Fractional Delay Farrow Filter

Josef Hoffmann

Contents

1 Fractional Delay Farrow Filter

1.1 Introduction

The Fractional Delay Farrow Filter is a digital filter that delays the discrete-time input signal by a fraction of the sample period. There are many applications where such a delay is necessary. As an example one can consider symbol synchronization in digital receivers, conversion between arbitrary sampling frequencies, echo cancellation, speech coding and speech synthesis, modeling of musical instruments, etc.

Here the design of fractional delay filters (FDF for short) using mathematical interpolation [1] is described. Their implementation using polynomial interpolation allows the calculation of intermediate values at any point between the samples. The polynomial is generally evaluated using the Horner scheme [3], which, due to the recursive calculation rule, requires an effort that only increases linearly with the degree of the polynomial. The appropriate filter are known as Farrow filters [2].

As an introduction, the Lagrangian interpolation is described, which is later used for the development of the Farrow filter.

1.2 Lagrange Interpolation

Lagrange interpolation is widely used in mathematics and technology [1], [2]. It can also be implemented for the interpolation of time-discrete signals via FIR

filters. Because this method is also used for fractional delay filters, the basics of Lagrange interpolation are presented.

Lagrange's theorem states that N distinct real or complex points $x_1.x_2, \ldots, x_N$ can be connected to N real or complex values $y_1, y_2, y_3, \ldots, y_N$ via a polynomial $p_{N-1}(x)$ such that:

$$p_{N-1}(x) = a_1 + a_2\ x + a_3\ x^2 + a_4\ x^3 + \cdots + a_N\ x^{N-1} \tag{1}$$

The coefficients of the polynomial $p_{N-1}(x)$ of degree $N-1$ can be calculated using the system of equations

$$a_1 + a_2\ x_i + a_3\ x_i^2 + a_4\ x_i^3 + \cdots + a_N\ x_i^{N-1} \quad \text{with} \quad i = 1, 2, 3, \ldots, N \tag{2}$$

Another form of the polynomial is better suited for evaluation. With the Lagrangian polynomials

$$l_k(x) = \frac{(x - x_1)(x - x_2)\ldots(x - x_{k-1})(x - x_{k+1})\ldots(x - x_N)}{(x_k - x_1)(x_k - x_2)\ldots(x_k - x_{k-1})(x_k - x_{k+1})\ldots(x_k - x_N)} \tag{3}$$

where $k = 1, 2, 3, \ldots, N$ one obtains the form for the interpolation polynomial $p_{N-1}(x)$:

$$p_{N-1}(x) = y = \sum_{k=1}^{N} y_k\ l_k(x) \tag{4}$$

It should be noted that the term $(x - x_k)$ is missing in the numerator and the term $(x_k - x_k)$ is missing in the denominator. The functions $l_k(x), k = 1, 2, 3 \ldots, N$ therefore have the property:

$$l_k(x_j) = \delta_{k,j} = \begin{cases} 0 & \text{when} \quad k \neq j \\ 1 & \text{when} \quad k = j \end{cases} \tag{5}$$

This means that the interpolating polynomial $p_{N-1}(x)$ takes the value y_i for $x = x_i$ and thus runs exactly through the base points x_i, y_i.

The formulas for the functions $l_k(x)$ from the mathematical literature have been adapted with regard to the implementation in MATLAB, so that the variables start with indices 1 instead of 0.

In the script `lagrange_test.m`, which is designed as a function to allow to connect at end the routine `lagrange_new.m`, in which the functions $l_k(x), k = 1, 2, \ldots, N$ and the interpolated values y for given x values are calculated, is listed here:

```
function lagrange_test
% Function to test the routine lagrange_new

xk = [0,3,5,7.5,10];    % Support points to be interpolated
yk = [1,5,3.2,-2.5,2];  % The values at the points xk
x = 0:0.2:10;           % The interpolation points
```

```
% ------ Interpolation with lagrange_new routine
[y,L] = lagrange_new(xk, yk, x);

figure(1);    clf;
plot(x,y,'k-','LineWidth',1);      hold on;
stem(xk,yk,'k-','LineWidth',1);   grid on;
title('Points to be interpolated and the interpolated function');
xlabel('x');     ylabel('y');

figure(2);    clf;
plot(x,L,'k-','LineWidth',1);      grid on;
title(' The functions l_k(x)');    hold on;

La = axis;
for n = 1:length(xk)
   plot([xk(n), xk(n)], [La(3),La(4)],'k--');
end
plot([La(1), La(2)], [1,1],'k--');

%################
function [y,L] = lagrange_new(xk,yk,x)
% From MATLAB Central by Qazi Ejaz
% https://de.mathworks.com/matlabcentral/fileexchange/
%                 45855-lagrange-interpolation?s_tid=srchtitle
nk = length(xk);
nx = length(x);
L = ones(nx,nk);
for j = 1:nk
    for i = 1:nk
        if (i ~= j)
           L(:,j) = L(:,j).*(x' - xk(i))/(xk(j)-xk(i));
        end
    end
end
y = yk*L';
```

The routine `lagrange_new.m` is an adapted version based on the url specified in program, by author Qazi Ejaz. As input variables the base points are specified with the vectors xk and yk and the vector x is also specified for the values to be interpolated.

As you can see in the test program `lagrange_test.m` five support points `xk = [0,3,5,7.5,10];` with `yk = [1,5,3.2,-2.5,2];` are chosen to be interpolated for the 51 values `x = 0:0.2:10;`.

In the Matrix `L` the functions $l_k(x)$ are calculated as columns. A good exercise is to understand the routine `lagrange_new`, in which the matrix `L` and the interpolated values `y` according to Eq. 3 and Eq. 4 are calculated. In Fig. 1, the support points x_k, y_k are shown together with the interpolated function $y(x)$. Fig. 2 shows the functions $l_k(x), k = 1, 2, \ldots, 5$. These functions have the

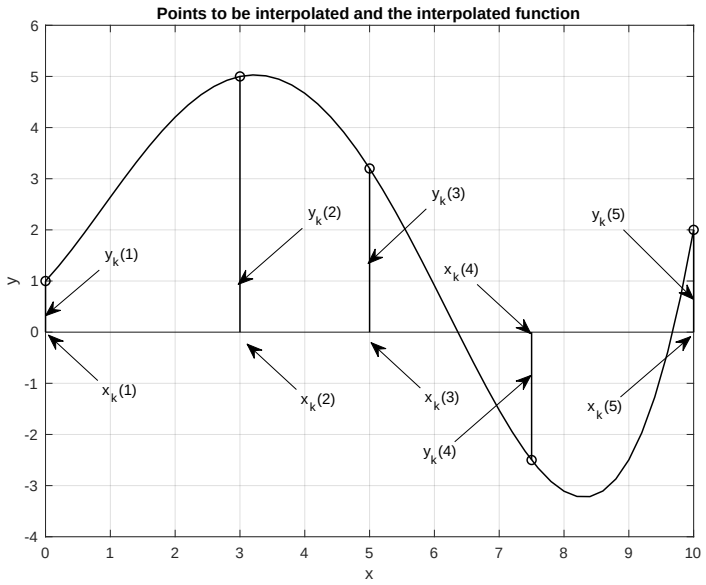

*Fig. 1: Points to be interpolated and the interpolated function (*lagrange_test.m)

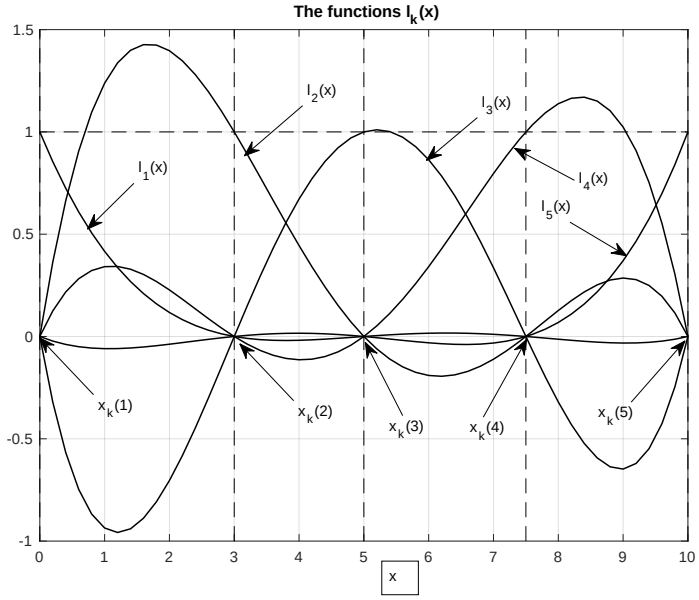

Fig. 2: The functions $l_k(x), k = 1, 2, 3, 4, 5$ *(*lagrange_test.m)

value one for their own vertices and zero for all other vertices. As an example, the function $l_2(x)$ is equal to one for $x_k(2) = 3$ and for all other support points $x_k(1) = 0, x_k(3) = 5, x_k(4) = 7.5, x_k(5) = 10$ this function is equal to zero. This

form ensures that the interpolated function $y(x)$ passes through all the support points.

In the next section, the Lagrange interpolation for the development of a FIR filter as a fractional delay filter is described. The efficient implementation is based on an arrangement known as the Farrow-Filter [3].

1.3 Fractional Delay Filter with Lagrange Interpolation

The fractional delay filter implemented using polynomial interpolation allows intermediate values to be calculated at any point between the samples. The choice of the degree for the interpolation polynomial is fundamentally free, although in practice mostly only third degree polynomials based on four support points are used. Only this cubic interpolation, which is also supported in MATLAB [4], is described further.

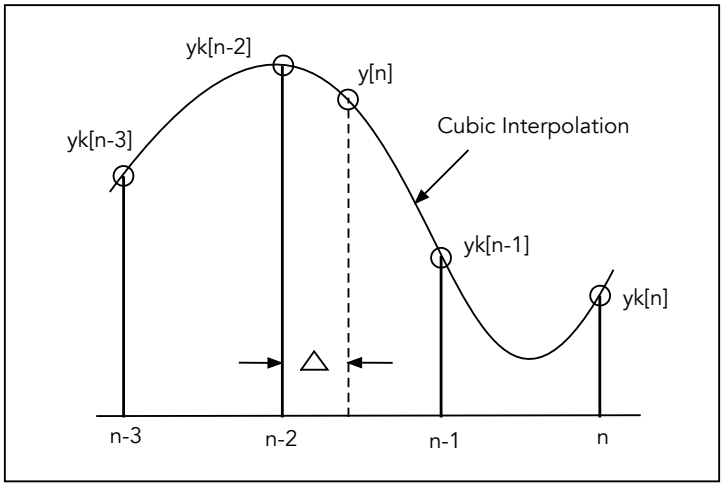

Fig. 3: The Cubic-Interpolation for the Moment $n-2+\Delta$

Fig. 3 shows the cubic interpolation for $y[n]$ at the point $n-2+\Delta$. The time instants are specified relative to the sampling period. To determine the four coefficients of the third-degree polynomial, four samples are required, which are arranged two to the left and two to the right of the value to be interpolated.

The time of the interpolated value $y[n]$ is $n-2+\Delta$ and the current time is n, so for notational convenience the time at which the calculation takes place for y is given and not the location of the interpolated value.

If Lagrangian interpolation is used, one can use the interpolation polynomial according to Eq. 3 and Eq. 4. Instead of the variables x_k, the sampling times $n, n-1, n-2, n-3$ and $n-2+\Delta$ should now be used.

The searched interpolated value $y[n]$ at time $N-2+\Delta$ is then given by:

$$y[n] = \sum_{k=n-3}^{n} y_k[k] l_k[n-2+\Delta] = \sum_{k=0}^{3} y_k[n-k] h[k] \tag{6}$$

This equation actually represents a convolution of the signal values $y_k[n]$ with the impulse response of an FIR filter with the coefficients:

$$h[k] = l_k[n-2+\Delta] \quad \text{for} \quad k = 0, 1, 2.3 \tag{7}$$

For the convolution, the coefficients in $h[k]$ are taken in reverse order, because the coefficients are rotated during the convolution and the equation shown above is thus correctly applied.

The coefficients $l_k[n-2+\Delta], k = n-3, n-2, n-1, n$ can be found as functions of Δ for the cubic interpolation. For $k = n-3$ we get:

$$\begin{aligned} & l_{[n-3]}[n-2+\Delta] = \\ & \frac{((n-2+\Delta)-(n-2))((n-2+\Delta)-(n-1))((n-2+\Delta)-n)}{((n-3)-(n-2))((n-3)-(n-1))((n-3)-n)} \\ & = \frac{\Delta(\Delta-1)(\Delta-2)}{(-1)(-2)(-3)} = \frac{\Delta^3 - 3\Delta^2 + 2\Delta + 0}{-6} \\ & = (-0,1667)\Delta^3 + 0,5\Delta^2 - 0,3333\Delta + 0 \end{aligned} \tag{8}$$

Similarly, for $k = n-2$:

$$\begin{aligned} & l_{[n-2]}[n-2+\Delta] = \\ & \frac{((n-2+\Delta)-(n-3))((n-2+\Delta)-(n-1))((n-2+\Delta)-n)}{((n-2)-(n-3))((n-2)-(n-1))((n-2)-n)} \\ & = \frac{\Delta(\Delta-1)(\Delta-2)}{(1)(-1)(-2)} = \frac{\Delta^3 - 2\Delta^2 - \Delta + 2}{2} \\ & = (0,5)\Delta^3 + (-1)\Delta^2 - 0,5\Delta + 1 \end{aligned} \tag{9}$$

For the remaining two coefficients only the final results are shown:

$$\begin{aligned} & l_{[n-1]}[n-2+\Delta] = (-0,5)\Delta^3 + 0,5\Delta^2 + \Delta + 0 \\ & l_{[n]}[n-2+\Delta] = 0,1667\Delta^3 + 0\Delta^2 + (-0,1667)\Delta + 0 \end{aligned} \tag{10}$$

The coefficients of the third-degree polynomials, summarized in a matrix C are:

```
-0.1667    0.5000   -0.3333         0
 0.5000   -1.0000   -0.5000    1.0000
-0.5000    0.5000    1.0000         0
 0.1667    0        -1.1667         0
```

The values $l_k[n-2+\Delta]$ for $k = n-3, n-2, n-1, n$ summarized in column vector l are determined from the following multiplication:

$$\mathrm{l} = \mathrm{C}\Delta \quad \text{with} \quad \Delta = [\Delta^3, \Delta^2, \Delta, 1]' \tag{11}$$

The running interpolated value $y[n]$ is then calculated by:

$$y[n] = [y_k[n-3, y_k[n-2, y_k[n-1, y_k[n]1 = \mathrm{y_k}1 \tag{12}$$

Written in a matrix form, the current interpolated value $y[n]$ is given by:

$$y[n] = [-\mathrm{y_k}-] \begin{bmatrix} | & | & | & | \\ \mathrm{C}_1 & \mathrm{C}_2 & \mathrm{C}_3 & \mathrm{C}_4 \\ | & | & | & | \end{bmatrix} \begin{bmatrix} \Delta^3 \\ \Delta^2 \\ \Delta \\ 1 \end{bmatrix} \tag{13}$$

Here $\mathrm{C}_1, \mathrm{C}_2, \mathrm{C}_3, \mathrm{C}_4$ are the columns of the matrix C.

If one uses to evaluate the polynomials $l_k[n-2+\Delta]$ with the Horner scheme instead of the direct implementation of Eq. 11, one obtains the filter structure shown in Fig. 4, which is named after C.W.Farrow [3]. The Delta parameter

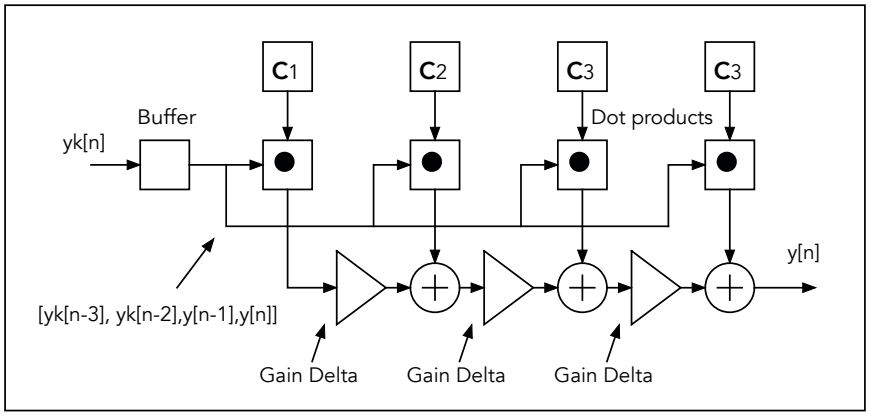

Fig. 4: Structure of cubic interpolation with Farrow filter

Δ appears here as a gain in three blocks and can easily be implemented with product blocks, as will be shown later in the Simulink model. In such a form, a new delta value can then be inserted at each step.

The buffer block updates the four values at each step by bringing in the current one and removing the last one. This buffer together with the dot products forms four FIR filters whose coefficients (or impulse responses) are equal to the columns of the matrix C, however, the order of the elements of these vectors must be reversed. The new structure is shown in Fig. 5.

1.4 Simulation of the Fractional Delay Filter with Lagrange Interpolation

Initially, the Fractional Delay Filter with Lagrange interpolation based on Eq. 6 and Eq. 13 is simulated. The Simulink model for this simulation is shown in Fig. 6.

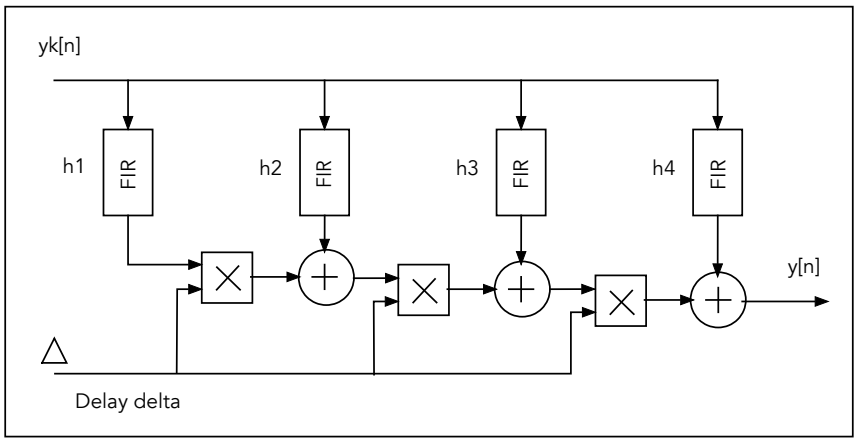

Fig. 5: The new structure of cubic interpolation with Farrow filter

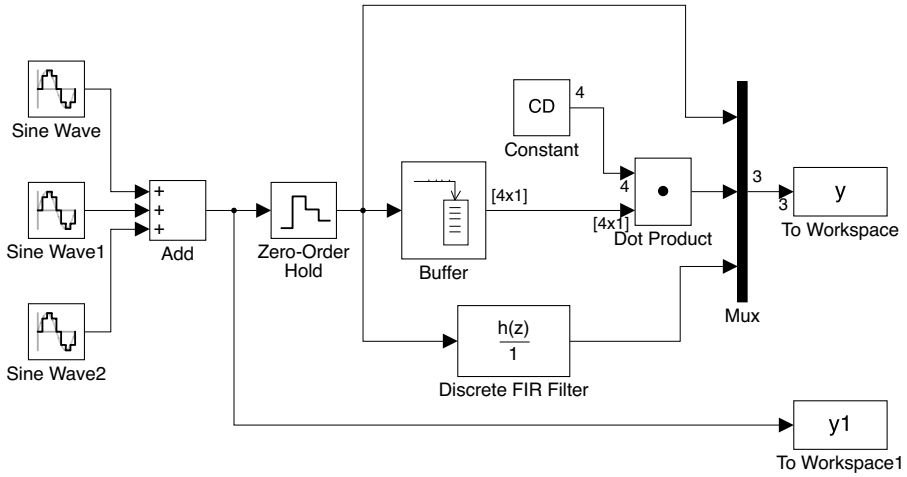

*Fig. 6: Simulink model for investigating a Lagrangian interpolation (*Farrow1.slx, Farrow_1.m)

The sum of three sinusoidal signals as time-continuous signals is used as input signal. These are sampled with a sampling frequency of 100000 Hz and for subsequent sampling at 1000 Hz they can be viewed as time-continuous signals. The sum of the input signals is sampled with the mentioned sampling frequency im `Zero-Order Hold` block.The frequencies of the input signals must be less than 500 Hz in order to fulfill the sampling theorem. Here the frequencies of 10 Hz, 50 Hz and 100 Hz were chosen.

The vectors $[y[n-3], y[n-2], y[n-1], y[n]]$ are formed with the `Buffer` block. The parameter `Output buffer size` is four and `Buffer overlap` is equal to

three. As a result, the buffer is updated with each step.

In the block `Dot Product` the scalar product according to Eq. 13 is realized. The constant from block `Constant` contains for a given value Delta Δ the vector $l = C\Delta$ according to Eq. 13. At the output of `Dot Product` you get the interpolated value at $y[n]$ calculated at step n (see Fig. 3). But it corresponds to the place $n - 2 + \Delta$. This must be taken into account when displaying the interpolated value.

The simulation is initialized and called from the script `Farrow_1.m`. It starts with choosing the sampling frequency $f_s = 1000$ Hz and sampling period. Then the time for the quasi-continuous input signals t is determined. The three input signals are also defined.

```
% Script Farrow_1.m, in which a fractional delay filter
% is examined. Works with model Farrow1.slx
clear;
% ------ Initializations
fs = 1000;     Ts = 1/fs;  % Sample frequency; Sample Period
Tf = 1;                    % Simulation duration
t = 0:Ts/100:Tf;           % Time for the time-continuous signals

fsig1 = 10;    ampl1 = 1;        % The three input signals
sig1 = ampl1*cos(2*pi*fsig1*t);
fsig2 = 50;    ampl2 = 1;
sig1 = ampl2*cos(2*pi*fsig2*t);
fsig3 = 100;    ampl3 = 1;
sig1 = ampl3*cos(2*pi*fsig3*t);

% ------ Matrix C of coefficients
Delta = 0.25;                    % The delay parameter
C = [-0.1667, 0.5, -0.3333, 0;
    0.5,-1,-0.5,1;
    -0.5, 0.5, 1,0;
    0.1667, 0, -0.1667, 0];

Delta_v = [Delta^3, Delta^2, Delta, 1]';
CD = C*Delta_v;

h = flipud(CD)'; % Impulse response of the FIR interpolation filter
```

You can now choose the delay Δ with the variable `Delta`. The matrix C is initialized and the product $C\Delta$ is stored in the vector `CD`.

The impulse response of the FIR interpolation filter is obtained by reversing the order in the vector `CD` and transposing it by `h = flipud(CD)'` (according to Eq. 7). The simulation is then called up with `sim` and the results are displayed.

```
% ------ Call up the simulation
sim('Farrow1',[0,Tf]);
t = y1.Time;
yc = y1.Data;
yinp = y.Data(:,1);      % Input signal
```

```
ydel = y.Data(:,2);       % Interpolated signal
ydel1 = y.Data(:,3);      % With FIR-Filter interpolated signal
```

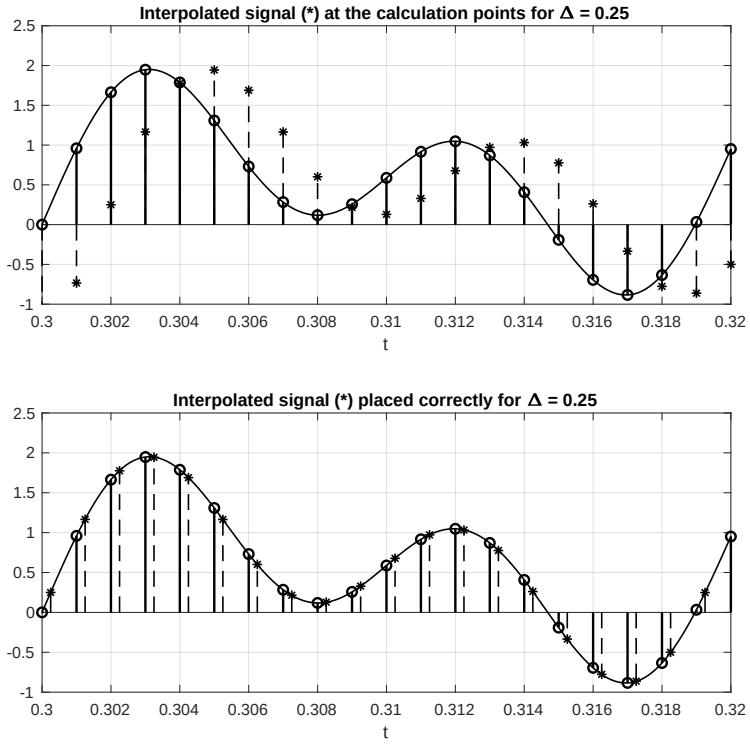

*Fig. 7: Interpolated signal (*) at the calculation points and correctly placed (*Farrow1.slx, Farrow_1.m)

In Fig. 7 above shows, the time-continuous and the sampled values, together with the interpolated values (which are marked with *) placed at locations where they were calculated. Below are the same signals shown with the interpolated values placed in their correct locations. To place the interpolated values correctly, the following command is used:

```
stem(((0:n-1)-2+Delta)*Ts, ydel,'k--','LineWidth',1,'Marker','*');
```

The time axis is shifted with `(-2+Delta)*Ts` which corresponds to the shift $n - 2 + \Delta$ from Fig. 3.

The same interpolation is implemented in the model from 6 using the FIR filter from the block `Discrete FIR Filter`. The impulse response of this filter `h` has only four values or four coefficients. As expected, you get the same results as in the representations from Fig. 7. With the zoom function of the representations one can observe the errors of the interpolation in some places.

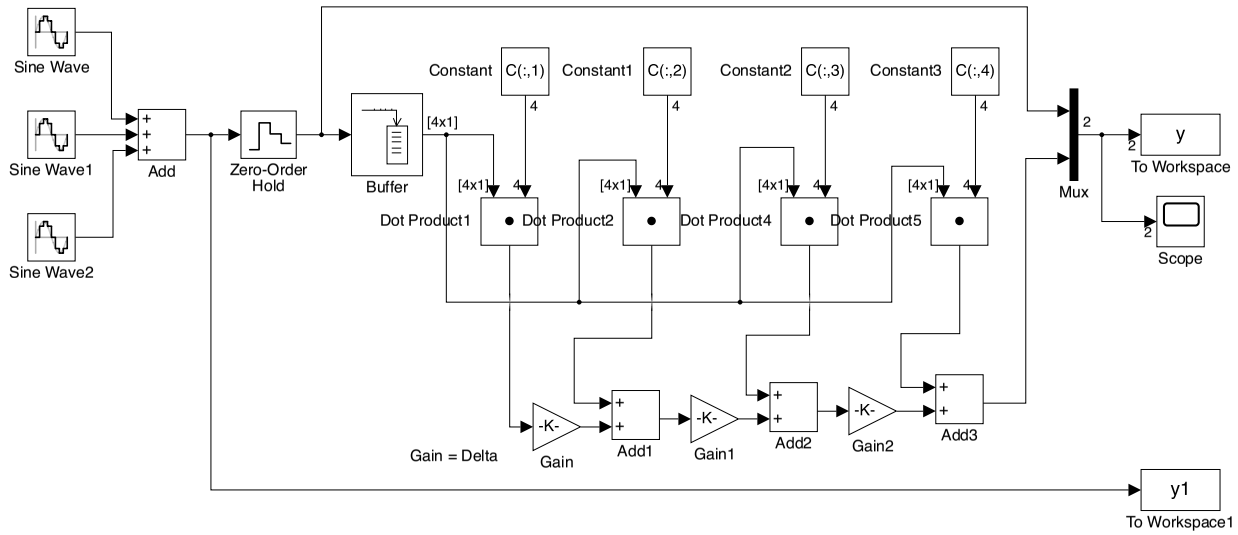

*Fig. 8: Simulink model for investigating a Lagrangian interpolation with Farrow-Filter (*Farrow2.slx, Farrow_2.m)

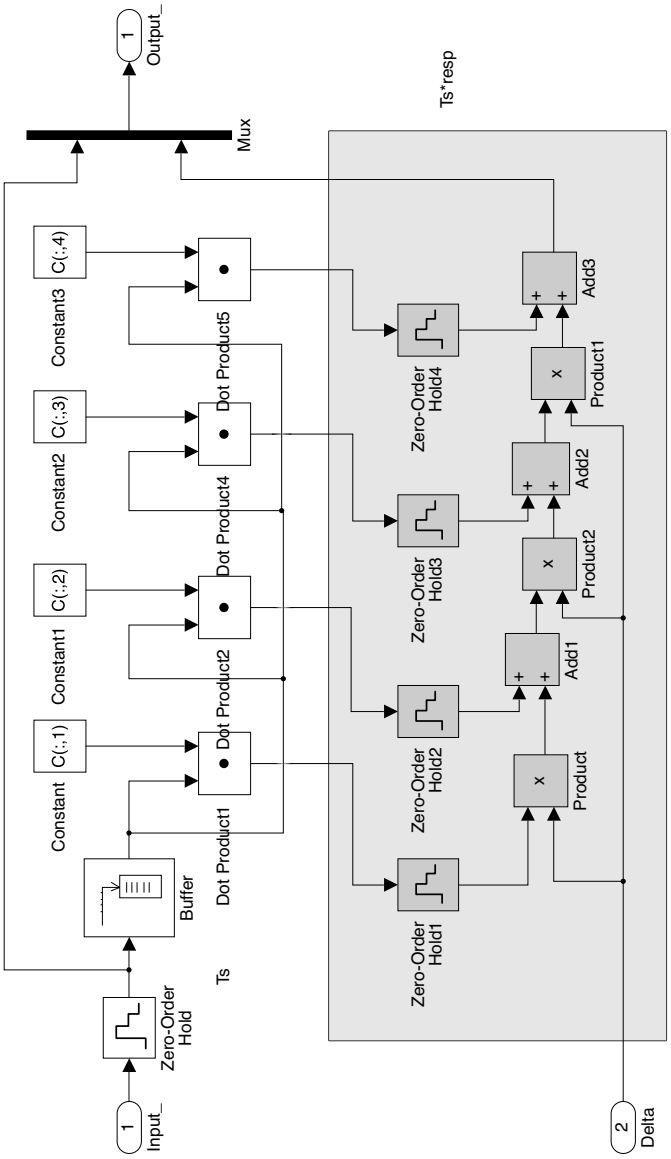

Fig. 9: Simulink model for the Farrow-Filter with changeable Δ (Farrow21.slx, Farrow_2.m)

1.5 Simulation of the Sample Rate Convertion with Farrow-Filter

The structure of the Farrow filter according to Fig. 4 is simulated with the Simulink model `Farow2.slx` from fig. 8. The model is initialized and called from the script `Farow_2.m`. The script is the same as the previous script `Farrow_1.m` and is no longer commented on here. The simulation call is only changed and refers to the new model. The results are the same as the previous ones from Fig. 7.

The model maps the structure from fig. 4 with Simulink blocks. As in the previous simulation, for the `Buffer` block, the parameter `Output buffer size` is four and `Buffer overlap` is equal to three. As a result, the buffer is updated with each step.

The `Gain, Gain1, Gain2` blocks are all parameterized with the delay Δ (Delta). The blocks `Constant, Constant1, ...` are initialized with the columns of the matrix C. Scalar products are realized with the blocks `Dot Product1, Dot Product2,....`

For the application as a Sample Rate Converter (SRC), the delays Δ must be changed with the sample period of the output signal `Ts*resp`. To make this possible, the model is changed as shown in Fig. 9. Here `Ts` is the sampling period of the input signal and `resp` is the SRC factor defined as the ratio `resp = Tout/Tin`, where `Tout` is the sampling period of the output signal and `Tin` is the sampling period of the input signal.

The blackened blocks in Fig. 9 work with the sampling period `Ts*resp` and the remaining blocks work with the sampling period `Ts`.

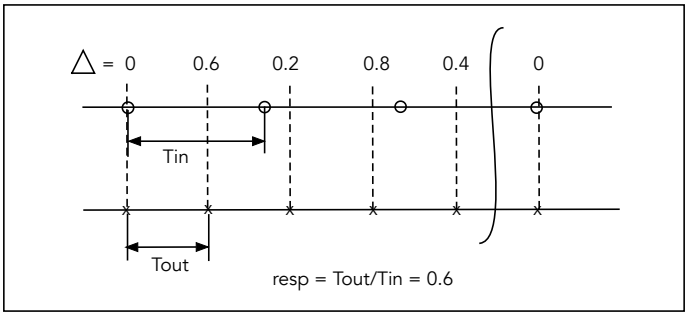

Fig. 10: Example of an Sample Rate Conversion with factor 0.6

For a better understanding fig. 10 shows the SRC for a factor `resp=0.6`. The sampling periods of the input signal, marked with o, are shown above and the sampling periods of the interpolated output signal, marked with x, are shown below.

The resulting necessary delays Δ are given above, assuming that the sampling period as relative value equals one `Tin = 1`. After five sampling periods of the output signal, the sequence of necessary delays Δ is repeated periodically.

Periodic sequences for the delays Δ are not obtained for all SRC factors, as will be shown later.

For this example with `resp = 0.6` you get the shown Δ values of a period by modulo one of the product $0.6 * (0:4)$:

$$\Delta(n) = mod(0.6 * n, 1), \quad n = 0, 1, 2, \ldots 4 \tag{14}$$

With the following MATLAB command you get the necessary delays over more sample periods n of the output signal:

```
Delta = mod(0.6*(0:20),1),
Delta =
  0  0.6000  0.2000  0.8000  0.4000  0  0.6000  0.2000  0.8000  0.4000
  0  0.6000  0.2000  0.8000  0.4000  0  0.6000  0.2000  0.8000  0.4000
```

For an SRC factor equal to 44.1/48, which is used to convert the CD's 44.1 kHz sampling frequency to the studio 48 kHz sampling frequency, there is no periodicity and the delays Δ must be determined for the entire length of the signal. For example, for `n = 0:20` the following values Δ without periodicity are obtained for this SRC factor:

```
Delta = mod(44.1/48*(0:20),1),
Delta =
  0  0.9188  0.8375  0.7563  0.6750  0.5938  0.5125  0.4313  0.3500
  0.2688  0.1875  0.1063  0.0250  0.9438  0.8625  0.7813  0.7000  0.6188
  0.5375  0.4563  0.3750  ...
```

The SRC with Farrow filter is simulated with the model `resample_sim1.slx`, which is initialized and called with the script `resample_sim_1.m`. The model is in fig. 11 shown. The model is initialized with the following instructions in the script:

```
% Script resample_sim_1.m, in which the SRC with a
% fractional delay Farrow filter is examined.
% Works with model resample_sim1.slx
clear;
% ------ Initializations
fs = 1000;     Ts = 1/fs;  % Sample frequency; Sample Period
Tf = 10;                   % Simulation duration
t = 0:Ts/100:Tf;           % Time for the time-continuous signals

fsig1 = 10;    ampl1 = 1;        % The three input signals
sig1 = ampl1*cos(2*pi*fsig1*t);
fsig2 = 50;    ampl2 = 1;
sig1 = ampl2*cos(2*pi*fsig2*t);
fsig3 = 100;    ampl3 = 1;
sig1 = ampl3*cos(2*pi*fsig3*t);

% ------ Resampling factor
resp = 0.6;                % resp = Tout/Tin (Resampling factor)
%resp = 5/3;                % resp = Tout/Tin (Resampling factor)
```

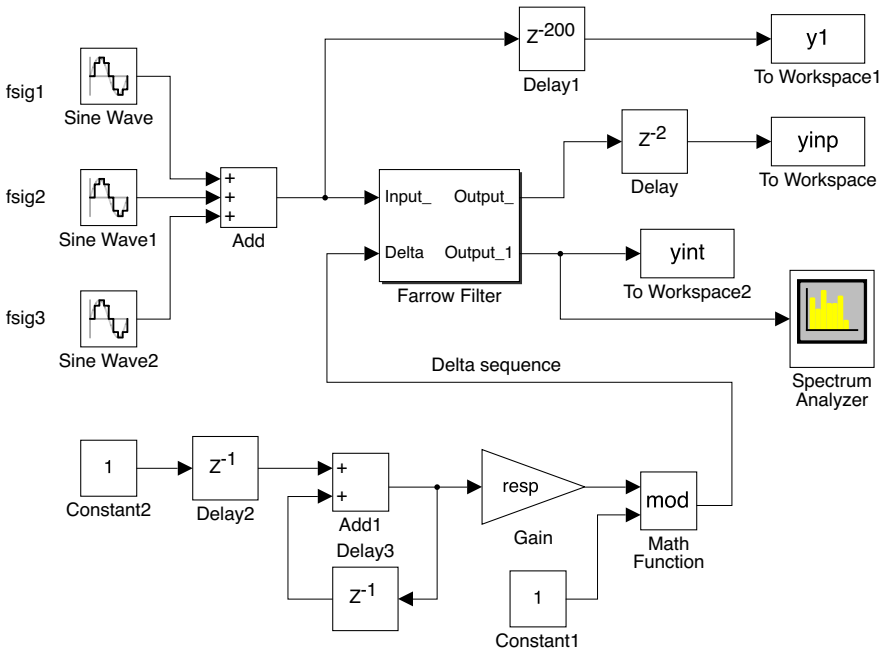

*Fig. 11: Simulink model of the SRC with Farrow Filter (*resample_sim1.slx, resample_sim_1.m)

```
%resp = 44.1/48;              % resp = Tout/Tin (Resampling factor)

%------ Matrix C of coefficients
C = [-0.1667, 0.5, -0.3333, 0;
    0.5,-1,-0.5,1;
    -0.5, 0.5, 1,0;
    0.1667, 0, -0.1667, 0];
%##########################################
% ------ Call up the simulation
sim('resample_sim1',[0,Tf]);
```

The `Farrow Filter` subsystem in the model is similar to the model in fig. 9, with the difference that the two outputs are no longer combined with a Mux block.

The input signals are sampled with a sampling frequency of `Ts/100` and practically form quasi-continuous signals. They are sampled in the subsystem with a sampling frequency `Ts = 1000` Hz.

The necessary delays are generated in the lower part of the model. The sequence `n` is generated with an accumulator, which is then multiplied by the SRC factor `resp` in the `Gain` block. The modulo one function is implemented in the `Math Function` block.

For a SRC factor `resp = 0.6` the results are shown in fig. 12 and fig. 13. Similar results are obtained for the SRC factor equal to 5/3, which are shown

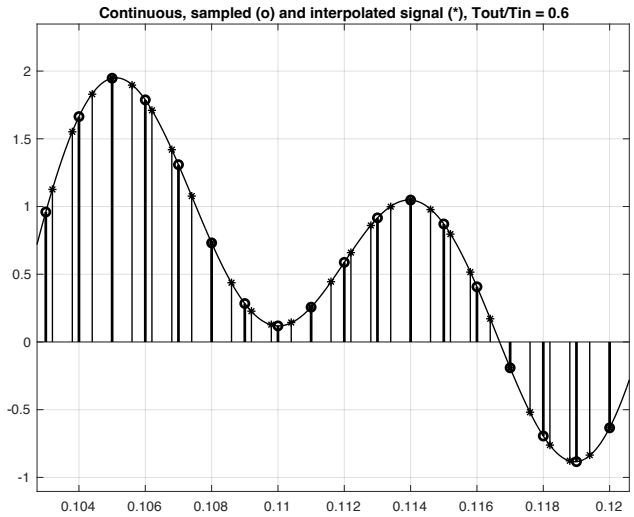

*Fig. 12: Continuous, sampled (o) and interpolated signal (*) for Tout/Tin = 0.6 (*resample_sim1.slx, resample_sim_1.m)

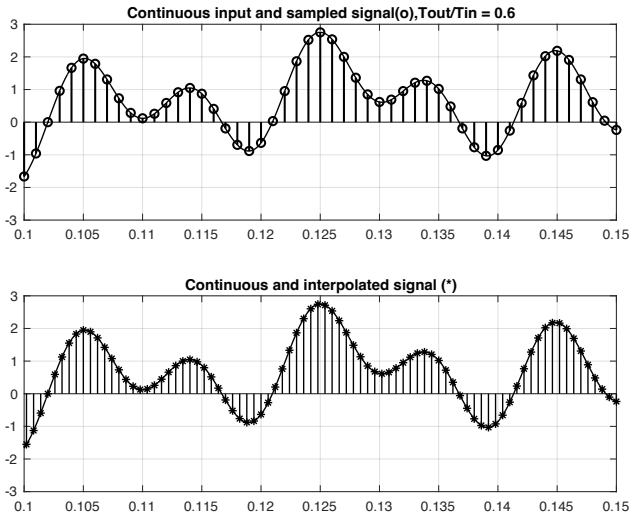

*Fig. 13: Continuous, sampled (o) and interpolated signal (*) for Tout/Tin = 0.6 (*resample_sim1.slx, resample_sim_1.m)

in fig. 14 and fig. 15.

The `Spectrum Analyzer` from the model can be used to estimate the quality of the conversion. For the SRC factor `resp = 5/3`, fig. 16 shows the power spectrum in dBW for signals with frequencies of 10 Hz, 50 Hz and 100 Hz. For amplitudes equal to one, the rms is $1/\sqrt{2}$ and accordingly the power of each

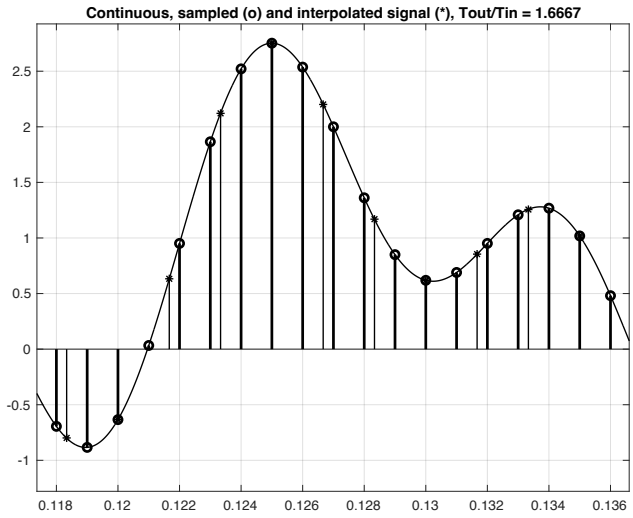

*Fig. 14: Continuous, sampled (o) and interpolated signal (*) for Tout/Tin = 5/3 (*resample_sim1.slx, resample_sim_1.m)

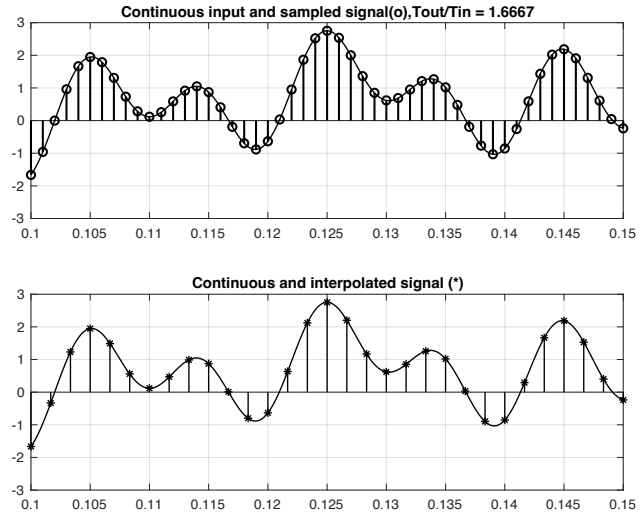

*Fig. 15: Continuous, sampled (o) and interpolated signal (*) for Tout/Tin = 5/3 (*resample_sim1.slx, resample_sim_1.m)

signal is 0.5 watts. In dBW, this power corresponds to a value of $10 * log_{10}(0.5)$ = -3 dBW. The first three lines in the spectrum correspond to the input signals with powers of -3 dBW and the remaining lines, attenuated with more than 80 dBW, are interferences due to the mixing products that are formed.

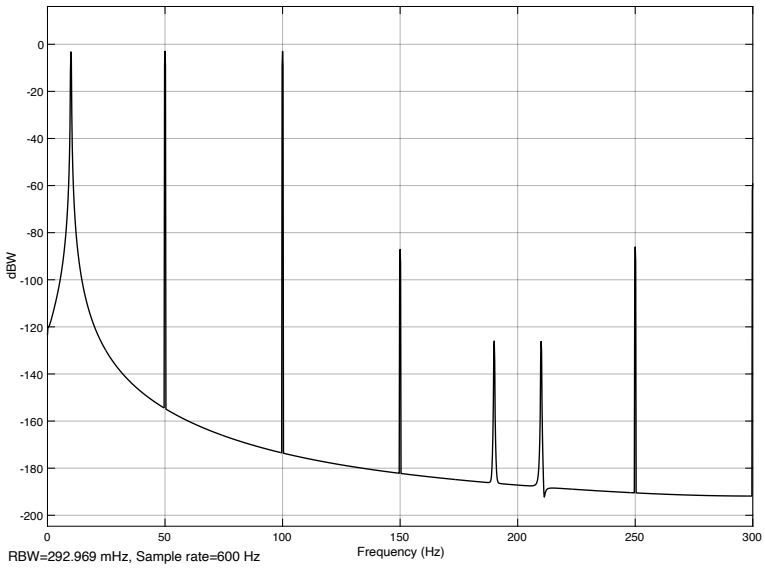

*Fig. 16: The Power-Spectrum of the resampled signal for Tout/Tin = 5/3 (*resample_sim1.slx, resample_sim_1.m)

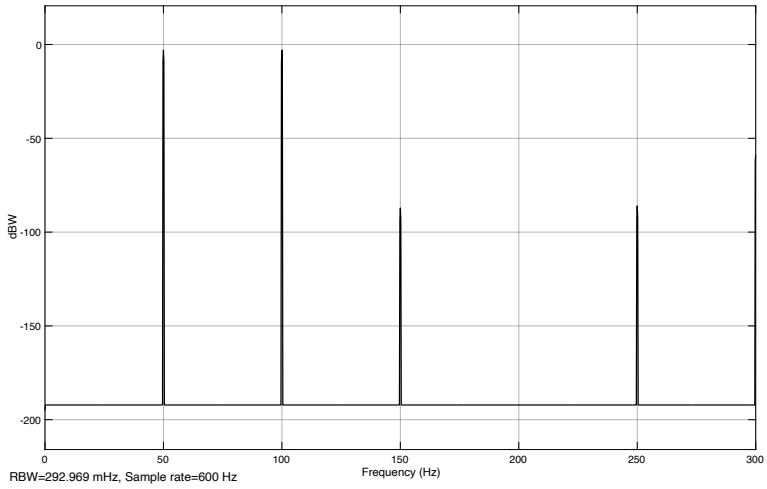

*Fig. 17: The Power-Spectrum of the resampled signal for Tout/Tin = 5/3 without the input signal of 10 Hz (*resample_sim1.slx, resample_sim_1.m)

The mixed products are relatively easy to identify here. The conversion errors with this method, which is based on an interpolation with a third-degree Lagrange polynomial, can be assumed to be the influence of a non-linearity. The mixed products result from this non-linearity. The fourth harmonic of the 50

Hz signal, together with the second harmonic of the 100 Hz signal, result in a signal with a frequency of 200 Hz. This signal mixed with the 10 Hz signal results in the two smaller lines around the 200 Hz frequency. The same 200 Hz signal mixed with the 50 Hz signal gives the two lines at 150 Hz and 250 Hz.

The reader can take one or two input signals one after the other and view the resulting mixed products. As an example fig. 17 shows the power spectrum for the case without the input signal of 10 Hz. With these frequencies, there is no longer any leakage effect compared to the spectrum in fig. 16, where this effect is clearly visible at the frequency of 10 Hz.

In these simulations, a sampling frequency of 1000 Hz was always assumed for the input signals and only the SRC was examined with various factors. A practical example for an SRC from 44.1 kHz to 48 kHz is simulated in the Simulink model `resample_sim3.slx` and script `resample_sim_3.m`. In this example, the input signals are sampled at a frequency of 44.1 kHz and the converted signal then has a sampling frequency of 48 kHz. The input signals must then have frequencies in the 44.1/2 kHz range in order not to violate the sampling theorem.

1.6 Summary

At the beginning of this article, the interpolation with Lagrange polynomial is shown, which is the basis for the fractional delay filter. Then this filter is developed and simulated with Simulink model for constant delays.

Furthermore, the Simulink model is changed in such a way that the delay value can be continuously changed in sync with the sampling rate of the output signal. This clock is generally different from the clock of the input signal. This property opens the possibility to use this filter for Sample Rate Conversion (SRC).

In the last part of the article the simulation of SRCs with different factors Tout/Tin = 0.6, 5/3 and 44.1/48 is realized.

The simulations with Simulink models are easy to understand because they represent block diagrams of the applications and are therefore very important for both education and industry.

The MATLAB scripts and the Simulink models are contained in the `Farrow_filter.zip` file, which can be accessed via the following link:
`https://dsprelated.com/blogimages/JosefHoffmann/farrow_programs.zip`

References

[1] Kreyszig E. *Advanced Engeneering Mathematics.* John Wiley & Sons, 2006.

[2] Farokh Marvasti, editor. *Nonuniform Sampling, Theory and Practice.* Kluwer Academic/Plenum Publishers, 2001.

[3] C. W. Farrow. A continuously variable digital delay element. *Proceedings IEEE International Symposium on Circuits and Systems*, pages 2641–2645, 1988.

[4] R.A. Losada. *Digital Filters with MATLAB*. The MathWorks, Inc., 2008.