Lab 2: Discrete-Time Signals and Systems

DSP Practical
Signal Processing Group
Technische Universität Darmstadt

November 4, 2014

1 Introduction

In digital signal processing, we focus on signals which are discrete in time, as opposed to continuous. This is due to the fact that it is convenient to store and process signals digitally, on a computer. Discrete-time signals can be generated by sampling analog signals, such as audio signals. However, there exist signals which are intrinsically discrete in time, such as the average monthly rainfall. Let us have a look at a simple example: Figure 1 shows a continuous-time output signal $\tilde{x}(t)$ of some system. This can be, for example, the voltage at a certain point in an electrical network. To process the signal, it is sampled with sampling period T at time instances $t = nT$, and measured on the observation interval $0 \leq n \leq N$.

This practical introduces how MATLAB can be used to generate and manipulate discrete-time signals by means of digital filtering. Here we use discrete-time systems which can be described by a linear difference equation. In the laboratory part, you will learn how to generate basic signals such as the unit step or sinusoids. More complex signals such as musical tones are also treated. You will learn how to apply filtering in MATLAB and investigate some properties of linear filters.

2 Discrete-Time Signals

A discrete-time signal $x(n)$ is only defined for certain time-indices $n \in \mathbb{Z}$, also called "samples", and can be understood as a sequence of amplitude

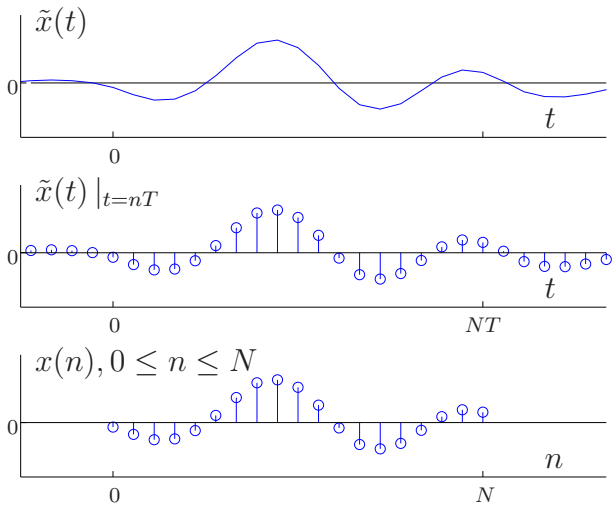

Figure 1: Output signal $\tilde{x}(t)$ of some system, sampled with sampling period T at time instances $t = nT$, and observed on the interval $0 \leq n \leq N$.

values, which can be real or complex. For our simulations, we will consider the amplitude axis as continuous, knowing that the standard double precision format in MATLAB is floating point with 15 decimal places.

2.1 Basic Signals

Some basic signals used in digital signal processing are

- Kronecker delta: $\delta(n) = \begin{cases} 1, & n = 0 \\ 0, & \text{otherwise} \end{cases}$
- Unit step: $u(n) = \begin{cases} 1, & n \geq 0 \\ 0, & \text{otherwise} \end{cases}$
- Exponential sequence: $x(n) = A \cdot e^{\alpha n}$
- Sinusoid: $x(n) = A \cdot \cos(\omega_0 n + \varphi_0)$
- Complex exponential: $x(n) = A \cdot e^{j(\omega_0 n + \varphi_0)}$

Here, A denotes a constant amplitude value, α is a damping constant for an exponential decay, ω_0 is a discrete-time frequency and φ_0 is an initial phase.

Note that the discrete-time frequency ω_0 is normalized and can take values from 0 to π. It is related to a physical frequency f_0 in Hz by $\omega_0 = 2\pi f_0/f_s$, where $f_s = 1/T$ is the sampling frequency.

2.2 Discrete-Time Convolution

Convolution is a fundamental operation between signals. For discrete-time signals it is defined as

$$y(n) = h(n) * x(n) = \sum_{k=-\infty}^{\infty} h(k)x(n-k). \tag{1}$$

Convolution is linear and commutative, so $x(n)$ and $h(n)$ can be interchanged. If the signals are of finite duration with length N_x and N_h, respectively, we obtain an output sequence of length $N_y = N_x + N_h - 1$.

2.3 Signal Generation and Convolution with MATLAB

Using the vector and matrix notation of MATLAB, we can generate basic signals conveniently, without using complicated programming constructs. The following MATLAB commands show how to to generate and plot 31 samples of a sinusoid.

```
>> n=-10:20;
>> x=0.8*sin(2*pi/10*n);
>> stem(n,x)
```

In this example, we use vector `n` to store time-index information and vector `x` to store amplitude information. For example, `x(1)` corresponds to time-index `n(1)=-10`, `x(2)` to time-index `n(2)=-9` and so on. For plotting the signal, we use the `stem` command to emphasize the discrete-time nature. The result is shown in the upper plot in Figure 2.

Another example demonstrates how a unit step and a Kronecker delta can be generated in MATLAB by applying relational operators on the vector of time-indices `n`, for example,

```
>> u=(n>=0);
>> delta=(n==0);
```

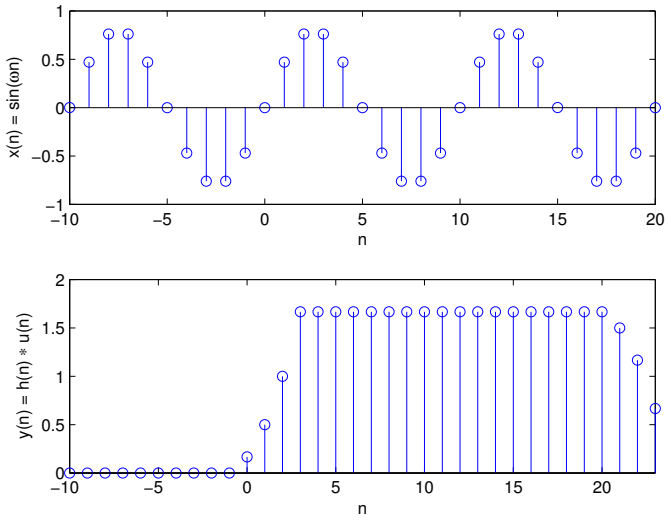

Figure 2: Discrete-time sinusoidal signal.

The convolution operation, as defined in (1), can be carried out in MATLAB by using the function `conv`. For instance, convolving the previously constructed unit step $u(n)$ with some $h(n) \neq 0$ for $n = 0, \ldots, 3$ can be done as follows:

```
>> h=(1:4)/6;
>> n=[n,21:23];
>> y=conv(u,h);
>> stem(n,y)
```

Note that we have appended the vector of time-indices to account for the time-shift. The result is shown in the lower plot in Figure 2.

2.4 Convolution Matrix Notation

Let us have a look at a possible implementation of the convolution operation in terms of matrix notation. We arrange the time-limited sequences $h(n)$ and $x(n)$ in vectors

$$\boldsymbol{h} = \big[h(0), h(1), \ldots, h(N_h - 1)\big]^T$$

and

$$\boldsymbol{x}(n) = \big[x(n), x(n-1), \ldots, x(n - N_x + 1)\big]^T,$$

which we do in MATLAB anyway. The convolution between $x(n)$ and $h(n)$ at time n can now be written as a scalar (inner) product

$$y(n) = \boldsymbol{h}^T \boldsymbol{x}(n) = \mathbf{x}^T(n)\boldsymbol{h}.$$

Also arranging the convolution result $y(n)$ in a vector

$$\boldsymbol{y} = \big[y(0), y(1), \ldots, y(N_x + N_h - 2)\big]^T$$

yields the convenient convolution matrix notation

$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x}.$$

The convolution matrix $\boldsymbol{H}$ is of dimension $(N_x+N_h-1)\times N_x$ and contains all possible time-shifts of $h(n)$. To show the structure of a convolution matrix, we use an example with $N_x = 4$ and $N_h = 3$:

$$\begin{pmatrix} y(0) \\ y(1) \\ y(2) \\ y(3) \\ y(4) \\ y(5) \end{pmatrix} = \begin{pmatrix} h(0) & 0 & 0 & 0 \\ h(1) & h(0) & 0 & 0 \\ h(2) & h(1) & h(0) & 0 \\ 0 & h(2) & h(1) & h(0) \\ 0 & 0 & h(2) & h(1) \\ 0 & 0 & 0 & h(2) \end{pmatrix} \begin{pmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{pmatrix} \tag{2}$$

3 Discrete-Time Systems

A discrete-time system transforms an input signal $x(n)$ into an output signal $y(n)$ using a discrete-time transformation $\mathcal{T}\{\cdot\}$:

$$y(n) = \mathcal{T}\{x(n)\}.$$

Two important system properties are commonly assumed: Linearity and time-invariance. Linearity means that the relation between the input and output of a system satisfies the superposition principle. Time-invariance means that the system does not change with time.

Within the scope of this practical, we will use linear time-invariant systems (LTI) only. As depicted in Figure 3, the input/output relation of LTI systems is completely described by the impulse response, which is defined by the system output to a Kronecker delta $h(n) = \mathcal{T}\{\delta(n)\}$.

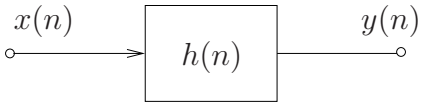

Figure 3: Input/output relation of LTI systems.

3.1 Difference Equation

The input/output relation of an LTI system can also be described mathematically, by a linear constant-coefficient difference equation

$$a_0 y(n) + a_1 y(n-1) + \cdots + a_p y(n-p) = b_0 x(n) + b_1 x(n-1) + \cdots + b_q x(n-q)$$

which can be rearranged as follows:

$$y(n) = \frac{1}{a_0} \left(\sum_{k=0}^{q} b_k x(n-k) - \sum_{k=1}^{p} a_k y(n-k) \right). \tag{3}$$

In the following, we take $a_0 = 1$. Note that besides the impulse response, LTI systems are also completely specified by the filter coefficients b_k for $k = 0, \ldots, q$ and a_k for $k = 1, \ldots, p$. A possible filter implementation in first canonical direct-from is shown in Figure 4 for the case $p = q$.

Note that the filter structure in Figure 4 can be separated into a recursive part, which is represented by the right branch with filter coefficients a_k, and a non-recursive part, which is represented by the left branch with filter coefficients b_k.

3.2 Transfer Function

Taking the z-transform of (3) with $a_0 = 1$, we obtain

$$Y(z) = \sum_{k=0}^{q} b_k X(z) z^{-k} - \sum_{k=1}^{p} a_k Y(z) z^{-k}.$$

Factoring $X(z)$ and $Y(z)$ out and rearranging terms, we come up with the general input/output relation in the z-domain

$$Y(z) = H(z) X(z),$$

where

$$H(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_q z^{-q}}{1 + a_1 z^{-1} + \cdots + a_p z^{-p}} \tag{4}$$

is called the transfer function. By evaluating $H(z)$ on the unit circle, i.e., inserting $z = e^{j\omega}$, we obtain the frequency response of the system $H(e^{j\omega})$.

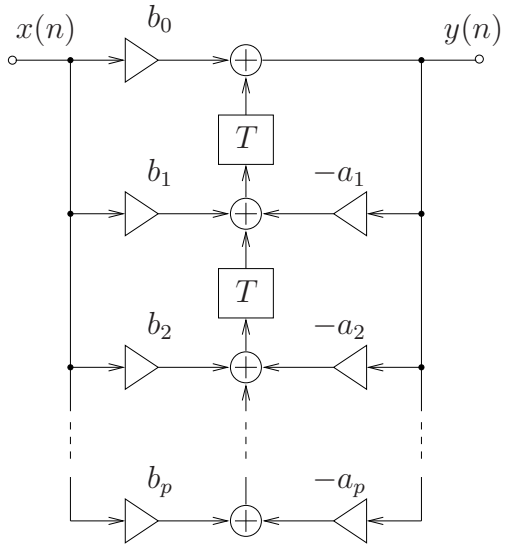

Figure 4: Filter implementation, first canonical direct-form.

3.3 Poles and Zeroes

The pole-zero representation can be derived by finding the roots of the numerator and denominator polynomials of $H(z)$ in (4)

$$H(z) = G\frac{(z-v_1)(z-v_2)\dots(z-v_q)}{(z-w_1)(z-w_2)\dots(z-w_p)},$$

where v_k for $k = 1, \dots, q$ are the zeros of $H(z)$, w_k for $k = 1, \dots, p$ are the poles of $H(z)$, and G is a gain factor.

System analysis can be conducted by checking the pole-zero locations in the z-plane. For example, for a causal system, if all poles lie inside the unit circle, the system is said to be stable. If, in addition, all zeroes lie inside the unit circle, the system is said to have minimum phase, i.e., its inverse system is also stable.

3.4 Filtering with MATLAB

This section shows how MATLAB can be used for filtering and system analysis. As an introductory example, we consider a second-order bandpass filter. A bandpass filter passes frequency components which are in the passband

and attenuates others. The considered transfer function is given by

$$H(z) = \frac{1 - z^{-2}}{1 - 2r\cos(\omega_0)z^{-1} + r^2 z^{-2}} \tag{5}$$

with center frequency ω_0 and bandwidth parameter $r \in (0, 1]$. The 3dB-bandwidth of this filter can be calculated to be $\Delta\omega = 2(1 - r)/\sqrt{r}$. For our example, we consider $\omega_0 = 0.5\pi$ and $r = 0.8$ corresponding to a bandwidth of $\Delta\omega \approx 0.15\pi$. In MATLAB, filter coefficients a_k and b_k, as denoted in (4), are arranged in vectors `b` and `a` , respectively. Using the described values we obtain

```
>> b=[1 0 -1];
>> a=[1 -2*0.8*cos(0.5*pi) 0.8^2]
a =
    1.0000  -0.0000   0.6400
```

The filtering operation can be carried out in MATLAB by using the `filter` function. For example, you can calculate the impulse response of our second-order bandpass by filtering the previously constructed Kronecker delta

```
>> h=filter(b,a,delta);
>> stem(n,h)
```

The result is shown in the upper plot of Figure 5. The frequency response can be calculated in MATLAB using the `freqz` function, and plotted as follows:

```
>> [H,w]=freqz(b,a);
>> plot(w,20*log10(abs(H)))
```

The result is shown in the lower plot of Figure 5, along with the center frequency and the 3dB frequencies.

We can find roots of polynomials by using the function `roots`, and convert back from root to polynomial representation using `poly`. For example, you can calculate the zeros of $H(z)$ by

```
>> roots(b)
ans =
    -1
     1
```

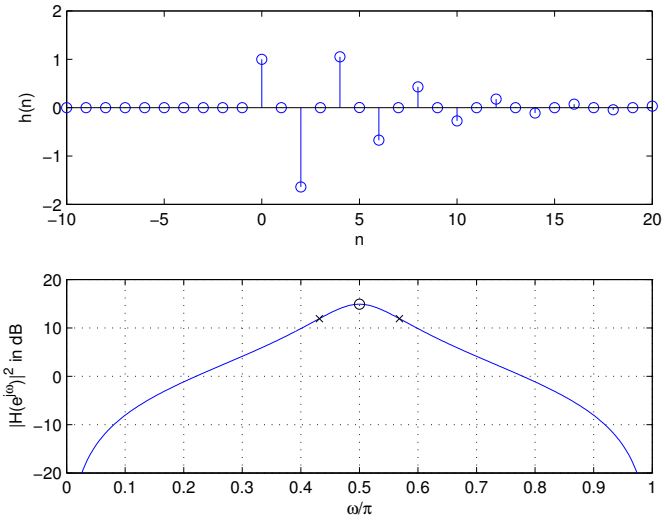

Figure 5: Impulse response and frequency response of the second order bandpass filter from the MATLAB code example.

and the poles of $H(z)$ by typing

```
>> roots(a)
ans =
    0.0000 + 0.8000i
    0.0000 - 0.8000i
```

4 Application Example

Musical signal synthesis tries to imitate a tone played by a real instrument as well as possible. To generate a waveform in the practical, we will use a simple harmonic structure and amplitude modulation for a typical waveform envelope. The considered model for the harmonic structure is

$$x(n) = \sum_{k=1}^{L} p_k \sin\left(\frac{2\pi f_k}{f_s} n\right), \quad n = 0, \ldots, N-1,$$

where f_1 denotes the fundamental frequency with amplitude $p_1 = 1$, and $f_k = kf_1$ for $k = 2, \ldots, L$ are the harmonic frequencies with amplitudes $p_k < p_1$, as shown in Figure 6. The duration of the tone is N/f_s, where N is the signal length.

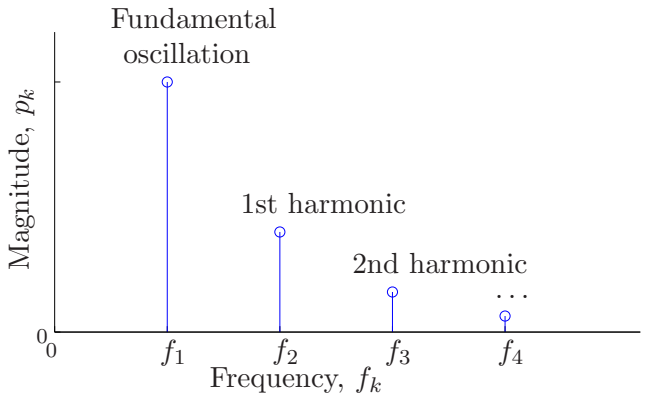

Figure 6: Harmonic structure of a musical tone.

To model the transient characteristics of a tone being played, i.e., onset, gradual decrease, extinguish, etc., an envelope function $A(n)$ is multiplied with $x(n)$. Figure 7 shows a simplified model for the envelope, defined exponentially, in three sections:

$$A(n) = \begin{cases} 1 - e^{-\alpha_1 n}, & 0 \leq n < n_1 \\ d + (1-d)e^{-\alpha_2 (n-n_1)}, & n_1 \leq n < n_2 \\ de^{-\alpha_3 (n-n_2)}, & n_2 \leq n < N \end{cases} \tag{6}$$

where α_1, α_2 and α_3 are parameters for the exponential decay, n_1 and n_2 are timing constants, and d is an amplitude parameter.

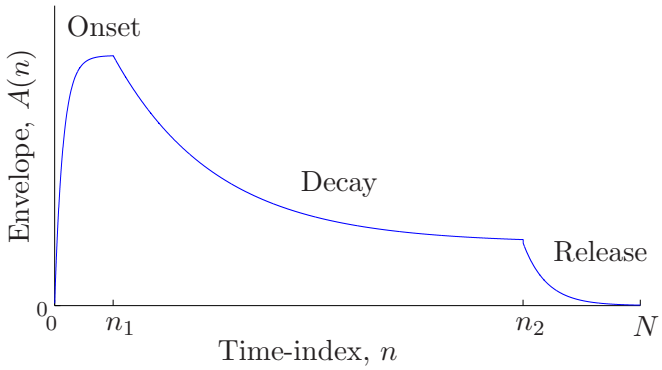

Figure 7: Typical musical tone envelope.

5 Preparation

1. Provide the MATLAB code to generate the following signals for $-10 \leq n \leq 20$:
   - $x_1(n) = \sin(0.12\pi n)$
   - $x_2(n) = u(n) - u(n-6)$
   - $x_3(n) = 0.9^n \cdot u(n)$
   - $x_4(n) = 0.5\delta(n-1) + \delta(n-2) + 0.5\delta(n-3)$
   - $x_5(n) = 0.9^n \cdot \cos(0.2\pi n)$
   - $x_6(n) = \frac{\sin(0.2\pi n)}{0.2\pi n}$
2. Calculate the convolution between $x(n) = \delta(n) + \delta(n-1) + \delta(n-2)$ and $h(n) = \delta(n) - \delta(n-1)$ by hand. Use both the direct definition and the convolution matrix notation.
3. You are given a second-order system with
$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}.$$
   - Under what conditions on a_1 and a_2 is $h(n)$ stable?
   - Write down the corresponding difference equation and calculate the causal impulse response $h(n)$ for $n = 0, 1$ and 2, using the input sequence $x(n) = \delta(n)$. Under what conditions will $h(n)$ be finite?
4. Consider the difference equation
$$y(n) - 0.25y(n-2) = x(n) - 0.25x(n-1).$$
   Write down the transfer function and find its poles and zeroes in the z-plane. Also calculate the first five values of the impulse response.

6 Experiments

6.1 Discrete-Time Signals

1. Use MATLAB to generate the discrete-time signals $x_1(n)$ to $x_6(n)$ from the preparation. Display them using the `subplot` command, and characterize them. Hint: You can use the `sinc` function to generate $x_6(n)$.

2. Write a MATLAB function `convmat.m` which takes two input vectors `x` and `h` and implements the linear convolution using the convolution matrix notation.

3. Generate an averaging filter with impulse response $h(n) = 0.2$ for $0 \leq n \leq 4$ and the following signals:

   - $x_1(n) = 1$ for $0 \leq n \leq 9$
   - $x_2(n) = \sin(\omega_0 n)$ for $0 \leq n \leq 35$ with $\omega_0 = 0.2\pi$ and $\omega_0 = 0.4\pi$

   Use `convmat` to calculate the convolution of $x_1(n)$ and $x_2(n)$ with $h(n)$. Interpret your results. Do you get the same results with `conv`?

6.2 Musical Tone Synthesis

1. Generate 1 second of a sinusoidal tone with frequency $f_1 = 392$ Hz. Use a sampling rate of $f_s = 8192$ Hz. Listen to the tone using `soundsc`.

2. Add 7 harmonics with frequencies $f_k = kf_1$ for $k = 2, \ldots, 8$ to the fundamental oscillation. The harmonics are weighted with magnitudes $p_k = 0.25^{k-1}$. Listen to the new tone; can you notice any difference?

3. Generate an envelope $A(n)$ using the provided function `envelope.m` with default values $n_1 = 240$, $n_2 = 7200$ and $d = 0.25$. Multiply it with the signal from 2 and have a look at your modulated signal, also in the frequency domain by plotting the FFT. Listen to your tone and compare it with those from 1 and 2.

4. Load `pianoG3.mat` to your workspace, the file contains a signal `g` recorded from a real piano at the same f_s. Plot its FFT and listen to it. Discuss how you could improve the 'realness' of your synthesized tone from 3.

6.3 Discrete-Time Systems

We want to 'measure' the magnitude of the frequency response of some system. This is done by exciting the system with test signals and measuring the magnitude relation between output and input. We know from theory that

after some transient effects, the steady state response of an LTI system to $x(n) = \cos(\omega_0 n)$ is

$$y(n) = \left|H(e^{j\omega_0})\right| \cos(\omega_0 n + \angle H(e^{j\omega_0})), \tag{7}$$

where $|H(e^{j\omega_0})|$ is the magnitude, and $\angle H(e^{j\omega_0})$ is the phase of the frequency response, both depending only on ω_0.

Consider the following system transfer function

$$H(z) = \frac{0.16 + 0.48z^{-1} + 0.48z^{-2} + 0.16z^{-3}}{1 + 0.13z^{-1} + 0.52z^{-2} + 0.3z^{-3}}.$$

1. Use MATLAB to calculate the impulse response and frequency response, as described in Section 3.4.

2. Now measure the frequency response magnitude using the described procedure and (7). As test input signal, use $x_k(n) = \cos(\omega_k n)$ for $n = 0, \ldots, 255$ at frequencies $w_k = \pi k/100$ for $0 \leq k \leq 100$. After filtering, exclude transient effects (skip 30 samples) and determine the magnitude of the output signal, which constitutes our measurement of $|H(e^{j\omega_k})|$.

3. Explain why this procedure is reasonable. Compare your measurements with the results from 1.

6.4 Bandpass Filtering

We want to identify bandpass pulses in an unknown, noisy signal by means of second-order bandpass filtering. For this we use the filter transfer function as in (5). A bandpass pulse is a sinusoidal signal which has been multiplied by a pulse-shaped window. It is characterized in the frequency domain by a center frequency ω_0 and a bandwidth $\Delta\omega$, which is inversely proportional to the window size.

1. Load `b3pulses.mat` to your workspace, the file contains a noisy signal vector `x` with three kinds of hidden bandpass pulses: one in the frequency range from 5 to 8 kHz, one between 10.5 and 15.5 kHz and the third in the band 18 to 20 kHz. The sampling frequency is 80 kHz.

2. Now calculate three sets of filter parameters corresponing to the frequency bands specified in 1. Check if `b` and `a` are correct by plotting the frequency response.

3. Now use the three sets of calculated filter coefficients to filter your signal. Determine how many bandpass pulses of each kind are contained in your signal. (If you could not find frequencies ω_0 and parameters r in task 2, use $\omega_0 = \{0.51, 1.02, 1.49\}$ and $r = \{0.89, 0.82, 0.92\}$ to calculate `b` and `a`.)