Листинг 1. Скрипт для генерации синусоидального сигнала в системе Octave

```
#!/usr/bin/octave -qf
pkg load audio # подгружаем библиотеку для работы со звуком
dur = 1.0; # длительность сигнала 1 секунда
fs = 16000; # частота дискретизации 16 кГц=16000 Гц
t = 0 : (1/fs) : dur; # генерируем вектор (матрицу-строку) содержащий
# отсчёты времени от нуля до 1 секунды через интервал дтскретизации
# T=(1/fs)=(1/16000) секунды
t=t'; # трансформируем матрицу-строку в матрицу-столбец, т.к функции
# генерации  звука работают только с матрицей-столбцом
s = sin( 2*pi*500*t ); # вычислям 16000 отсчётов синусоидального
# сигнала с  частотой 500 Гц
plot(t,s); # строим график синусоиды
axis(([0,0.01])'); # на отрезке от 0 до 0.01 секунды
wavwrite(s,fs,"sinus.wav"); # записываем сгенерированыый сигнал в
#wav-файл, который затем можно проиграть в любом аудиопроигрывателе.
sound(s,16000); # отправляем сгенерированый  звук на звуковую карту.
# В динамиках компьютера слышен тон частотой 500 Гц
# и длительностью 1 секунда
pause;
```

Листинг 2. Алгоритм для генерации звуковых сигналов для программ на языке C

```
unsigned signal[length]; // объявляем массив содержащий отсчёты
    сигнала
generate_signal(signal); // заполняем массив отсчётами сигнала,
    которые
                           // генерирует функция generate_signal
init_audio_ouput_device(); //инициализируем звуковую карту
send_signal_to_soundcard(); // отправляем сигнал на звуковую карту
```

Листинг 3. Программа на C для генерации тонального сигнала частотой 1000 Гц.

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <sys/soundcard.h>
#include <math.h>

#define SIZE (48)
#define PI (4*atan(1))

int fd_out;
const int freq = 1000; // частота сигнала, Гц
int sample_rate = 48000; // частота дискретизации 48 кГц
short buf[SIZE]; // буфер для хранения отсчётов сигнала - длина
    буфера - один
период синусоиды (48000 Гц)/(1000 Гц) = 48 отсчётов

static void generate_sinewave (void) // заполняем буфер отсчётами
// синусоидального сигнала
{
```

```
21   int i;
22   for (i=0;i<SIZE;i++) buf[i] = (short
23 int)lround(32767*sin(2*PI*i*freq/sample_rate));
24 }
25
26 void write_sinewave(void) // посылаем сгенерированный звуковой сигнал
27 // на звуковую карту
28 {
29   if (write (fd_out, buf, sizeof (buf)) != sizeof (buf))
30     {
31       perror ("Audio write");
32       exit (-1);
33     }
34 }
35
36
37 static int open_audio_device (char *name, int mode) //
       инициализировать
38 // звуковую карту
39 {
40   int tmp, fd;
41
42   if ((fd = open (name, mode, 0)) == -1)
43     {
44       perror (name);
45       exit (-1);
46     }
47
48
49   tmp = AFMT_S16_NE; // установить разрядность 16 бит
50   if (ioctl (fd, SNDCTL_DSP_SETFMT, &tmp) == -1)
51     {
52       perror ("SNDCTL_DSP_SETFMT");
53       exit (-1);
54     }
55
56   if (tmp != AFMT_S16_NE)
57     {
58       fprintf (stderr,
59          "The device doesn't support the 16 bit sample format.\n");
60       exit (-1);
61     }
62
63   tmp = 1; // установить моно режим
64   if (ioctl (fd, SNDCTL_DSP_CHANNELS, &tmp) == -1)
65     {
66       perror ("SNDCTL_DSP_CHANNELS");
67       exit (-1);
68     }
69
70   if (tmp != 1)
71     {
72       fprintf (stderr, "The device doesn't support mono mode.\n");
```

```
73        exit (-1);
74      }
75
76    // установить частоту дискретизации 48 кГц.
77    if (ioctl (fd, SNDCTL_DSP_SPEED, &sample_rate) == -1)
78      {
79        perror ("SNDCTL_DSP_SPEED");
80        exit (-1);
81      }
82
83    return fd;
84  }
85
86  int main (int argc, char *argv[])
87  {
88
89    char *name_out = "/dev/dsp"; // имя устройства звуковой карты
90
91    if (argc > 1) name_out = argv[1];
92
93    fd_out = open_audio_device (name_out, O_WRONLY);
94
95    generate_sinewave(); // готовим синусоидальный сигнал
96
97    while (1) write_sinewave (); // посылаем в бесконечном цикле период
          синусоиды
98  на звуковую карту
99
100   exit (0);
101 }
```

Листинг 4. Скрипт на GNU/Octave для реализации построения вычисления спектра сигнала.

```
1  #!/usr/bin/octave -qf
2  Ns=1000; # число отсчётов сигнала
3  F1=100; # частота первого тона 100 Гц
4  F2=200; # частота второго тона 200 гц
5  dur=0.5; # длительность сигнала 0.5 сек
6  A0=0.6; # постоянная составляющая 0.6 В
7  A1=0.8; # амплиткда первого тона 0.8 В
8  A2=1.4; # амплитуда второго тона 1.4В
9  An=6; # амплитуда шума 6 В
10 Fs=Ns/dur; # частота дисктеризации. Делим число выборок
11 # на длительность сигнала.
12 t=(0:dur/Ns:dur)'; # создаём вектор из 2000 отсчёов по вермени
13 Signal=A0 + A1*sin(2*pi*F1*t) + A2*sin(2*pi*F2*t); # сигнал
14 # Сигнал состоит из постоянной составляющей и двух тонов амплитудой
15 # A1 и A2
16 S=fft(Signal); # вычисляем ДПФ, то есть спектр
17 F=0:(Fs/Ns):Fs; # частота
18 subplot(2,1,1);
19 plot(t,Signal); # строим график сигнала
20 subplot(2,1,2);
```

```
plot(F,abs(S)); # строим график спектра
pause;
```

Листинг 5. Скрипт на GNU/Octave для реализации построения вычисления спектра сигнала.

```
#!/usr/bin/octave -qf
pkg load signal
Ns=1000; # число отсчётов сигнала
F1=100; # частота первого тона 100 Гц
F2=200; # частота второго тона 200 гц
dur=0.5; # длительность сигнала 0.5 сек
A0=0.6; # постоянная составляющая 0.6 В
A1=0.8; # амплиткда первого тона 0.8 В
A2=1.4; # амплитуда второго тона 1.4В
An=6; # амплитуда шума 6 В
Fs=Ns/dur; # частота дискретизации. Делим число выборок
# на длительность сигнала.
t=(0:dur/Ns:dur)'; # создаём вектор из 2000 отсчёов по вермени
Signal=A0 + A1*sin(2*pi*F1*t) + A2*sin(2*pi*F2*t); # сигнал
# Сишнал состоит из постоянной составляющей и двух тонов амплитудой
# A1 и A2
S=fft(Signal); # вычисляем ДПФ, то есть спектр
S=(2/Ns)*abs(S(1:Ns/2)); # отсекаем половину спектра, которая не
    несёт полезной
информации
S(1)=0.5*S(1); # нормируем постоянную составляющую
F=0:(Fs/Ns):Fs-1; # частота
F=F(1:(length(F)/2)); # отсекаем половину частоты

subplot(2,1,1);
plot(t,Signal); # строим график сигнала
subplot(2,1,2);
plot(F,abs(S)); # строим график спектра
axis([0,300]);

pause;
```

Листинг 6. Код на языке C для реализации цифрового фильтра.

```
float Filter(float xn)
{
nm1=N-1;
yn=0;
for (k=0;k,nm1;++k) {      //смещение данных чтобы
   x[nm1-k]=x[nm1-k-1];    // обсвободить место для новой выборки
   x[0]=xn;
}
for (k=0;k,N;++k) {
   yn=yn+h[k]*x[k];        //данные фильтруются и формируется
}                          //новая выборка
return yn;                 // возвращается выходная выборка фильтра
}
```