The Reaktor Core tutorial to accompany "Frequency-scaling of discrete-time LTI system responses"

Vadim Zavalishin*

May 5, 2008

WARNING. This tutorial is not a beginner one. The basic knowledge of DSP theory as well as a good command of Reaktor Core is assumed. Additionally, the reader is supposed to be familiar with the topics covered in "Preserving the LTI system topology in s- to z-plane transforms."

Also, this is a tutorial and not a walkthrough. Be prepared to that a good amount of the material will be available only in the form of hints, with corresponding exercises.

1 Introduction

The article "Frequency-scaling of discrete-time LTI system responses", which this tutorial is supposed to accompany, describes a small set of DSP techniques for the design of discrete-time LTI systems. The purpose of this tutorial is to provide examples of how these techniques can be used within the Reaktor Core environment.

In this tutorial, we are going to restrict ourselves to the topology-preserving method, since the transform method does not require any specific Reaktor Core techniques.

As a basis for our examples we will use the *2 Pole SV* macro from the Reaktor Core standard library. This macro implements a state-variable filter, using two backward difference-transformed integrators. The delayless feedback loop problem is addressed by the introduction of additional z^{-1} delays, resulting in certain artifacts in the filter's response behavior. We will apply the method described in the article to preserve those artifacts.

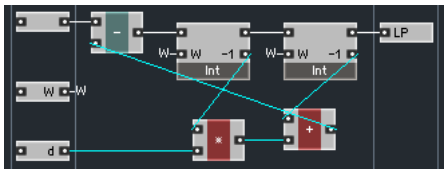

Figure 1: Redesigned *Filter* macro structure.

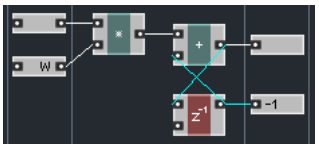

Figure 2: Backward difference-transformed integrator.

In order to be able to keep an overview of the filter structure, whose complexity is going to be increased, we redesign the internal part of the *Filter* macro contained within *2 Pole SV* as shown in Fig. 1. The *BPz* and *HPz* ports were removed, since now we have z^{-1} modules inside the integrators (Fig. 2). Notice, that the integrators must have the *Solid* property disabled! The *BP* and *HP* ports were removed for the sake of reduced complexity.

*Native Instruments GmbH

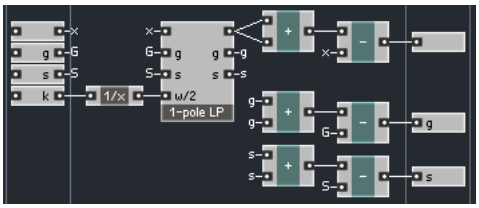

Figure 3: Discrete-time implementation of Eq. 5.

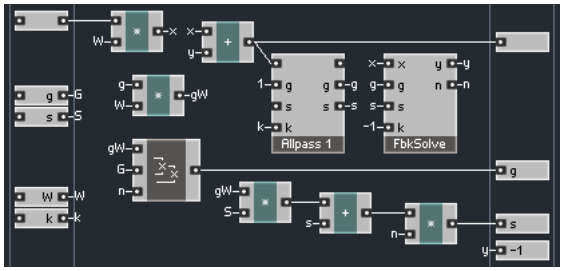

Figure 4: Integrator with the z^{-1} module replaced by an allpass filter. The n output of the *FbkSolve* macro produces the normalization constant $1/(1+gk)$.

2 Changing the delay time of the z^{-1} block

Now we need to build a digital model of Eq. 5[1], which will provide a variable delay time in place of z^{-1} modules. In the Reaktor Core tutorial accompanying "Preserving the LTI system topology in s- to z-plane transforms" we have used a 1-pole lowpass filter to build a ladder filter implementation. Using this 1-pole lowpass filter as a basis, and remembering that for 1-pole filters

$$H_{\mathrm{AP}}(s) = 2H_{\mathrm{LP}}(s) - 1$$

we build the allpass filter in Fig. 3. Apparently, $\omega/2$ must be equal to $1/k$. Remember to turn off the *Solid* property here as well.

Next, we replace the z^{-1} in the integrator by the allpass filter (Fig. 4). Since our integrator now contains a delayless path, we need to change the structure of the *Filter* macro, which therefore becomes pretty similar to the one we used in the tutorial for "Preserving the LTI system topology in s- to z-plane transforms."

Now we are ready to use the method described in the article.

Questions and exercises

1. Explain the relationship of Fig. 1 and Fig. 2 to the original structure of the *2 Pole SV* macro.
2. Modify the structure of the *Filter* macro to contain the Fig. 4 integrators.

3 Changing the sampling rate

Let's imagine we like the artifacts produced by the *2 Pole SV* macro at the 44.1kHz sampling rate, and would like to retain those at other sampling rates.

Therefore, first we modify the *Norm* macro inside of *2 Pole SV* to pretend we are always running at 44.1kHz.

Now, to compute the value of k we need to pick up the frequencies ω_d and ω'_d for Eq. 4. Let's choose the specified filter cutoff value (at the F input) as such frequencies:

$$\omega_d = \frac{2\pi F}{44100\ \mathrm{Hz}} \qquad \omega'_d = \frac{2\pi F}{f_{\mathrm{SR}}}$$

That's basically it, our filter implementation is ready.

Next, we need some test setup to check whether our implementation really works and really makes any difference. Actually, the difference will be lying in the very high frequency area. Therefore, it is suggested to take a sine wave at pitch 127 and put it alternatively through the original *2 Pole SV* filter and through the version we just built. The suggested values of filter parameters are the cutoff pitch 113 (coresponding to 5588Hz) and zero resonance, the reason being that higher pitches will be clipped by the *Clip Max* macro at 44.1kHz. We also use the *Simple Scope* macro to display the output waveform.

Using the specified test setup, we can observe that at 44.1kHz there is no difference between the two filters. At 88.2kHz, the orignal filter's output is getting smaller in the amplitude, the effect becoming

[1]The equation numbers labelled with Eq. refer to the equations in the article.

stronger at higher sampling rates. The output signal of the new filter, on the other hand, is always of approximately the same amplitude.

Questions and exercises

1. Build the filter explained in this section.

4 Changing the cutoff

Now we consider the case of preseving the response shape while changing the filter's cutoff. Let's pretend we like the response shape the filter has at the normalized cutoff value $\omega_d = 0.1$ (corresponding to 700Hz cutoff at 44.1kHz). Therefore we modify the *Norm* module to produce the constant value of 0.1 and compute k from

$$\omega_d = 0.1 \qquad \omega'_d = \frac{2\pi F}{f_{\mathrm{SR}}}$$

We can use basically the same test setup, except that it's suggested to use an even higher frequency sine (10kHz) to make the results better noticeable. Vary the filter cutoff over the entire range to observe the differences in the response.

Questions and exercises

1. Build the filter explained in this section.
2. Implement the 'mixed case', when a certain range of the cutoff (choose one) has to be stretched to the entire possible cutoff range.

5 Homework

Questions and exercises

1. Build an alternative to Fig. 3 allpass filter, by implementing Eq. 3 in the canonical or transposed canonical form. Make sure your implementation has the 'thru' g and s ports. Use this structure to build alternative implementations of the filters in this tutorial. Compare the modulated-case performance of these new filters to their other versions.
2. The z^{-1} modules of the *2 Pole SV* macro perform a dual function. Firstly, they are part of the backward difference-transformed integrators. Secondly, they provide a delay in (initally) delayless feedback loops. Make a version of the *2 Pole SV* where only the second function of the z^{-1} modules is replaced by the allpass filters. Compare this version to the one built in this tutorial. You may also want to analytically compute the transfer functions of both filters and compare their amplitude and phase responses.
3. In case you happen to know a discrete-time structure, to which you think it makes sense to apply the method described in the article, implement a corresponding version of this structure in Reaktor Core.

Acknowledgements

REAKTOR® and *REAKTOR Core Technology*® are registered trademarks of Native Instruments GmbH.