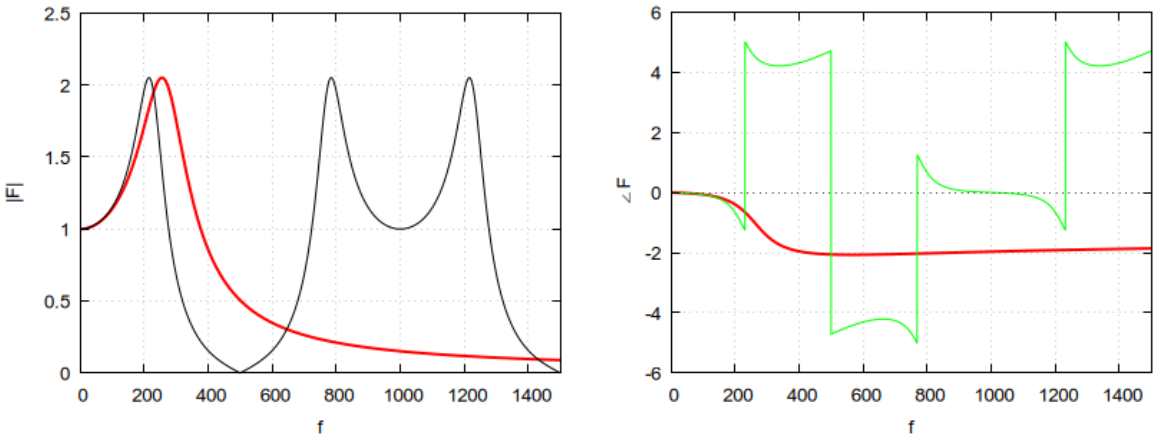
27. Построение совместных графиков для непрерывной и дискретизированной ПФ. Следует отметить, что график для дискретизированной функции симметричен относительно частоты Найквиста (500 в данном случае0 и повторяется относительно частоты дискретизации, равной 1000.



красная и лиловая — действительная и мнимая части для непрерывной

чёрная и зелёная — действительная и минимая части для дискретизированной

Аналогично, АЧХ и ФЧХ (красная- непрерывная)



На этом графике АЧХ хорошо видно смещение, характерное для применения подстановки Тастина. Величина смещения частоты (frequency warping) может быть оценена как $\omega_a = 2 \cdot fd \tan\left(\omega_d \frac{1}{2fd}\right)$.

Построение графика ФЧХ для непрерывной и дискретизированной функций: красный — для непрерывной, зелёный — для дискретизированной

Следует отметить, что фаза является непрерывной и необходимо применять отдельные алгоритмы коррекции переходов через значения кратные $\pi/2$

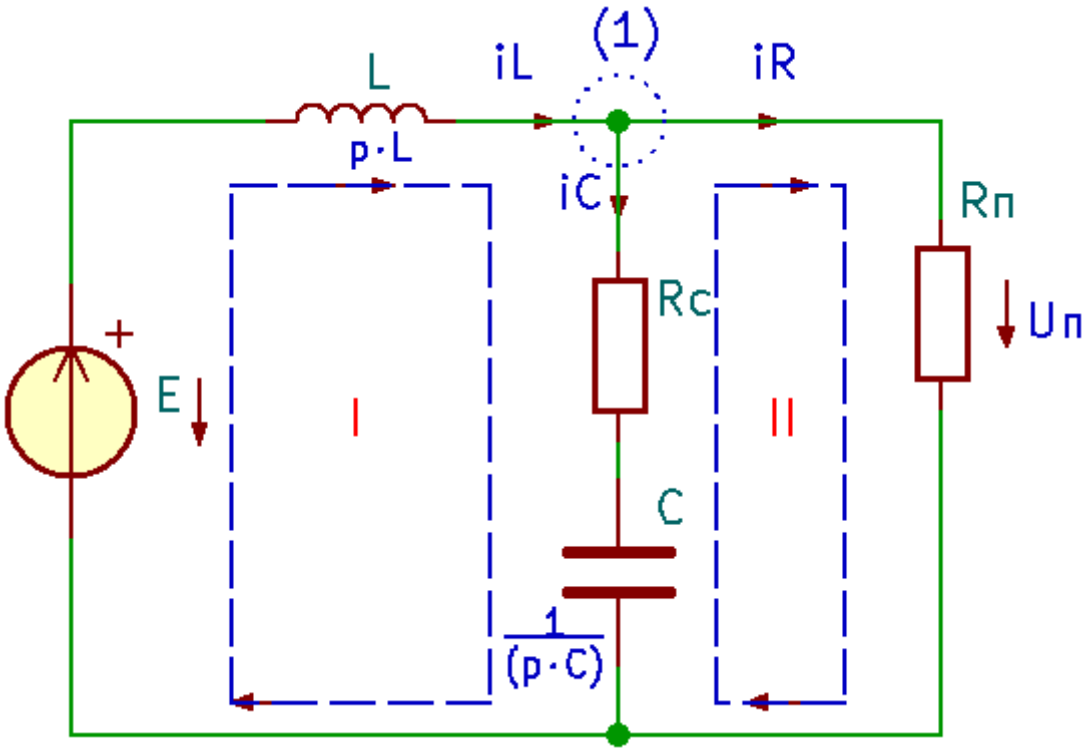
6.3 Дискретизация произвольной непрерывной передаточной функции методом согласования нулей и полюсов

28. При дискретизации методом согласования частотных характеристик можно избежать эффекта смещения частоты, однако, необходима коррекция амплитуды. Амплитуду можно скорректировать при наличии не нулевых нулей передаточной функции. Обычно используется для формирования критерия нулевая частота и частота Найквиста. Однако, можно использовать любые точки для нахождения корректирующего коэффициента.

При наличии нуля, включая кратного, можно использовать дополнительный критерий, например, скорость изменения комплексно-частотной характеристики, например, на нулевой частоте. Данный пример будет рассмотрен отдельно.

6.3.1 Дискретизация передаточной функции без наличия нулей равных нулю

Порядок дискретизации следующий. Повторим дискретизацию для схемы:



29. Задаём исходную систему уравнений по правилам Кирхгофа

$$\left[L\ iL\ p + \frac{iC}{C\ p} + Rc\ iC - E, -\left(\frac{iC}{C\ p}\right) + Rn\ iR - Rc\ iC, -iR + iL - iC, Un = Rn\ iR \right]$$

Используется метод нулей и полюсов для ПФ второго порядка. Дана передаточная функция в общем виде:

$$\frac{C\ E\ Rc\ Rn\ p+E\ Rn}{(C\ L\ Rn+C\ L\ Rc)\ p^2+(C\ Rc\ Rn+L)\ p+Rn}$$

Выделение числителя и знаменателя передаточной функции

$$C\ E\ Rc\ Rn\ p+E\ Rn$$
$$(C\ L\ Rn+C\ L\ Rc)\ p^2+(C\ Rc\ Rn+L)\ p+Rn$$

Нахождение корней для числителя и знаменателя

$$\left[p=-\left(\frac{1}{C\ Rc}\right)\right]$$
$$p=\frac{\pm\sqrt{(C^2\ Rc^2-4\ C\ L)\ Rn^2-2\ C\ L\ Rc\ Rn+L^2}-C\ Rc\ Rn-L}{2\ C\ L\ Rn+2\ C\ L\ Rc}$$

Переход от непрерывного корня к дискретному

ZP:exp(p/fd);

$$\%e^{\frac{p}{fd}}$$

30. Запись в общем виде числителя и знаменателя получаемой ПФ исходя из корней

$$z-\%e^{-\left(\frac{1}{C\ Rc\ fd}\right)}$$
$$\left(z-\%e^{\frac{\sqrt{(C^2\ Rc^2-4\ C\ L)\ Rn^2-2\ C\ L\ Rc\ Rn+L^2}-C\ Rc\ Rn-L}{(2\ C\ L\ Rn+2\ C\ L\ Rc)\ fd}}\right)\left(z-\%e^{-\left(\frac{\sqrt{(C^2\ Rc^2-4\ C\ L)\ Rn^2-2\ C\ L\ Rc\ Rn+L^2}+C\ Rc\ Rn+L}{(2\ C\ L\ Rn+2\ C\ L\ Rc)\ fd}\right)}\right)$$

Передаточная функция в общем виде. Производится домножение на корректирующий коэффициент k

$$\frac{k\left(z-\%e^{-\left(\frac{1}{C\ Rc\ fd}\right)}\right)}{\left(z-\%e^{\frac{\sqrt{(C^2\ Rc^2-4\ C\ L)\ Rn^2-2\ C\ L\ Rc\ Rn+L^2}-C\ Rc\ Rn-L}{(2\ C\ L\ Rn+2\ C\ L\ Rc)\ fd}}\right)\left(z-\%e^{-\left(\frac{\sqrt{(C^2\ Rc^2-4\ C\ L)\ Rn^2-2\ C\ L\ Rc\ Rn+L^2}+C\ Rc\ Rn+L}{(2\ C\ L\ Rn+2\ C\ L\ Rc)\ fd}\right)}\right)}$$

31. Чтобы избежать неточностей при наличии комплексно-сопряжённых величин, задаём численные параметры в виде целочисленных дробей

factor([C=510e-6,L=630e-6,Rc=0.5,Rn=20,fd=1000,E=1]);

$$\left[C=\frac{3\ 17}{2^5\ 5^5},L=\frac{3^2\ 7}{2^5\ 5^5},Rc=\frac{1}{2},Rn=2^2\ 5,fd=2^3\ 5^3,E=1\right]$$

Получаются комплексно-сопряжённые корни

$$\frac{\%e^{-\left(\frac{134000}{43911}\right)}\ k\left(\%e^{200/51}\ z-1\right)}{\left(\%e^{\frac{19100}{43911}}\ z-\%i\sin\left(\frac{100\sqrt{548999}}{43911}\right)-\cos\left(\frac{100\sqrt{548999}}{43911}\right)\right)\left(\%e^{\frac{19100}{43911}}\ z+\%i\sin\left(\frac{100\sqrt{548999}}{43911}\right)-\cos\left(\frac{100\sqrt{548999}}{43911}\right)\right)}$$

32. Обратить внимание - комплексно-сопряжённые корни при использовании целых чисел должны сократиться, как приведено далее

$$\frac{\%e^{-\left(\frac{137444}{43911}\right)}\ k\ \left(\%e^{4}\ z-\%e^{\frac{4}{51}}\right)}{\%e^{\frac{38200}{43911}}\ z^{2}-2\ \%e^{\frac{19100}{43911}}\ \cos\left(\frac{100\ \sqrt{548999}}{43911}\right)\ z+1}$$

33. Важным является следующее — при использовании плавающей запятой и символических вычислениях может оказаться, что присутствует «исчезающе малая» но не равная нулю, вернее, полностью сокращаемая, мнимая часть. Это происходит потому, что могут остаться значения, не равные точно тем, разность которых позволяет сократить, например, 1-0.999 = 0.001. Таким образом, может остаться малая, например $1\cdot10^{-18}$ мнимая часть, в то время как теоретически её быть не должно при раскрытии скобок при произведении комплексно-сопряжённых величин. Поэтому, если в оконечном выражении имеется мнимая часть, её необходимо проверить на малое значение и убрать командой взятия только действительной части realpart.

34. Условие - равенство на нулевой частоте значений для КЧХ непрерывной и дискретной ПФ

Значение на нулевой частоте для непрерывной ПФ, подставляем p=0

получаем 1

значение на нулевой частоте для дискретной ПФ получается равынм

$$\frac{\%e^{-\left(\frac{137444}{43911}\right)}\left(\%e^{4}-\%e^{\frac{4}{51}}\right)\ k}{-\left(2\ \%e^{\frac{19100}{43911}}\ \cos\left(\frac{100\ \sqrt{548999}}{43911}\right)\right)+\%e^{\frac{38200}{43911}}+1}$$

Приравниваем к значению для непрерывной равном 1 и разрешаем относительно *k*

$$\left[k=1.6012789035833999\right]$$

35. находим КЧХ для непрерывной и дискретной ПФ

$$-\left(\frac{1000000\ \left(51\ \%i\ \pi\ f+100000\right)}{131733\ \pi^{2}\ f^{2}-57300000\ \%i\ \pi\ f-100000000000}\right)$$

$$-\left(\frac{35567599\ \%e^{-\left(\frac{137444}{43911}\right)}\left(\%e^{\frac{\%i\ \pi\ f}{500}+4}-\%e^{\frac{4}{51}}\right)}{22211995\left(2\ \%e^{\frac{\%i\ \pi\ f}{500}+\frac{19100}{43911}}\ \cos\left(\frac{100\ \sqrt{548999}}{43911}\right)-\%e^{\frac{\%i\ \pi\ f}{250}+\frac{38200}{43911}}-1\right)}\right)$$

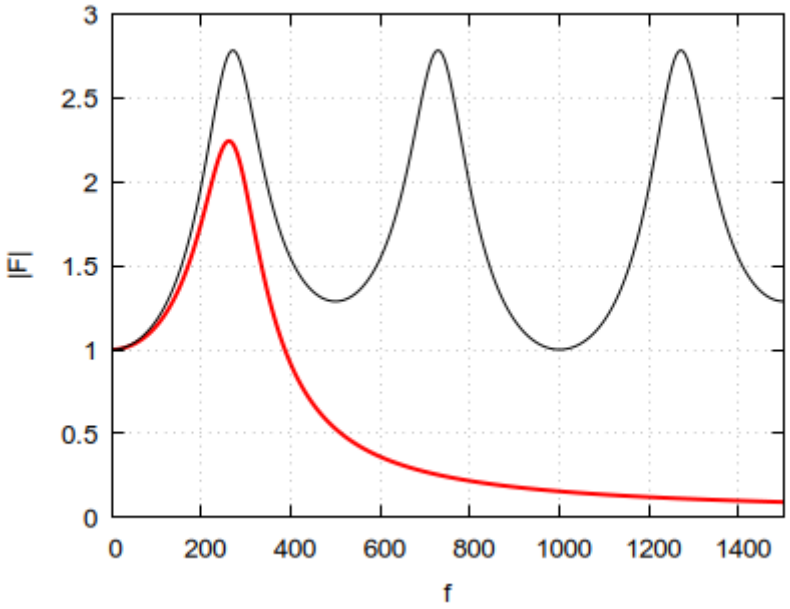
В численном виде АЧХ непрерывной

$$\frac{1000000\ \sqrt{2601\ \pi^{2}\ f^{2}+10000000000}}{\sqrt{\left(131733\ \pi^{2}\ f^{2}-100000000000\right)^{2}+3283290000000000\ \pi^{2}\ f^{2}}}$$

В численном виде АЧХ дискретизированной

```
(0.07000028939181281 sqrt(2982.1278220790164 - 118.10553074135807 cos(0.006283185307179587 f)))/sqrt(
1.3565059084780073 (1.2647407201693455 sin(0.006283185307179587 f) sin(0.012566370614359173 f)+
1.2647407201693455 cos(0.006283185307179587 f) cos(0.012566370614359173 f))+2.3867711577445694
(2.0 cos(0.012566370614359173 f)+0.05411953393769624)+0.7188071860327268 cos(0.006283185307179587 f)+
6.696676559441352)
```

36. Построение графика АЧХ для непрерывной и дискретизированной функций



В итоге получается передаточная функция в общем виде

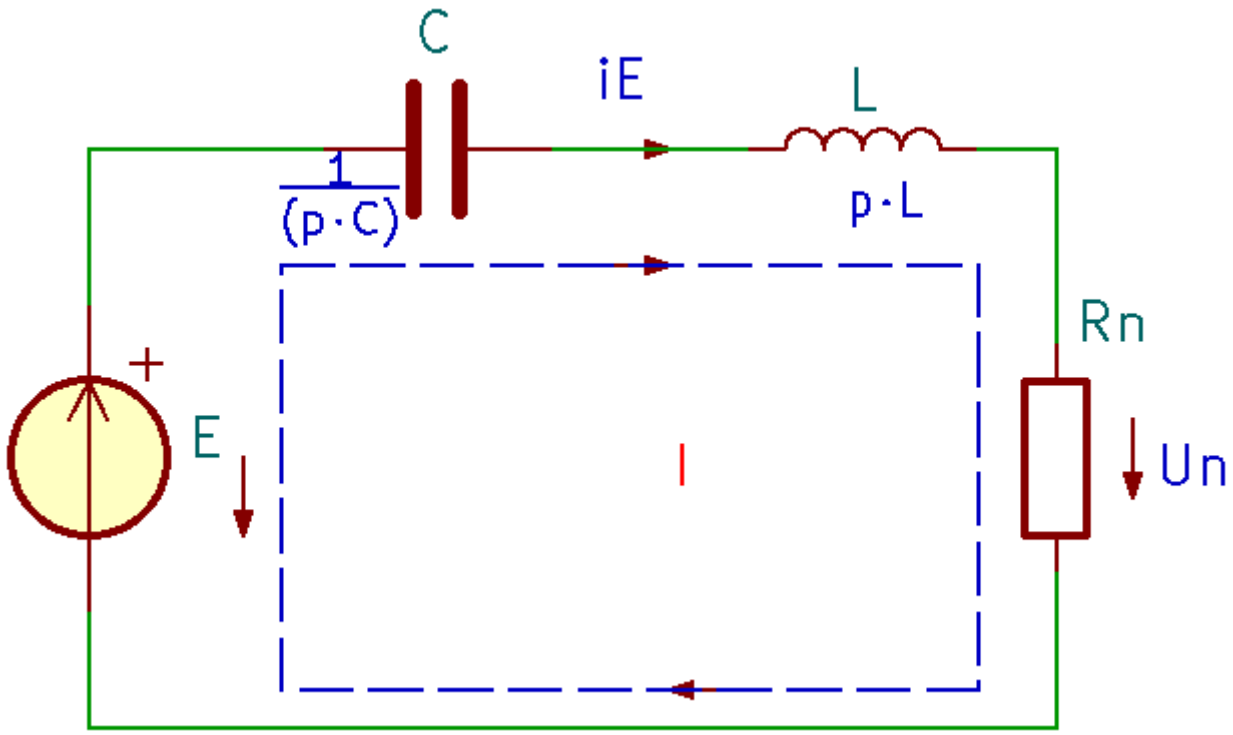
$$\frac{0.07000028939181281\,(54.598150033144236\,z - 1.0815891259105044)}{2.3867711577445694\,z^2 + 0.3594035930163634\,z + 1.0}$$

Можно подставить z=1 и убедиться в единичной АЧХ на нулевой частоте

1.0000000000000013

6.3.2 Дискретизация передаточной функции с нулём равным нулю

37. Пример дискретизации с «нулевым нулём», когда один из корней числителя равен нулю для непрерывной передаточной функции. В этом случае имеется особенность, что на постоянной составляющей определяемой нулевой частотой выходное значение равно нулю.



Дано: схема, составляется система уравнений

$$\left[L\ iE\ p + \frac{iE}{C\ p} + Rn\ iE - E, Un = Rn\ iE \right]$$

Решением будет являться выражение

$$\frac{C\ E\ Rn\ p}{C\ L\ p^2 + C\ Rn\ p + 1}$$

38. повторяем процедуру получения корней

Выделение числителя и знаменателя передаточной функции

$$C\ E\ Rn\ p$$

$$C\ L\ p^2 + C\ Rn\ p + 1$$

Нахождение корней для числителя и знаменателя

$$[p = 0]$$

$$\left[p = -\left(\frac{\sqrt{C}\ \sqrt{C\ Rn^2 - 4\ L} + C\ Rn}{2\ C\ L} \right), p = \frac{\sqrt{C}\ \sqrt{C\ Rn^2 - 4\ L} - C\ Rn}{2\ C\ L} \right]$$

39. Исходя из количества нулей и полюсов получается числитель и знаменатель дискретной передаточной функции соответственно исходя из найденных корней непрерывной

$$z-1$$

$$\left(z-\%e^{\frac{\sqrt{C}\,\sqrt{C\,Rn^2-4\,L}-C\,Rn}{2\,C\,L\,fd}}\right)\left(z-\%e^{-\left(\frac{\sqrt{C}\,\sqrt{C\,Rn^2-4\,L}+C\,Rn}{2\,C\,L\,fd}\right)}\right)$$

Передаточная функция в общем виде. Производится домножение на корректирующий коэффициент k

$$\frac{k\,(z-1)}{\left(z-\%e^{\frac{\sqrt{C}\,\sqrt{C\,Rn^2-4\,L}-C\,Rn}{2\,C\,L\,fd}}\right)\left(z-\%e^{-\left(\frac{\sqrt{C}\,\sqrt{C\,Rn^2-4\,L}+C\,Rn}{2\,C\,L\,fd}\right)}\right)}$$

40. Чтобы избежать громоздких выражений, задаём численные параметры

factor([C=1500e-6,L=1500e-6,Rn=0.5,fd=1000,E=1]);

$$\left[C=\frac{3}{2^4\,5^3},L=\frac{3}{2^4\,5^3},Rn=\frac{1}{2},fd=2^3\,5^3,E=1\right]$$

Переход к КЧХ для непрерывной и дискретной:

КЧХ непрерывной

$$\frac{2\,\%i\,\pi\,C\,E\,Rn\,f}{-\left(4\,\pi^2\,C\,L\,f^2\right)+2\,\%i\,\pi\,C\,Rn\,f+1}$$

КЧХ дискретной

$$\frac{\left(\%e^{\frac{2\,\%i\,\pi\,f}{fd}}-1\right)k}{\left(\%e^{\frac{2\,\%i\,\pi\,f}{fd}}-\%e^{\frac{\sqrt{C}\,\sqrt{C\,Rn^2-4\,L}-C\,Rn}{2\,C\,L\,fd}}\right)\left(\%e^{\frac{2\,\%i\,\pi\,f}{fd}}-\%e^{-\left(\frac{\sqrt{C}\,\sqrt{C\,Rn^2-4\,L}+C\,Rn}{2\,C\,L\,fd}\right)}\right)}$$

41. Производная непрерывной КЧХ для непрерывной ПФ

$$\frac{2\,\%i\,\pi\,C\,E\,Rn\left(4\,\pi^2\,C\,L\,f^2+1\right)}{\left(4\,\pi^2\,C\,L\,f^2-2\,\%i\,\pi\,C\,Rn\,f-1\right)^2}$$

Значение производной непрерывной в нуле

$$2\,\%i\,\pi\,C\,E\,Rn$$

производная дискретной КЧХ

$$-\left(\left(2\,\%e^{\frac{2\,\%i\,n\,f}{fd}+\frac{\sqrt{C\,Rn^2-4\,L}}{2\sqrt{C}\,L\,fd}+\frac{Rn}{L\,fd}}\left(\%e^{\frac{4\,\%i\,n\,f}{fd}+\frac{\sqrt{C\,Rn^2-4\,L}}{2\sqrt{C}\,L\,fd}+\frac{Rn}{L\,fd}}-2\,\%e^{\frac{2\,\%i\,n\,f}{fd}+\frac{\sqrt{C\,Rn^2-4\,L}}{2\sqrt{C}\,L\,fd}+\frac{Rn}{L\,fd}}+\%e^{\frac{\sqrt{C\,Rn^2-4\,L}}{\sqrt{C}\,L\,fd}+\frac{Rn}{2\,L\,fd}}-\%e^{\frac{\sqrt{C\,Rn^2-4\,L}}{2\sqrt{C}\,L\,fd}}+\%e^{\frac{Rn}{2\,L\,fd}}\right)\%i\,\pi\,k\right)/\left(\left(\%e^{\frac{2\,\%i\,n\,f}{fd}+\frac{Rn}{2\,L\,fd}}-\%e^{\frac{\sqrt{C\,Rn^2-4\,L}}{2\sqrt{C}\,L\,fd}}\right)^2\left(\%e^{\frac{2\,\%i\,n\,f}{fd}+\frac{\sqrt{C\,Rn^2-4\,L}}{2\sqrt{C}\,L\,fd}+\frac{Rn}{2\,L\,fd}}-1\right)^2fd\right)\right)$$

Значение производной в нулевой точке частоты для дискретной

$$-\left(\frac{2\,\%e^{\frac{\sqrt{C\,Rn^2-4\,L}}{2\sqrt{C}\,L\,fd}+\frac{Rn}{L\,fd}}\left(\%e^{\frac{\sqrt{C\,Rn^2-4\,L}}{\sqrt{C}\,L\,fd}+\frac{Rn}{2\,L\,fd}}-\%e^{\frac{\sqrt{C\,Rn^2-4\,L}}{2\sqrt{C}\,L\,fd}+\frac{Rn}{L\,fd}}-\%e^{\frac{\sqrt{C\,Rn^2-4\,L}}{2\sqrt{C}\,L\,fd}}+\%e^{\frac{Rn}{2\,L\,fd}}\right)\%i\,\pi\,k}{\left(\%e^{\frac{\sqrt{C\,Rn^2-4\,L}}{2\sqrt{C}\,L\,fd}}-\%e^{\frac{Rn}{2\,L\,fd}}\right)^2\left(\%e^{\frac{\sqrt{C\,Rn^2-4\,L}}{2\sqrt{C}\,L\,fd}+\frac{Rn}{2\,L\,fd}}-1\right)^2fd}\right)$$

Для упрощения применяем численные параметры;

производная КЧХ в точке 0 непрерывная (численно) $0.00471238898038469\,\%i$

производная КЧХ в точке 0 для дискретной (численно)

$$-((0.008768891483818188\,(-(0.3057726152603939\,\%i)-0.4060058810716169)\,(0.6015956960564122\,\%i+0.7988007376601508)\,\%i\,k)/((0.6015956960564122\,\%i-0.38255967520549505)^2\,(0.7107013398713987\,\%i-0.05632843076042182)^2))$$

42. нахождение корректирующего коэффициента приравниванием непрерывной и дискретной

$$[k=0.2731431435426538]$$

Итоговая дискретная ПФ в общем виде:

$$\frac{32507981\,\%e^{\frac{\sqrt{5}\,\%i}{2\sqrt{3}}+\frac{1}{3}}\,(z-1)}{119014450\left(\%e^{\frac{1}{6}}\,z-\%e^{\frac{\sqrt{5}\,\%i}{2\sqrt{3}}}\right)\left(\%e^{\frac{\sqrt{5}\,\%i}{2\sqrt{3}}+\frac{1}{6}}\,z-1\right)}$$

Построение графиков

находим численные КЧХ для непрерывной и дискретной ПФ

$$-\left(\frac{1500\,\%i\,\pi\,f}{9\,\pi^2\,f^2-1500\,\%i\,\pi\,f-1000000}\right)$$

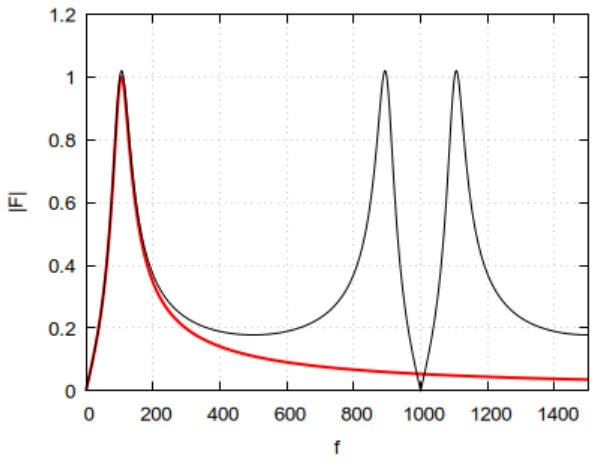
$$\frac{32507981\,\%e^{\frac{\sqrt{5}\,\%i}{2\sqrt{3}}+\frac{1}{3}}\left(\%e^{\frac{\%i\,\pi\,f}{500}}-1\right)}{119014450\left(\%e^{\frac{\%i\,\pi\,f}{500}+\frac{1}{6}}-\%e^{\frac{\sqrt{5}\,\%i}{2\sqrt{3}}}\right)\left(\%e^{\frac{\%i\,\pi\,f}{500}+\frac{\sqrt{5}\,\%i}{2\sqrt{3}}+\frac{1}{6}}-1\right)}$$

АЧХ для непрерыывной и дискретизированной соответственно

$$\frac{1500\,\pi\,f}{\sqrt{(9\,\pi^2\,f^2-1000000)^2+2250000\,\pi^2\,f^2}}$$

$$(0.381201964955201\sqrt{2.0-2.0\cos(0.006283185307179587\,f)})/(\mathrm{sqrt}(1.1813604128656459\,(-(1.2031913921128243\sin(0.006283185307179587\,f))-1.5976014753203016\cos(0.006283185307179587\,f))+2.3956124250860897)\sqrt{2.3956124250860897-2.3627208257312917\cos(0.0011547005383792518\,(5.441398092702653\,f+559.0169943749476))})$$

43. Построение графика АЧХ для непрерывной (красная линия) и дискретизированной функций. Видно соответствие по частоте и равенство производных вблизи 0 (скорость роста кривой)



6.4 Использование непосредственного задания полюсов в виде комплексно-сопряжённых значений

В данном разделе рассматривается непосредственное задание мнимой и дествительной частей комплексно-сопряженных корней, отвечающих за числитель и/или знаменатель дискретной передаточной функции.

44. задаём передаточную функцию в общем виде как отношение корней нулей и полюсов

$W_1 = \frac{\prod_{i=1}^{N} (z - z_n[i])}{\prod_{i=1}^{D} (z - z_d[i])}$, для второго порядка получается $\frac{(z - zn_1)(z - zn_2)}{(z - zd_1)(z - zd_2)}$

Предположим что корни комплексно-сопряжённые. x - действительная часть, y — мнимая. для действительных корней y = 0, для чисто мнимых x = 0, будем рассматривать только знаменатель.

45. Выражение для корней знаменателя

$$\left[zd_1 = \%i\ y + x,\ zd_2 = x - \%i\ y\right]$$

Получаемая передаточная функция с комплексно-сопряжёнными корнями

$$\frac{(z - zn_1)(z - zn_2)}{(z - \%i\ y - x)(z + \%i\ y - x)}$$

Производится сокращение мнимой части при раскрытии скобок для комплексно-сопряжённых величин:

$$\frac{(z - zn_1)(z - zn_2)}{z^2 - 2\ x\ z + y^2 + x^2}$$

Переход в КЧХ с подстановкой $z = \%e^{\frac{2\ \%i\ \pi\ f}{fd}}$

$$\frac{\left(\%e^{\frac{2\ \%i\ \pi\ f}{fd}} - zn_1\right)\left(\%e^{\frac{2\ \%i\ \pi\ f}{fd}} - zn_2\right)}{y^2 + x^2 - 2\ \%e^{\frac{2\ \%i\ \pi\ f}{fd}}\ x + \%e^{\frac{4\ \%i\ \pi\ f}{fd}}}$$

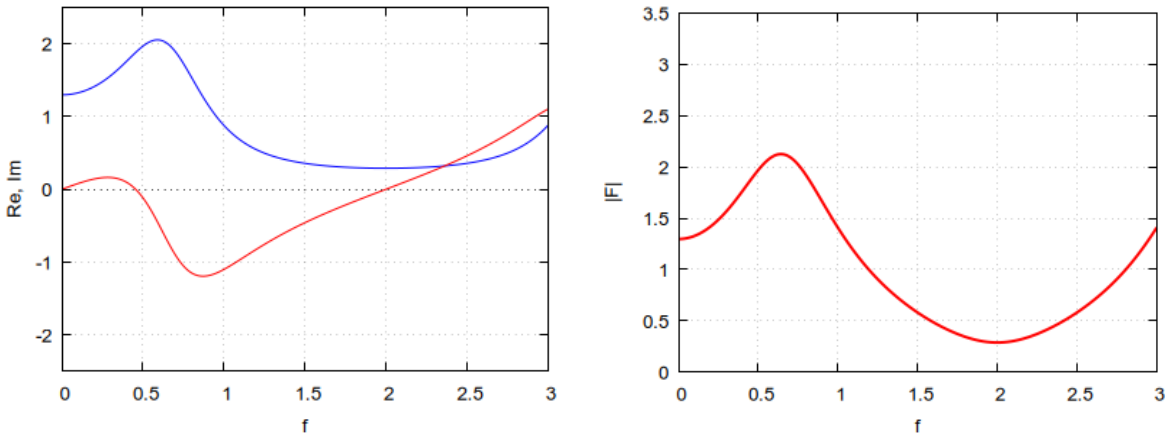
46. После преобразований и подстановки получается

$$\frac{\%i \sin\left(\frac{4\,\pi\,f}{fd}\right) + \cos\left(\frac{4\,\pi\,f}{fd}\right) - \%i\ zn_2 \sin\left(\frac{2\,\pi\,f}{fd}\right) - \%i\ zn_1 \sin\left(\frac{2\,\pi\,f}{fd}\right) - zn_2 \cos\left(\frac{2\,\pi\,f}{fd}\right) - zn_1 \cos\left(\frac{2\,\pi\,f}{fd}\right) + zn_1\ zn_2}{y^2 + x^2 - 2\ \%i \sin\left(\frac{2\,\pi\,f}{fd}\right) x - 2\cos\left(\frac{2\,\pi\,f}{fd}\right) x + \%i \sin\left(\frac{4\,\pi\,f}{fd}\right) + \cos\left(\frac{4\,\pi\,f}{fd}\right)}$$

АЧХ как модуль КЧХ:

$$\text{sqrt}\left(2\,zn_1\,zn_2\cos\left(\frac{4\,\pi\,f}{fd}\right)-2\,zn_1\,zn_2^2\cos\left(\frac{2\,\pi\,f}{fd}\right)-2\,zn_1^2\,zn_2\cos\left(\frac{2\,\pi\,f}{fd}\right)-2\,zn_2\cos\left(\frac{2\,\pi\,f}{fd}\right)-2\,zn_1\cos\left(\frac{2\,\pi\,f}{fd}\right)+zn_1^2\,zn_2^2+zn_2^2+2\,zn_1\,zn_2+zn_1^2+1\right)/$$
$$\sqrt{y^4+2\,x^2\,y^2-4\cos\left(\frac{2\,\pi\,f}{fd}\right)x\,y^2+2\cos\left(\frac{4\,\pi\,f}{fd}\right)y^2+x^4-4\cos\left(\frac{2\,\pi\,f}{fd}\right)x^3+2\cos\left(\frac{4\,\pi\,f}{fd}\right)x^2+4\,x^2-4\cos\left(\frac{2\,\pi\,f}{fd}\right)x+1}$$

Графики КЧХ и АЧХ



47. Запись дискретной передаточной функции в формате цифрового фильтра.

Производится подстановка $z=1/zm$

$$\frac{\left(zn_1\;zm-1\right)\left(zn_2\;zm-1\right)}{y^2\;zm^2+x^2\;zm^2-2\;x\;zm+1}$$

Выделяется числитель и знаменатель

$$zn_1\;zn_2\;zm^2-zn_2\;zm-zn_1\;zm+1$$

$$y^2\;zm^2+x^2\;zm^2-2\;x\;zm+1$$

Находятся коэффициенты цифрового фильтра, коэффициенты знаменателя изначально нормированы (свободный элемент равен 1, при 0-й степени zm, $zm^0=1$)

коэффициенты числителя

$$\left[1,-zn_2-zn_1,zn_1\;zn_2\right]$$

коэффициенты знаменателя

$$\left[1,-\left(2\;x\right),y^2+x^2\right]$$

в численном виде для построения характеристик с отсчётами зададим полюса— значения мнимой и действительной частей, для нулей — действительные части.

$$\left[zn_1=0.4,zn_2=-0.6,x=0.3,y=0.5,fd=4\right]$$

Коэффициенты, соответствующие полюсам и нулям для числителя и знаменателя соответственно

$$\left[1,0.19999999999999996,-0.24\right]$$

$$\left[1,-0.6,0.33999999999999997\right]$$

7 Полезные утилиты и вспомогательные скрипты

В Приложении 1 представлены Markdown-документы, на основе которых могут быть сформированы утилиты для работы с xPack, преобразования изображений, обработка результатов тестирования, непосредственно сами тесты цифровых фильтров.

Код 1. Markdown по Python-скрипту, формирующего графики по выводу тестов цифровых фильтров

```
# Анализ скрипта plot_tests.py

Данный документ представляет собой подробное описание работы скрипта `plot_tests.py`, который предназначен для парсинга результатов тестов цифрового фильтра из текстового файла и визуализации этих данных в виде графиков. Скрипт обрабатывает коэффициенты фильтра, временные отсчёты и строит диаграмму нулей и полюсов на комплексной плоскости.

## 1. Обработка данных и регулярные выражения

### Основные компоненты

Скрипт использует следующие фреймворки и модули:
- `matplotlib.pyplot` — для построения графиков.
- `numpy` — для численных вычислений.
- `re` — для обработки текста с помощью регулярных выражений.
- `os` — для проверки существования файлов.

### Формат входных данных

Входные данные ожидается в файле `test_output.txt`. Файл содержит несколько тестов, каждый из которых начинается с заголовка вида `[ТЕСТ 1.1]`.

### Регулярные выражения

Скрипт активно использует регулярные выражения для извлечения данных из текстового файла:

1. **Разделение на блоки тестов**:
```python
test_blocks = re.split(r'\[ТЕСТ[\s\d\.]+.*?\]', content)
test_names = re.findall(r'\[ТЕСТ[\s\d\.]+.*?\]', content)
```
Эти выражения разделяют файл на отдельные блоки тестов и извлекают их названия.

2. **Извлечение коэффициентов `b` и `a`**:
```python
b_matches = re.findall(r'b([0-2]):\s*([+-]?\d*\.?\d+)', block)
a_matches = re.findall(r'a([1-2]):\s*([+-]?\d*\.?\d+)', block)
```
Эти выражения находят коэффициенты фильтра, такие как `b0`, `b1`, `b2`, `a1`, `a2`, и извлекают их индексы и значения.

3. **Парсинг строк с временными отсчётами**:
```

- Для строк вида `Шаг 0 | Вход: 0.7424 | Выход: 0.2474`:
```python
step_out_match = re.match(r'Шаг\s+(\d+)\s+\|\s+(?:Вход:\s+([-\d\.]+)\s+\|\s+)?Выход:\s+([-\d\.]+)', line)
```
- Для строк вида `0 | 0.7424 | 0.2474`:
```python
simple_match = re.match(r'(\d+)\s+\|\s+([-\d\.]+)\s+\|\s+([-\d\.]+)(?:\s*\|\s*.+)*$', line)
```

4. **Определение заголовков таблиц**:
Скрипт определяет строки-заголовки (например, `Шаг | Вход | Выход`) с помощью проверки наличия ключевых слов (`Step`, `Вход`, `Output` и т.д.) и символа `|`.

Обработка многоканальных данных

Скрипт определяет, содержит ли тест многоканальные данные, по наличию слов `Канал`, `Channel`, `Ch` и т.п. Для таких данных используется универсальный парсер, который разбивает строки по символу `|` и извлекает числовые значения, игнорируя нечисловые поля.

2. Вычисление нулей и полюсов

Скрипт вычисляет нули и полюса передаточной функции фильтра с помощью функции `compute_poles_and_zeros(b, a)`.

Алгоритм

1. **Нормализация коэффициентов**:
Передаточная функция нормализуется так, чтобы коэффициент `a0` был равен 1:
```python
if a[0] != 1:
b = b / a[0]
a = a / a[0]
```

2. **Вычисление нулей**:
Нули — это корни полинома числителя. Поскольку коэффициенты `b` заданы в порядке `b0 + b1*z^-1 + b2*z^-2`, для нахождения нулей используется обратный порядок коэффициентов:
```python
zeros = np.roots(b[::-1])
```

3. **Вычисление полюсов**:
Полюса — это корни полинома знаменателя: `1 + a1*z^-1 + a2*z^-2`. Для вычисления используется полином вида `z^2 + a1*z + a2`, что соответствует:
```python
poles = np.roots([1] + a[1:].tolist())
```

Особенности

- Если коэффициенты отсутствуют, возвращаются пустые массивы.
- Обработка пустых или нулевых коэффициентов выполнена корректно.

3. Построение графиков

Каждый тест визуализируется на одном графике с двумя подграфиками.

Левый подграфик: временные отсчёты

- **Одноканальные данные**:
- Входной сигнал (`input`) — чёрный цвет, маркер `o`.
- Выходной сигнал (`output`) — зелёный цвет, маркер `s`.

- **Многоканальные данные**:
- Каждый канал имеет свой цвет и маркер.
- Используется автоматическое определение типа сигнала (ступень, импульс, синус) на основе названия канала или позиции.
- Легенда включает названия каналов и параметры фильтра.

Правый подграфик: нули и полюса

- **Единичная окружность** — серая пунктирная линия.
- **Нули** — синие круги с чёрной границей.
- **Полюса** — красные крестики.
- Ось X — действительная часть, ось Y — мнимая часть.
- График масштабируется с учётом всех нулей и полюсов и добавляет небольшое поле вокруг.

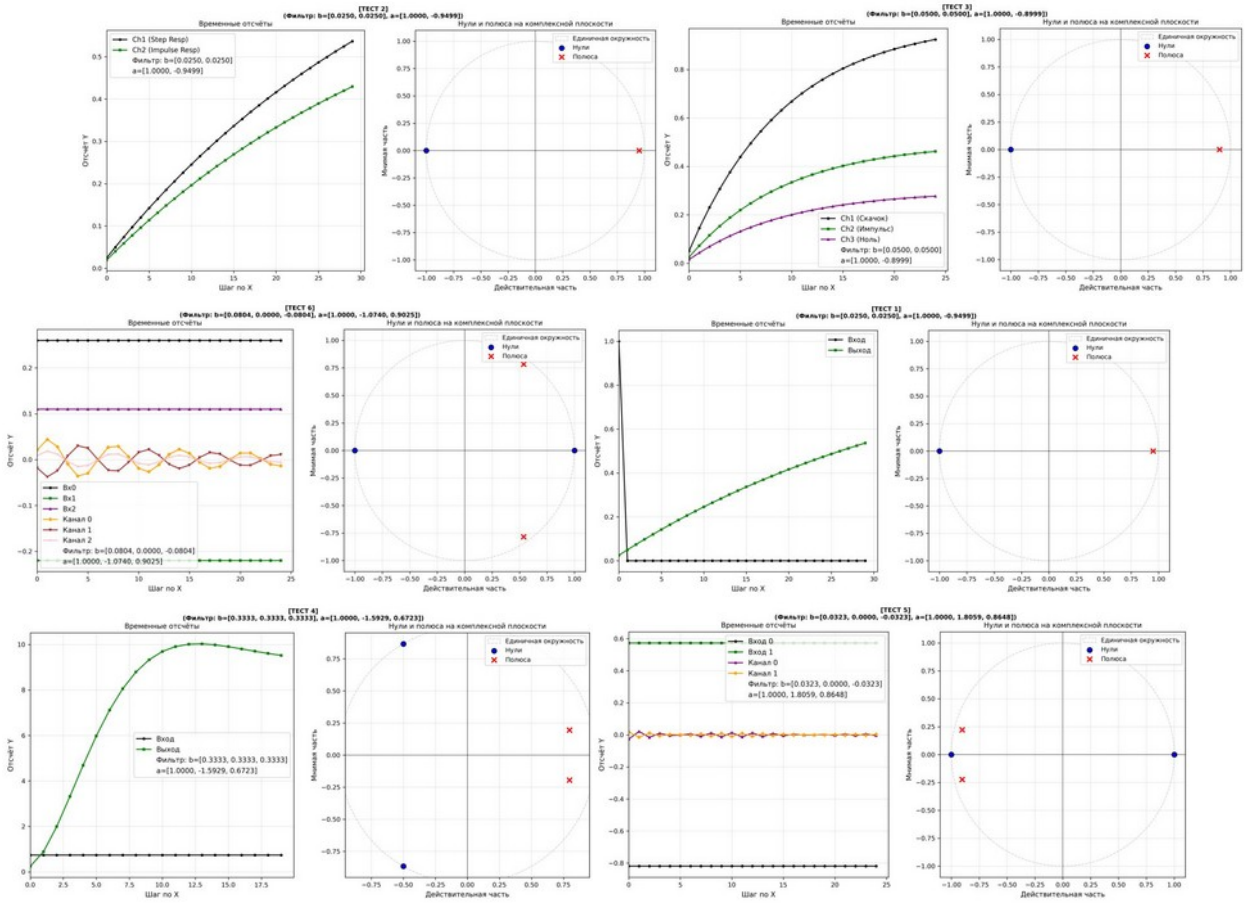
Форматирование

- Заголовок графика содержит название теста и коэффициенты фильтра.
- Размер шрифта — 10.
- Сетка — полупрозрачная, для улучшения читаемости.
- `plt.tight_layout()` и `plt.subplots_adjust()` используются для оптимального размещения элементов.

Заключение

Скрипт `plot_tests.py` представляет собой мощный инструмент для визуализации результатов тестирования цифровых фильтров. Он эффективно использует регулярные выражения для парсинга разнородных текстовых данных, корректно вычисляет нули и полюса передаточной функции и строит информативные графики, позволяющие анализировать поведение фильтра во временной и частотной областях.

Пример вывода графиков:



Код 2. Markdown кода Maxima по реализации цифрового фильтра

 # Анализ циклов for в DigFilterOverallExCoeff.mac

Данный документ предоставляет детальное описание циклов `for`, используемых в файле DigFilterOverallExCoeff.mac для реализации умножения с накоплением и сдвига буферов при цифровой фильтрации.

Синтаксис цикла for в Maxima

В Maxima цикл `for` используется для повторения операций и имеет следующие основные формы:
- `for i: start thru end do body` — цикл с инкрементом на 1
- `for i: start step step_size thru end do body` — цикл с заданным шагом
- `for i in list do body` — итерация по элементам списка

Все циклы в DigFilterOverallExCoeff.mac используют первую форму с инкрементом на 1.

Цикл умножения с накоплением

Реализация в DigFiltStep

```maxima
for i:1 thru Nx do
(Acc:float(Acc+X[i]*Cx[i])),
for i:1 thru Ny do
(Acc:float(Acc-Y[i]*Cy[i]))
```

Эти два последовательных цикла реализуют основную вычислительную часть цифрового фильтра — умножение с накоплением (MAC operations).

Пошаговое выполнение первого цикла

Цель: Вычисление взвешенной суммы входных значений (часть числителя передаточной функции)

1. **Инициализация**: Переменная `i` устанавливается в 1
2. **Проверка условия**: Проверяется, что `i <= Nx` (длина коэффициентов числителя)
3. **Выполнение тела цикла**:
- Вычисление произведения `X[i] * Cx[i]` (входное значение на коэффициент)
- Прибавление результата к аккумулятору `Acc`
- Преобразование результата в число с плавающей точкой с помощью `float()`
4. **Инкремент**: Переменная `i` увеличивается на 1
5. **Повторение**: Возврат к шагу 2, пока условие истинно

Математическая интерпретация:
```
Acc = Σ(X[i] * Cx[i]) для i = 1..Nx
```

Это классическая операция свертки, используемая в цифровой обработке сигналов.

Пошаговое выполнение второго цикла

Цель: Вычитание взвешенной суммы предыдущих выходных значений (часть знаменателя передаточной функции)

1. **Инициализация**: Переменная `i` устанавливается в 1
2. **Проверка условия**: Проверяется, что `i <= Ny` (длина коэффициентов знаменателя)
3. **Выполнение тела цикла**:
- Вычисление произведения `Y[i] * Cy[i]` (предыдущее выходное значение на коэффициент)
- Вычитание результата из аккумулятора `Acc`
- Преобразование результата в число с плавающей точкой
4. **Инкремент**: Переменная `i` увеличивается на 1
5. **Повторение**: Возврат к шагу 2, пока условие истинно

Математическая интерпретация:
```
Acc = Acc - Σ(Y[i] * Cy[i]) для i = 1..Ny
```

Особенности реализации

1. **Использование float()**: Каждое промежуточное вычисление преобразуется в число с плавающей точкой, что предотвращает накопление символьных выражений и обеспечивает численную стабильность.

2. **Последовательное выполнение**: Циклы выполняются один за другим, что соответствует математической структуре разностного уравнения IIR-фильтра.

3. **Оптимизация порядка операций**: Сначала вычисляется сумма по входным значениям, затем вычитается сумма по выходным значениям, что минимизирует количество операций.

Циклы сдвига буферов

Сдвиг входного буфера

```maxima
T1:X[1],
for i:2 thru Nx do
(T2:X[i],X[i]:T1,T1:T2)
```

Цель: Сдвиг элементов входного буфера вправо для освобождения места под новое значение.

Пошаговое выполнение:

1. **Инициализация**: Первый элемент буфера `X[1]` сохраняется во временной переменной `T1`
2. **Цикл**: Для каждого индекса `i` от 2 до `Nx`:
- Текущее значение `X[i]` сохраняется во временной переменной `T2`
- Предыдущее значение `T1` записывается в `X[i]`
- Переменная `T1` обновляется значением `T2` (текущим значением элемента)

Визуализация процесса:

Исходный буфер: $[x_0, x_1, x_2, x_3, ..., x_{n-1}]$

После сдвига: $[x_0, x_0, x_1, x_2, ..., x_{n-2}]$

Где x_0 — новое входное значение, которое будет записано в `X[1]` перед выполнением цикла сдвига.

Сдвиг выходного буфера

```maxima
T1:Y[1],
for i:2 thru Ny do
(T2:Y[i],Y[i]:T1,T1:T2)
```

```

**Цель**: Сдвиг элементов выходного буфера вправо для хранения истории выходных значений.

Процесс идентичен сдвигу входного буфера, но применяется к выходному буферу `Y`.

### Алгоритм сдвига: детальный анализ

**Механизм двух переменных**:

Алгоритм использует две временные переменные (`T1` и `T2`) для реализации сдвига:

- `T1` — хранит значение, которое нужно записать в текущий элемент
- `T2` — временно хранит текущее значение элемента перед его перезаписью

**Пошаговый пример для буфера [A, B, C, D]**:

1. `T1:A` (сохранение первого элемента)
2. `i=2`: `T2:B`, `X[2]:A`, `T1:B` → результат: [A, A, C, D]
3. `i=3`: `T2:C`, `X[3]:B`, `T1:C` → результат: [A, A, B, D]
4. `i=4`: `T2:D`, `X[4]:C`, `T1:D` → результат: [A, A, B, C]

**Инвариант цикла**: После каждой итерации `T1` содержит значение исходного элемента `X[i-1]`, которое будет записано в следующий элемент.

## Взаимодействие циклов в DigFiltStep

Циклы в функции `DigFiltStep` работают в следующей последовательности:

1. **Установка нового значения**: `X[1]:InputX` — новое входное значение помещается в первый элемент входного буфера
2. **Умножение с накоплением**: Два цикла вычисляют выходное значение фильтра
3. **Сохранение результата**: `Y[1]:Acc` — вычисленное значение сохраняется как первое выходное значение
4. **Сдвиг буферов**: Два цикла сдвига перемещают значения в буферах, готовя их к следующему шагу

## Эффективность реализации

### Вычислительная сложность

- **Умножение с накоплением**: O(Nx + Ny) — линейная сложность по длине фильтра
- **Сдвиг буферов**: O(Nx + Ny) — также линейная сложность
- **Общая сложность одного шага**: O(Nx + Ny)

### Оптимизационные аспекты

1. **Минимизация временных переменных**: Используются только необходимые временные переменные
```

2. **Последовательный доступ к памяти**: Циклы проходят по массивам последовательно, что хорошо для кэширования
3. **Отсутствие рекурсии**: Используются итеративные циклы, что эффективнее для численных вычислений

Заключение

Циклы `for` в DigFilterOverallExCoeff.mac реализуют две ключевые операции цифровой фильтрации:

1. **Умножение с накоплением** — основная вычислительная операция, реализующая разностное уравнение фильтра
2. **Сдвиг буферов** — операция обновления состояния фильтра для последующих шагов

Такая реализация демонстрирует эффективное использование базовых конструкций Maxima для решения задач цифровой обработки сигналов, сочетая математическую точность с вычислительной эффективностью.

Код 3. Markdown кода Maxima по дискретизации

Резюме Maxima-файла: TransferFuncs1and2ord.mac

Общее описание

Файл `TransferFuncs1and2ord.mac` написан на языке Maxima и предназначен для моделирования и анализа передаточных функций (ПФ) электрических цепей. Основная цель — сравнение непрерывных и дискретизированных систем, а также исследование методов дискретизации, таких как преобразование Тастина.

Файл разбит на несколько логических частей:
1. **Дискретизация ПФ первого порядка** на примере RC-цепи.
2. **Дискретизация ПФ второго порядка** на примере RLC-цепи.
3. **Метод нулей и полюсов** для синтеза дискретной ПФ второго порядка.
4. **Пример с нулевым нулём** и настройка коэффициента усиления по производной.
5. **Общий вид цифрового фильтра** второго порядка.

---

1. Дискретизация ПФ первого порядка (RC-цепь)

Исходные данные
* **Схема:** Последовательная цепь из источника напряжения `E`, резистора `R` и конденсатора `C`.
* **Выход:** Напряжение на конденсаторе `Un`.

Процесс
1. **Составление системы уравнений:** Используется закон Кирхгофа для тока и

напряжения.
2. **Решение системы:** Maxima решает систему относительно переменной состояния `Un`, получая непрерывную передаточную функцию `Un(p) = E / (R*C*p + 1)`.
3. **Дискретизация:** К непрерывной ПФ применяется билинейное преобразование (Тастина) `p = 2*fd*(z-1)/(z+1)`, где `fd` — частота дискретизации.
4. **Построение характеристик:** Строится совместный график:
* **КЧХ (Комплексная частотная характеристика):** Действительная и мнимая части для непрерывной и дискретной ПФ.
* **АЧХ (Амплитудно-частотная характеристика):** Модуль передаточной функции.
* **ФЧХ (Фазо-частотная характеристика):** Аргумент передаточной функции.

Вывод: Визуализация позволяет убедиться в том, что дискретная модель корректно аппроксимирует поведение непрерывной системы в заданном диапазоне частот.

---

2. Дискретизация ПФ второго порядка (RLC-цепь)

Исходные данные
* **Схема:** Более сложная RLC-цепь (два контура).
* **Параметры:** `Rc`, `Rn`, `L`, `C`.

Процесс
1. **Составление системы уравнений:** Применяются законы Кирхгофа для двух контуров и первого узла.
2. **Решение системы:** Maxima решает систему, получая более сложную непрерывную ПФ второго порядка.
3. **Дискретизация:** Снова применяется преобразование Тастина.
4. **Построение характеристик:** Аналогично пункту 1, строятся совместные графики КЧХ, АЧХ и ФЧХ.

Вывод: Процедура демонстрируется на более сложной системе, подтверждая универсальность метода Тастина.

---

3. Метод нулей и полюсов

Этот раздел показывает альтернативный метод синтеза дискретной ПФ по заданным нулям и полюсам непрерывной системы.

Процесс
1. **Нахождение нулей и полюсов:** Из непрерывной ПФ извлекаются корни числителя (нули) и знаменателя (полюса).
2. **Отображение в Z-область:** Непрерывные полюса `p` отображаются в дискретные `z` по формуле `z = exp(p/fd)`. Это позволяет получить общую форму дискретной ПФ как отношение произведений `(z - zn_i)` и `(z - zd_i)`.
3. **Определение коэффициента k:** Чтобы гарантировать совпадение характеристик на нулевой частоте, коэффициент `k` находится из условия `W_continuous(p=0) =

W_discrete(z=1)` путем решения уравнения.
4. **Построение АЧХ:** Строится график АЧХ для непрерывной и полученной дискретной ПФ для сравнения.

Вывод: Этот метод позволяет напрямую конструировать цифровой фильтр, зная только полюса и нули его аналогового прототипа.

---

4. Пример с нулевым нулём

Рассматривается случай, когда непрерывная ПФ имеет нуль в начале координат (p=0). В этом случае простое совпадение значений на нулевой частоте не работает (обе ПФ дают ноль).

Решение
* **Метод:** Используется совпадение **производных** КЧХ на нулевой частоте.
* **Процесс:** Вычисляются производные dH(f)/df для непрерывной и дискретной ПФ при `f=0`. Коэффициент `k` находится из уравнения, приравнивающего эти производные.

Вывод: Этот метод расширяет возможности настройки коэффициента усиления дискретной модели в особых случаях.

---

5. Общий вид цифрового фильтра второго порядка

Процесс
1. **Формула ПФ:** Задается общая форма ПФ второго порядка с комплексно-сопряженными полюсами.
2. **Получение коэффициентов:** Формула ПФ преобразуется в стандартный вид цифрового фильтра `H(z) = (b0 + b1*z^(-1) + b2*z^(-2)) / (1 + a1*z^(-1) + a2*z^(-2))`.
3. **Извлечение коэффициентов:** С помощью функции `makelist(coeff(...))` из числителя и знаменателя извлекаются коэффициенты `bN` и `aN`.
4. **Построение характеристик:** Для заданных численных значений нулей, полюсов и частоты дискретизации строятся графики АЧХ и КЧХ.

Вывод: Этот блок является шаблоном для генерации коэффициентов настоящего цифрового фильтра (например, IIR-фильтра) по заданным полюсам и нулям.

Заключение

Файл `TransferFuncs1and2ord.mac` является комплексным учебным и исследовательским инструментом для работы с передаточными функциями. Он охватывает ключевые этапы анализа и синтеза цифровых систем: от моделирования аналоговых схем до получения коэффициентов цифровых фильтров и визуализации их характеристик.

Ключевые функции Maxima и их применение

В скрипте активно используются следующие функции Maxima. Ниже приведено более подробное описание их применения:

1. `solve(eq, vars)`
* **Назначение:** Решает систему уравнений `eq` относительно переменных `vars`.
* **Применение в скрипте:** Фундаментальная функция для анализа цепей. Используется для нахождения переменной состояния (например, напряжения `Un`) путем решения системы уравнений, составленных по законам Кирхгофа. Результат — выражение для передаточной функции.

2. `num(expr)` и `denom(expr)`
* **Назначение:** `num` возвращает числитель выражения `expr`, `denom` — знаменатель.
* **Применение в скрипте:** Критически важны для метода нулей и полюсов. После получения передаточной функции `sln`, эти функции позволяют отделить полином числителя (`W1N`) и знаменателя (`W1D`), которые затем используются для нахождения корней.

3. `solve(eq, var)`
* **Назначение:** Находит корни уравнения `eq` относительно переменной `var`.
* **Применение в скрипте:** Применяется к полиному числителя и знаменателя для нахождения **нулей** (`W2N: solve(W1N, p)`) и **полюсов** (`W2D: solve(W1D, p)`) непрерывной передаточной функции. Это основа для последующего синтеза дискретной ПФ.

4. `realpart(z)` и `imagpart(z)`
* **Назначение:** Возвращают действительную и мнимую части комплексного числа `z`.
* **Применение в скрипте:** Используются для разложения комплексной частотной характеристики (КЧХ) на действительную и мнимую части (`V5K: [realpart(V5), imagpart(V5)]`). Эта разбивка необходима для построения графика КЧХ, где по одной оси откладывается реальная часть, а по другой — мнимая.

5. `cabs(z)` и `carg(z)`
* **Назначение:** `cabs` возвращает модуль (абсолютное значение) комплексного числа. `carg` возвращает его аргумент (фазу).
* **Применение в скрипте:** Эти функции напрямую дают АЧХ и ФЧХ системы. `cabs(V5)` вычисляет АЧХ непрерывной системы, а `carg(V5)` — ее ФЧХ. Аналогично для дискретной системы.

6. `demoivre(expr)`
* **Назначение:** Преобразует комплексные экспоненты вида `exp(%i*x)` в тригонометрическую форму `cos(x) + %i*sin(x)` по формуле Эйлера.
* **Применение в скрипте:** Появляется после подстановки `z = exp(%i*2*%pi*f/fd)`. Эта функция упрощает выражение для КЧХ дискретной системы, заменяя комплексную экспоненту на синусы и косинусы, что делает выражение более удобным для анализа и последующего упрощения.

7. `trigsimp(expr)` и `trigreduce(expr)`
* **Назначение:** Эти функции упрощают тригонометрические выражения. `trigsimp` использует основные тождества (например, `sin^2 + cos^2 = 1`), а `trigreduce` преобразует произведения и степени тригонометрических функций в суммы.
* **Применение в скрипте:** Используются на этапе `V6K:trigsimp([realpart(V6),imagpart(V6)])` и далее. После преобразования `demoivre` получаются громоздкие выражения. `trigsimp` и `trigreduce` позволяют привести их к более компактному и читаемому виду перед вычислением модуля и аргумента.

8. `factor(expr)` и `expand(expr)`
* **Назначение:** `factor` раскладывает выражение на множители, `expand` — раскрывает скобки.
* **Применение в скрипте:** `factor` используется повсеместно для упрощения итоговых выражений передаточных функций. `expand`, наоборот, используется в разделе с общим видом фильтра для раскрытия скобок в выражении `W5`, чтобы получить полиномиальную форму, пригодную для извлечения коэффициентов.

9. `makelist(expr, i, i0, i1)`
* **Назначение:** Создает список, вычисляя выражение `expr` для значений индекса `i` от `i0` до `i1`.
* **Применение в скрипте:** Используется для автоматического извлечения коэффициентов цифрового фильтра. Например, `makelist(coeff(W5N,zm,i),i,0,hipow(W5N,zm))` создает список коэффициентов `bN` при степенях `z^(-i)` в числителе передаточной функции.

10. `coeff(poly, var, n)`
* **Назначение:** Возвращает коэффициент при степени `var^n` в полиноме `poly`.
* **Применение в скрипте:** Работает в паре с `makelist`. Используется для извлечения конкретного коэффициента при заданной степени `zm` (которая представляет `z^(-1)`). Это позволяет программно получить все коэффициенты фильтра в виде массива.

11. `hipow(poly, var)`
* **Назначение:** Возвращает наивысшую степень переменной `var` в полиноме `poly`.
* **Применение в скрипте:** Используется в связке с `makelist` для определения верхней границы цикла. `hipow(W5N,zm)` возвращает степень полинома числителя, что позволяет автоматически перебрать все коэффициенты от `z^0` до `z^(-n)`.

Код 4. Markdown: настройки VSCode и автоматизация отладки

```
# Обзор настроек VS Code

Данный документ содержит описание конфигурационных файлов `.vscode`, найденных в проекте, а также анализ их содержимого и внешних вызовов.

## Файлы конфигурации

В директории `.vscode` найдены следующие JSON-файлы:

1. `launch.json`
```

2. `tasks.json`

---

1. launch.json — Конфигурация отладки

Этот файл определяет конфигурации запуска отладчика в VS Code.

Конфигурации:

1.1. RISC-V QEMU Debug
- **Тип:** `cppdbg` (отладка C++)
- **Режим:** Запуск (`launch`)
- **Программа:** `${workspaceFolder}/app.elf`
- **Точка останова при входе:** `true`
- **Рабочая директория:** `${workspaceFolder}`
- **Консоль:** Встроенная (`externalConsole: false`)

Отладчик:
- **Режим:** GDB
- **Путь к отладчику:** `${workspaceFolder}/gdb-wrapper.sh`
- **Адрес сервера:** `localhost:1234`

Команды инициализации:
- Включение красивого вывода: `-enable-pretty-printing`
- Синтаксис ассемблера: Intel

Зависимости:
- **preLaunchTask:** `Start QEMU GDB Server` — запускается перед отладкой
- **postDebugTask:** `Stop QEMU` — выполняется после завершения отладки

1.2. RISC-V QEMU Debug (attach)
- **Тип:** `cppdbg`
- **Режим:** Присоединение к процессу (`attach`)
- **Программа:** `${workspaceFolder}/app.elf`
- **Процесс:** Выбирается через команду `pickProcess`
- Подключается к тому же серверу GDB: `localhost:1234`
- Использует только одну команду инициализации: включение красивого вывода.

Внешние вызовы:
- Запускает скрипт `gdb-wrapper.sh` как обёртку для GDB
- Использует `make debug` (через задачу `Start QEMU GDB Server`) для запуска QEMU

---

2. tasks.json — Определение задач

Файл содержит список задач, которые можно запускать из VS Code.

Список задач:

1. **Build RISC-V Project**
- **Команда:** `make all`
- **Тип:** Сборка (по умолчанию)

2. **Clean Build**
- **Команда:** `make clean`

3. **Start QEMU GDB Server**
- **Команда:** `make debug`
- **Фоновая задача**
- **Мониторинг:** Отслеживает начало и завершение процесса QEMU

4. **Stop QEMU**
- **Команда:** `pkill -f qemu-system-riscv32 || true`
- Принудительно завершает процесс QEMU

5. **Run in QEMU**
- **Команда:** `make run`

Внешние вызовы:
- Запускает `make` с различными целями: `all`, `clean`, `debug`, `run`
- Использует `pkill` для остановки QEMU
- Задача `Start QEMU GDB Server` отслеживает процесс через регулярное выражение и сигналы начала/завершения

Выводы

Проект настроен для разработки под архитектуру RISC-V с использованием QEMU в качестве эмулятора и отладки через GDB.

Основные особенности:
- Используется обёртка `gdb-wrapper.sh` для запуска GDB
- Отладка интегрирована с задачами `make`
- Процесс QEMU запускается и останавливается автоматически
- Поддерживается как запуск, так и подключение к уже работающему процессу

Данные настройки типичны для embedded-разработки на RISC-V с использованием VS Code.

Код 5. Markdown: файлы .xpack_cache.txt - хранит актуальную версию инструментов xPack, gdb-wrapper.sh - скрипт запуска gdb с заданными параметрами, find_xpack.sh — скрипт поиска актуальной версии xPack среди установленных (версии не устанавливаются автоматически, только обновляются из установленных)

 # Подробное описание xPack, gdb-wrapper.sh, find_xpack.sh и .xpack_cache.txt

Данный документ предоставляет детальный разбор скриптов и конфигураций, используемых в проекте для работы с инструментами RISC-V, включая отладчик GDB и эмулятор QEMU.

1. Назначение и структура xPack

xPack — это система управления пакетами, разработанная специально для инструментов разработки (toolchains), таких как компиляторы, отладчики и эмуляторы. Она используется для удобной установки, управления версиями и изоляции этих инструментов.

Структура установки xPack

Инструменты устанавливаются в следующую иерархию директорий:

```
~/.local/xPacks/ # Базовая директория
├── @xpack-dev-tools/ # Пространство имен для официальных xPack-��нструментов
├── riscv-none-elf-gcc/ # Пакет компилятора RISC-V
│   ├── 12.1.0-2.1/ # Старая версия пакета
│   ├── 14.2.0-3.1/ # Средняя версия пакета
│   └── 15.2.0-1.1/ # Самая новая версия пакета
│   └── .content/ # Собственно содержимое пакета
│   ├── bin/ # Исполняемые файлы (gcc, g++, gdb)
│   ├── lib/ # Библиотеки компилятора (libgcc)
│   └── ...
└── qemu-riscv/ # Пакет эмулятора QEMU для RISC-V
└── 7.0.0-1.1/ # Единственная установленная версия QEMU
└── .content/ # Собственно содержимое пакета
└── bin/ # Исполняемые файлы (qemu-system-riscv32)
```

- **`~/.local/xPacks`**: Стандартная директория для пользовательских пакетов xPack.
- **`@xpack-dev-tools`**: Поддиректория, содержащая пакеты от разработчика xPack.

Динамическое обновление версий и семантическое версионирование

В системе xPack используется **двойная система версионирования**, что критически важно для понимания их роли.

Компоненты номера версии (на примере `15.2.0-1.1` и `7.0.0-1.1`)

Номер версии пакета состоит из двух частей, разделенных дефисом:

1. **Основная версия (например, `15.2.0` или `7.0.0`)**:
* Это **версия самого инструмента** (в данном случае, компилятора GCC или эмулятора QEMU).
* `15` и `7` — Мажорная версия. Означает крупный релиз с возможными изменениями, которые могут нарушать обратную совместимость. *Обратите внимание, что это разные

инструменты, и их мажорные версии несопоставимы (GCC 15 vs QEMU 7).
* `2` и `0` — Минорная версия. Обозначает добавление новых функций и улучшений, которые, как правило, являются обратно совместимыми.
* `0` — Патч-версия. Исправления ошибок и незначительные улучшения, полностью обратно совместимые.

2. **Версия xPack-обертки (например, `1.1`)**:
* Это **версия пакета xPack**, который содержит и упаковывает данный инструмент.
* `1` — Минорная версия обертки. Указывает на новые функции или значительные изменения в скриптах установки, структуре пакета или способе настройки.
* `1` — Патч-версия обертки. Исправления ошибок в скриптах установки, обновление зависимостей или исправление багов в скриптах запуска.

Примеры:
* Пакет `riscv-none-elf-gcc/15.2.0-1.1` означает: **Компилятор GCC версии 15.2.0**, и это **первая патч-версия** (1.1) xPack-обертки для этого GCC.
* Пакет `qemu-riscv/7.0.0-1.1` означает: **Эмулятор QEMU версии 7.0.0**, и это **первая патч-версия** (1.1) xPack-обертки для этого QEMU.

Как обновляются версии динамически

Скрипт `find_xpack.sh` обрабатывает разные инструменты **независимо** и основную версию каждого из них **сравнивает только с другими версиями того же самого инструмента**.

1. **Для компилятора GCC**:
* Скрипт сканирует директорию `@xpack-dev-tools/riscv-none-elf-gcc`.
* Он находит все версии: `12.1.0-2.1`, `14.2.0-3.1`, `15.2.0-1.1`.
* Он извлекает **только основные версии инструмента GCC**: `12.1.0`, `14.2.0`, `15.2.0`.
* Сравнивает их по правилам семантического версионирования и выбирает **`15.2.0`** как самую высокую.
* Таким образом, он выбирает полный путь `15.2.0-1.1`.

2. **Для эмулятора QEMU**:
* Скрипт сканирует директорию `@xpack-dev-tools/qemu-riscv`.
* Он находит только одну версию: `7.0.0-1.1`.
* Поскольку других версий нет, он выбирает именно ее.
* Извлечь нечего, так как `7.0.0` и `7.0.0-1.1` — единственная доступная комбинация.

Важно: Скрипт **никогда не сравнивает `15.2.0` (GCC) с `7.0.0` (QEMU)**. Цифры `15` и `7` относятся к разным программным продуктам и не имеют между собой отношения. Каждый инструмент имеет свою собственную линейку версий, и скрипт выбирает последнюю версию для каждого независимо.

Проверка исполняемых файлов

- **GDB** (`riscv-none-elf-gdb`): Это 64-битное ELF-исполняемое приложение для архитектуры x86-64. Оно динамически связано и предназначено для запуска на хост-

системе (Linux x86_64) для отладки целевой RISC-V системы.
- **QEMU** (`qemu-system-riscv32`): Аналогично, это 64-битное ELF-исполняемое приложение для x86-64, которое эмулирует целевую платформу RISC-V (32-битную в данном случае).

2. Файл `.xpack_cache.txt`

Этот файл является **кэшем конфигурации**, сгенерированным скриптом `find_xpack.sh`. Он хранит абсолютные пути к найденным инструментам, что позволяет избежать повторного поиска при каждом запуске.

Содержимое файла

```bash
RISCV_XPACK_XPKG_PATH=/home/timur/.local/xPacks/@xpack-dev-tools/riscv-none-elf-gcc/15.2.0-1.1/.content
RISCV_XPACK_GCC_VERSION=15.2.0
RISCV_XPACK_GDB_PATH=/home/timur/.local/xPacks/@xpack-dev-tools/riscv-none-elf-gcc/15.2.0-1.1/.content/bin/riscv-none-elf-gdb
RISCV_XPACK_QEMU_PATH=/home/timur/.local/xPacks/@xpack-dev-tools/qemu-riscv/7.0.0-1.1/.content/bin/qemu-system-riscv32
RISCV_XPACK_LIBGCC_PATH=/home/timur/.local/xPacks/@xpack-dev-tools/riscv-none-elf-gcc/15.2.0-1.1/.content/lib/gcc/riscv-none-elf/15.2.0/rv32i/ilp32
RISCV_XPACK_LIBM_PATH=/home/timur/.local/xPacks/@xpack-dev-tools/riscv-none-elf-gcc/15.2.0-1.1/.content/riscv-none-elf/lib/rv32i/ilp32
```

Назначение переменных

- **`RISCV_XPACK_XPKG_PATH`**: Корневая директория компилятора. Используется как база для построения других путей.
- **`RISCV_XPACK_GCC_VERSION`**: Версия компилятора GCC (15.2.0), извлеченная из структуры пакета. Обратите внимание, что здесь указана только **основная версия**, без префикса xPack.
- **`RISCV_XPACK_GDB_PATH`**: Прямой путь к исполняемому файлу отладчика GDB.
- **`RISCV_XPACK_QEMU_PATH`**: Прямой путь к исполняемому файлу эмулятора QEMU.
- **`RISCV_XPACK_LIBGCC_PATH`**: Путь к библиотеке `libgcc.a`, которая предоставляет низкоуровневые функции, необходимые для компиляции (например, арифметика с плавающей точкой, если нет FPU).
- **`RISCV_XPACK_LIBM_PATH`**: Путь к библиотеке `libm.a`, которая предоставляет математические функции (sin, cos, sqrt и т.д.).

Проверка библиотек

Обе библиотеки существуют и доступны:
- `libgcc.a`: 1.7 МБ, содержит функции для работы с 32-битной RISC-V (`rv32i`) с ABI `ilp32`.
- `libm.a`: 1.4 МБ, содержит математические функции для той же целевой архитектуры.

3. Скрипт `find_xpack.sh`

Этот скрипт отвечает за **автоматическое обнаружение** и **настройку** инструментов xPack. Он является центральным элементом в процессе конфигурации.

Ключевые функции

1. **Поиск путей (`SEARCH_PATHS`)**: Скрипт ищет пакеты в стандартных директориях:
- `~/.local/xPacks` (пользовательские пакеты)
- `/opt/xPacks` (системные пакеты)
- `/usr/local/xPacks` (альтернативная системная директория)

2. **Поиск последней версии (`find_latest_tool`)**: Перебирает все поддиректории пакета (например, `riscv-none-elf-gcc`) и выбирает версию с наивысшим номером, используя семантическое сравнение версий. **Ключевой момент**: сравнение происходит по основной части версии (до дефиса), игнорируя версию xPack-обертки, и **проводится только в пределах одного типа инструмента**.

3. **Определение версии GCC (`extract_gcc_version`)**: Сначала ищет директорию с версией в `lib/gcc/`, а если не находит, извлекает из имени директории пакета. **Извлекается только основная версия (например, `15.2.0`)**.

4. **Поиск путей к библиотекам (`find_library_path`, `find_libm_path`)**: Ищет библиотеки `libgcc.a` и `libm.a` для целевой архитектуры `rv32i/ilp32`. Использует обнаруженную **основную версию GCC** для поиска в правильной директории.

5. **Формирование вывода (`output`)**: Генерирует набор переменных в формате `KEY=VALUE`, который может быть использован Makefile или другими скриптами.

Режимы работы

- **Режим вывода в stdout**: Если не передан аргумент, скрипт выводит переменные на стандартный поток. Это удобно для отладки.
- **Режим записи в файл**: Если передан один аргумент (например, `.xpack_cache.txt`), скрипт записывает переменные в этот файл, создавая или перезаписывая его.

4. Скрипт `gdb-wrapper.sh`

Этот скрипт является **оберткой для запуска отладчика GDB**. Его основная задача — **обеспечить правильный контекст** для запуска GDB, загрузив переменные из кэша.

Работа скрипта

1. **Загрузка кэша**: Скрипт определяет свой собственный путь (`${0%/*}`) и загружает файл `.xpack_cache.txt` из той же директории.
```bash
CACHE_FILE="${0%/*}/.xpack_cache.txt"
if [ -f "$CACHE_FILE" ]; then
export $(grep RISCV_XPACK_GDB_PATH "$CACHE_FILE" | xargs)
fi
```
Заметьте, что скрипт загружает переменные **только если файл существует**. Для

экономии вычислений он извлекает не весь файл, а только строку, содержащую `RISCV_XPACK_GDB_PATH`, и экспортирует ее.

2. **Проверка переменной**: Скрипт проверяет, была ли успешно загружена переменная `RISCV_XPACK_GDB_PATH`. Если нет, он завершается с ошибкой.
```bash
if [ -z "$RISCV_XPACK_GDB_PATH" ]; then
echo "Error: RISCV_XPACK_GDB_PATH not found in $CACHE_FILE" >&2
exit 1
fi
```

3. **Запуск GDB**: Скрипт использует `exec` для замены своего процесса на процесс GDB, передавая ему все аргументы, полученные извне (например, из VS Code).
```bash
exec "$RISCV_XPACK_GDB_PATH" "$@"
```
Использование `exec` является лучшей практикой, так как оно не создает лишнего процесса.

Заключение

Система настроена следующим образом:

1. **`find_xpack.sh`** один раз запускается для настройки и генерирует файл **`.xpack_cache.txt`** с путями.
2. **`gdb-wrapper.sh`** при каждом запуске отладки загружает этот кэш и запускает отладчик GDB по найденному пути.

Этот подход обеспечивает гибкость (инструменты можно переустановить или обновить в любом месте) и надежность (отладка не зависит от переменных окружения в терминале). Эмулятор QEMU также настроен аналогичным образом, что позволяет полноценно тестировать и отлаживать RISC-V приложения в среде разработки.

Код 6. Markdown полезная утилита: преобразование из png в jpg с итерацией по качеству и масштабу для заданной величины файла

 # Скрипт convert_png_to_jpg.sh: Подробное описание алгоритма

Этот bash-скрипт автоматически конвертирует все файлы `.png` в каталоге (и его подкаталогах) в формат `.jpg`, а также оптимизирует уже существующие файлы `.jpg` и `.jpeg`. Основная цель — привести размер каждого изображения к диапазону **40–60 КБ**, используя математический метод **деления отрезка пополам (Bisection Method)**.

Общая структура работы скрипта

1. **Инициализация**: Установка целевых размеров и создание временной директории.
2. **Конвертация PNG**: Поиск всех `.png` файлов и их конвертация в `.jpg` с оптимизацией размера.
3. **Оптимизация JPG**: Анализ и, при необходимости, перекодирование существующих `.jpg` файлов.
4. **Очистка**: Удаление временных файлов.

Детальный алгоритм оптимизации (Функция `convert_to_size_range`)

Центральной частью скрипта является функция `convert_to_size_range`, которая применяет метод бисекции для поиска оптимального размера изображения.

Шаг 1: Проверка на достаточную малость (без масштабирования)

Скрипт сначала пробует конвертировать изображение без изменения размера (100% масштаб), но с высоким качеством (85%).

```bash
convert "$input_file" -strip -quality 85 "$temp_output"
```

Если получившийся файл уже укладывается в целевой диапазон (≤ 60 КБ), дальнейшая оптимизация не требуется, и этот результат используется.

Шаг 2: Метод деления отрезка пополам (Bisection Method)

Если файл на предыдущем шаге слишком велик, запускается итерационный процесс бисекции.

* **Инициализация границ**:
* `xn` (нижняя граница): 10% — минимальный масштаб, до которого можно уменьшать.
* `xk` (верхняя граница): 100% — текущий масштаб изображения.
* `scale`: Текущее значение масштаба для применения.
* `max_iterations`: Максимальное количество итераций (7), так как метод бисекции очень эффективен (7 итераций дают точность ~1% от исходного диапазона).

* **Итерационный цикл**:
Пока размер файла находится вне диапазона 40–60 КБ и не достигнуто максимальное количество итераций, выполняются следующие шаги:

1. **Вычисление середины интервала**: На каждой итерации скрипт вычисляет новое значение масштаба, равное середине текущего интервала.
```bash
local dx=$(( (xk - xn) / 2 ))
scale=$(( xn + dx ))
```

2. **Применение масштаба**: Изображение перекодируется с новым масштабом (по

умолчанию качество 80%).
```bash
convert "$input_file" -strip -resize "${scale}%" -quality $quality "$temp_output"
```

3. **Проверка результата**: Измеряется размер полученного файла.

4. **Корректировка границ**:
* **Если размер > 60 КБ**: Изображение всё ещё слишком большое. Это означает, что оптимальный масштаб лежит в **нижней половине** текущего интервала. Скрипт сужает интервал, устанавливая верхнюю границу `xk` на текущее значение `scale`.
* **Если размер < 40 КБ**: Изображение слишком маленькое. Оптимальный масштаб находится в **верхней половине** интервала. Скрипт устанавливает нижнюю границу `xn` на текущее значение `scale`.

Этот процесс последовательно уменьшает интервал неопределенности вдвое на каждой итерации, быстро сходясь к оптимальному значению.

Шаг 3: Финальная корректировка качества

После завершения цикла бисекции, если размер изображения всё ещё превышает 60 КБ, скрипт делает последнюю попытку — снижает качество до 50% без дальнейшего изменения масштаба:

```bash
convert "$input_file" -strip -resize "${scale}%" -quality 50 "$temp_output"
```

Это позволяет спасти файлы, которые сложно ужать только за счёт масштабирования.

Обработка файлов

* **Поиск PNG**: Скрипт использует команду `find` для рекурсивного поиска всех файлов с расширением `.png` (без учёта регистра), исключая скрытые директории (`.git`, `.vscode`) и виртуальные окружения (`venv`, `.venv`).
* **Конвертация**: Для каждого найденного `.png` файла вызывается функция `convert_to_size_range`, которая создаёт соответствующий `.jpg` файл в том же к��талоге.
* **Оптимизация JPG**: Аналогичный процесс проходит для всех существующих `.jpg` и `.jpeg` файлов. Если их размер выходит за пределы 40–60 КБ, они перекодируются.

Заключение

Скрипт `convert_png_to_jpg.sh` — это эффективный инструмент для массовой оптимизации изображений. Использование метода бисекции позволяет с минимальным количеством итераций найти баланс между качеством и размером файла, что значительно превосходит простое линейное уменьшение масштаба.

Код 7. Markdown полезная утилита: преобразование из png в png с итерацией по глубине цвета и масштабу для заданной величины файла

 # Алгоритм работы скрипта `convert_png_to_png.sh`

Данный скрипт предназначен для оптимизации изображений в формате PNG с целью приведения их размера к диапазону от 40 до 60 КБ. Оптимизация достигается за счет двух основных методов: ****уменьшения цветовой палитры (квантования)**** и ****масштабирования изображения****. Скрипт использует утилиты `convert` (из ImageMagick) и `xargs` для параллельной обработки файлов.

Общая структура и настройки

1. ****Инициализация переменных:**** Скрипт экспортирует глобальные переменные `MIN_SIZE` (40 КБ) и `MAX_SIZE` (60 КБ) как целевые границы размера файла. Также создается временная директория `/tmp/optimize_png` для хранения промежуточных результатов.

2. ****Функция `get_size_kb`:**** Эта вспомогательная функция принимает путь к файлу и возвращает его размер в килобайтах (округленный вверх). Она использует команду `stat`, а в качестве резервного варианта — `ls -l`, что делает скрипт более универсальным для разных операционных систем.

3. ****Экспорт функций:**** Ключевым моментом является использование `export -f` для функций `get_size_kb`, `convert_to_size_range` и `process_single_file`. Это позволяет функциям быть доступными в дочерних процессах, создаваемых командой `xargs -P 0`, что обеспечивает параллельную обработку.

Алгоритм оптимизации одного изображения

Основная работа выполняется в функции `convert_to_size_range`, которая пошагово обрабатывает один входной файл.

Этап 1: Оптимизация цветовой палитры и постеризация

На этом этапе скрипт пытается достичь целевого размера, не меняя физических размеров изображения.

1. ****Цикл по цветовым шагам:**** Скрипт последовательно пробует уменьшить количество цветов в изображении, используя массив `colors_steps=(256 192 128 96 64 48 32 24 16 8)`. Обработка начинается с 256 цветов и продолжается в сторону меньшего количества.

2. ****Адаптивная постеризация:**** При каждом шаге сокращения палитры устанавливается соответствующий уровень постеризации (`-posterize`), который помогает сгладить визуальные артефакты квантования. Правила подбора уровня:
* 256-128 цветов: постеризация 6 уровней.
* 128-64 цветов: постеризация 4 уровня.
* 64-24 цветов: постеризация 3 уровня.
* Менее 24 цветов: постеризация 2 уровня.

3. **Конвертация и проверка размера:** Для каждой комбинации цветов и постеризации выполняется команда `convert` с флагами:
* `-strip`: Удаление метаданных (EXIF, комментариев), что уменьшает размер файла.
* `-dither None`: Отключение дизеринга для более предсказуемого результата.
* `-posterize N`: Применение постеризации.
* `-colors N`: Сокращение палитры до `N` цветов.
* `-define png:compression-level=9`: Использование максимального уровня сжатия PNG.
4. **Оценка результата:** После каждой конвертации измеряется размер временного файла. Если размер попадает в целевой диапазон (40-60 КБ), оптимизация успешна (`optimized_by_color=true`) и цикл завершается. Если размер становится меньше `MIN_SIZE`, дальнейшее сокращение палитры нецелесообразно, и цикл также прерывается.

Этап 2: Бисекционное масштабирование

Если после этапа 1 размер изображения все еще не в целевом диапазоне, запускается цикл масштабирования.

1. **Инициализация границ:** Задаются начальные границы для бисекции: `xn=10` (10%) и `xk=100` (100%, полный размер). Переменная `scale` инициализируется на 100%.

2. **Итерационный процесс:** Цикл выполняется до тех пор, пока размер не попадет в диапазон, либо пока не будет достигнуто максимальное количество итераций (7) или границы не сойдутся.

3. **Расчет нового масштаба:** На каждой итерации вычисляется середина интервала: `dx = (xk - xn) / 2`, и `scale = xn + dx`. Это и есть принцип бисекции (дихотомии).

4. **Масштабирование и конвертация:** Изображение масштабируется до `scale%` с использованием тех же параметров, что и на этапе 1 (с текущими значениями `target_colors` и `target_posterize`).

5. **Сужение границ:** Измеряется новый размер:
* Если размер **больше** `MAX_SIZE`, значит изображение все еще слишком большое, и верхняя граница (`xk`) сдвигается вниз к `scale`.
* Если размер **меньше** `MIN_SIZE`, значит изображение слишком маленькое, и нижняя граница (`xn`) сдвигается вверх к `scale`.

Такой подход позволяет за небольшое количество итераций эффективно найти оптимальный размер масштабирования.

6. **Финализация:** Как только итерации завершены, финальный оптимизированный файл копируется из временного хранилища в выходной файл (с префиксом `_` в имени) и временный файл удаляется.

Основной процесс обработки (`process_single_file`)

Функция `process_single_file` является точкой входа для каждого файла, переданного `xargs`.

1. **Подготовка имен:** На основе пути входного файла (`$png_file`) строится путь для выходного файла (`$optimized_png_file`), который добавляет префикс `_` к базовому имени (например, `image.png` -> `_image.png`).

2. **Вывод информации:** В поток ошибок (`>&2`) выводится информация о начале обработки исходного файла и его размере.

3. **Вызов `convert_to_size_range`:** Эта функция вызывается для выполнения всей логики оптимизации.

4. **Вывод результата:** После завершения оптимизации выводится сообщение о создании результата и его итоговом размере.

Поиск и параллельный запуск

1. **Фильтрация:** Скрипт определяет переменную `FIND_EXCLUSIONS`, которая исключает из поиска скрытые директории (начинающиеся с `.`), виртуальные окружения Python, а также **любые PNG-файлы, которые уже начинаются с символа `_`**. Это предотвращает повторную обработку уже оптимизированных файлов.

2. **Целевая директория:** `TARGET_DIR` устанавливается либо в аргумент командной строки, либо по умолчанию в директорию, где находится сам скрипт.

3. **Поиск и запуск:** Команда `find` ищет все файлы с расширением `.png` в целевой директории, применяя фильтры исключения. Результат поиска передается в `xargs`, который запускает команду `bash -c 'process_single_file "{}"'` для каждого найденного файла.

4. **Параллелизм:** Ключ `-P 0` в `xargs` означает, что `xargs` будет использовать максимальное количество процессов, равное количеству логических ядер процессора, что значительно ускоряет обработку большого количества файлов.

5. **Завершение:** В конце скрипта удаляется временная директория `/tmp/optimize_png`, и выводится сообщение об успешном завершении.