Polyphase Decomposition

The Decomposition

• Consider an arbitrary sequence \( \{x[n]\} \) with a \( z \)-transform \( X(z) \) given by
  \[
  X(z) = \sum_{n=-\infty}^{\infty} x[n] z^{-n}.
  \]
  • We can rewrite \( X(z) \) as
  \[
  X(z) = \sum_{k=0}^{M-1} z^{-k} X_k(z^M)
  \]
  where
  \[
  X_k(z) = \sum_{n=-\infty}^{\infty} x_k[n] z^{-n} = \sum_{n=-\infty}^{\infty} x[Mn+k] z^{-n}
  \]
  \[0 \leq k \leq M-1\]

Polyphase Decomposition

• The subsequences \( \{x_k[n]\} \) are called the polyphase components of the parent sequence \( \{x[n]\} \)
  • The functions \( X_k(z) \), given by the \( z \)-transforms of \( \{x_k[n]\} \), are called the polyphase components of \( X(z) \)

Polyphase Decomposition

• The relation between the subsequences \( \{x_k[n]\} \) and the original sequence \( \{x[n]\} \) are given by
  \[x_k[n] = x[Mn+k], \quad 0 \leq k \leq M-1\]
  • In matrix form we can write
  \[
  X(z) = \begin{bmatrix}
  X_0(z^M) \\
  X_1(z^M) \\
  \vdots \\
  X_{M-1}(z^M)
  \end{bmatrix} =
  \begin{bmatrix}
  1 & z^{-1} & \ldots & z^{-(M-1)}
  \end{bmatrix}
  \begin{bmatrix}
  X_0(z^M) \\
  X_1(z^M) \\
  \vdots \\
  X_{M-1}(z^M)
  \end{bmatrix}
  \]

Polyphase Decomposition

• A multirate structural interpretation of the polyphase decomposition is given below

Polyphase Decomposition

• The polyphase decomposition of an FIR transfer function can be carried out by inspection
  • For example, consider a length-9 FIR transfer function:
    \[ H(z) = \sum_{n=0}^{8} h[n] z^{-n} \]

Polyphase Decomposition

• Its 4-branch polyphase decomposition is given by
  \[
  H(z) = E_0(z^4) + z^{-1} E_1(z^4) + z^{-2} E_2(z^4) + z^{-3} E_3(z^4)
  \]
  where
  \[
  E_0(z) = h[0] + h[4] z^{-1} + h[8] z^{-2}
  \]
  \[
  E_1(z) = h[1] + h[5] z^{-1}
  \]
  \[
  E_2(z) = h[2] + h[6] z^{-1}
  \]
  \[
  E_3(z) = h[3] + h[7] z^{-1}
  \]
Polyphase Decomposition

- The polyphase decomposition of an IIR transfer function \( H(z) = P(z)/D(z) \) is not that straightforward.
- One way to arrive at an \( M \)-branch polyphase decomposition of \( H(z) \) is to express it in the form \( P'(z)/D'(z^M) \) by multiplying \( P(z) \) and \( D(z) \) with an appropriately chosen polynomial and then apply an \( M \)-branch polyphase decomposition to \( P'(z) \).

Example - Consider:

\[
H(z) = \frac{1-2z^{-1}}{1+3z^{-1}}
\]

- To obtain a 2-band polyphase decomposition we rewrite \( H(z) \) as:

\[
H(z) = \frac{1-2z^{-1}}{1+3z^{-1}} = \frac{1-5z^{-1}+6z^{-2}}{1-9z^{-2} + \frac{6z^{-2}}{1-9z^{-2}} - \frac{5z^{-1}}{1-9z^{-2}}}
\]

- Therefore,

\[
H(z) = E_0(z^2) + z^{-1}E_1(z^2)
\]

where \( E_0(z) = \frac{1-5z^{-1}}{1-9z^{-2}} \), \( E_1(z) = \frac{-5}{1-9z^{-2}} \).

Note: The above approach increases the overall order and complexity of \( H(z) \).
- However, when used in certain multirate structures, the approach may result in a more computationally efficient structure.
- An alternative more attractive approach is discussed in the following example.

Example - Consider the transfer function of a 5th order Butterworth lowpass filter with a 3-dB cutoff frequency at 0.5π:

\[
H(z) = \frac{0.0527864(1+z^{-1})^3}{1+0.633436854z^{-1}+0.0557281z^{-2}}
\]

It is easy to show that \( H(z) \) can be expressed as:

\[
H(z) = \frac{1}{2} \left[ \frac{0.105573+z^{-2}}{1+0.105573z^{-1}} + z^{-1} \left( \frac{0.52786+z^{-2}}{1+0.52786z^{-1}} \right) \right]
\]

Therefore, \( H(z) \) can be expressed as:

\[
H(z) = E_0(z^2) + z^{-1}E_1(z^2)
\]

where

\[
E_0(z) = \frac{1}{2} \left( \frac{0.105573+z^{-1}}{1+0.105573z^{-1}} \right)
\]

\[
E_1(z) = \frac{1}{2} \left( \frac{0.52786+z^{-1}}{1+0.52786z^{-1}} \right)
\]

Note: In the above polyphase decomposition, branch transfer functions \( E_i(z) \) are stable allpass functions.
- Moreover, the decomposition has not increased the order of the overall transfer function \( H(z) \).
FIR Filter Structures Based on Polyphase Decomposition

• We shall demonstrate later that a parallel realization of an FIR transfer function $H(z)$ based on the polyphase decomposition can often result in computationally efficient multirate structures.

• Consider the $M$-branch Type I polyphase decomposition of $H(z)$:

$$H(z) = \sum_{k=0}^{M-1} z^{-k} E_k(z^M)$$

• A direct realization of $H(z)$ based on the Type I polyphase decomposition is shown below.

FIR Filter Structures Based on Polyphase Decomposition

• The transpose of the Type I polyphase FIR filter structure is indicated below.

• An alternative representation of the transpose structure shown on the previous slide is obtained using the notation $R_{i}(z^M) = E_{M-1-i}(z^M)$, $0 \leq i \leq M-1$.

• Substituting the above notation in the Type I polyphase decomposition we arrive at the Type II polyphase decomposition:

$$H(z) = \sum_{i=0}^{M-1} z^{-(M-1-i)} R_{i}(z^M)$$

• A direct realization of $H(z)$ based on the Type II polyphase decomposition is shown below.

FIR Filter Structures Based on Polyphase Decomposition

• A direct realization of $H(z)$ based on the Type II polyphase decomposition is shown below.

Computationally Efficient Decimators

• Consider first the single-stage factor-of-$M$ decimator structure shown below.

• We realize the lowpass filter $H(z)$ using the Type I polyphase structure as shown on the next slide.
Computationally Efficient Decimators

- Using the cascade equivalence #1 we arrive at the computationally efficient decimator structure shown below on the right.

\[ y[n] = x[n] \cdot F_{1}(z) + x[n-M] \cdot F_{2}(z) \]

Decimator structure based on Type I polyphase decomposition

To illustrate the computational efficiency of the modified decimator structure, assume \( H(z) \) to be a length-\( N \) structure and the input sampling period to be \( T = 1 \).

- Now the decimator output \( v[n] \) in the original structure is obtained by downsampling the filter output \( v[n] \) by a factor of \( M \).

\[ v[n] = y[Mn] \]

It is thus necessary to compute \( v[n] \) at
\[ n = \ldots, -2M, -M, 0, M, 2M, \ldots \]

- Computational requirements are therefore \( N \) multiplications and \( (N-1) \) additions per output sample being computed.

- However, as \( n \) increases, stored signals in the delay registers change.

Hence, all computations need to be completed in one sampling period, and for the following \((M-1)\) sampling periods the arithmetic units remain idle.

The modified decimator structure also requires \( N \) multiplications and \((N-1)\) additions per output sample being computed.

Computationally Efficient Decimators and Interpolators

- However, here the arithmetic units are operative at all instants of the output sampling period which is \( 1/M \) times that of the input sampling period.

- Similar savings are also obtained in the case of the interpolator structure developed using the polyphase decomposition.

Figures below show the computationally efficient interpolator structures.
Computationally Efficient Decimators and Interpolators

- More efficient interpolator and decimator structures can be realized by exploiting the symmetry of filter coefficients in the case of linear-phase filters \(H(z)\).
- Consider for example the realization of a factor-of-3 (\(M = 3\)) decimator using a length-12 Type 1 linear-phase FIR lowpass filter.

Computationally Efficient Decimators and Interpolators

- The corresponding transfer function is:
- A conventional polyphase decomposition of \(H(z)\) yields the following subfilters:
  - \(E_0(z) = h[0] + h[3]z^{-1} + h[5]z^{-2} + h[2]z^{-3}\)
  - \(E_2(z) = h[2] + h[5]z^{-1} + h[3]z^{-2} + h[0]z^{-3}\)

Computationally Efficient Decimators and Interpolators

- Note that \(E_1(z)\) still has a symmetric impulse response, whereas \(E_0(z)\) is the mirror image of \(E_2(z)\).
- These relations can be made use of in developing a computationally efficient realization using only 6 multipliers and 11 two-input adders as shown on the next slide.

A Useful Identity

- The cascade multirate structure shown below appears in a number of applications:
  \[x[n] \xrightarrow{\uparrow L} H(z) \xrightarrow{\downarrow L} y[n]\]
- Equivalent time-invariant digital filter obtained by expressing \(H(z)\) in its \(L\)-term Type I polyphase form \(\sum_{k=0}^{L-1} z^{-k} E_k(z^L)\) is shown below.
  \[x[n] \xrightarrow{E_0(z)} y[n]\]

Arbitrary-Rate Sampling Rate Converter

- The estimation of a discrete-time signal value at an arbitrary time instant between a consecutive pair of known samples can be solved by using some type of interpolation.
- In this approach an approximating continuous-time signal is formed from a set of known consecutive samples of the given discrete-time signal.
### Arbitrary-Rate Sampling Rate Converter

- The value of the approximating continuous-time signal is then evaluated at the desired time instant.
- This interpolation process can be directly implemented by designing a digital interpolation filter.

### Ideal Sampling Rate Converter

- In principle, a sampling rate conversion by an arbitrary conversion factor can be implemented as follows.
- The input digital signal is passed through an ideal analog reconstruction lowpass filter whose output is resampled at the desired output rate as indicated below.

$$\hat{x}_a(t) = \sum_{l=-\infty}^{\infty} x(t) g_a(t - lT)$$

If the analog filter is chosen to bandlimit its output to the frequency range $F_a < F_T/2$, its output $\hat{x}_a(t)$ can then be resampled at the rate $F_T$.

### Ideal Sampling Rate Converter

- Let the impulse response of the analog lowpass filter be denoted by $g_a(t)$.
- Then the output of the filter is given by

$$\hat{x}_a(t) = \sum_{l=-\infty}^{\infty} x(t) g_a(t - lT)$$

If the analog filter is chosen to bandlimit its output to the frequency range $F_a < F_T/2$, its output $\hat{x}_a(t)$ can then be resampled at the rate $F_T$.

### Ideal Sampling Rate Converter

- Problem statement: Given $N_1 + N_2 + 1$ input signal samples, $x[k]$, $k = -N_1, \ldots, N_2$, obtained by sampling an analog signal $x_a(t)$ at $t = t_k = t_0 + kT_{in}$, determine the sample value $x_a(t_0 + kT_{in}) = y[\alpha]$ at time instant $t' = t_0 + kT_{in}$ where $-N_1 \leq \alpha \leq N_2$.
- Figure on the next slide illustrates the interpolation process by an arbitrary factor.

### Ideal Sampling Rate Converter

- We describe next a commonly employed interpolation algorithm based on a finite weighted sum of input samples.
Lagrange Interpolation Algorithm

- Here, a polynomial approximation \( \hat{x}_a(t) \) to \( x_a(t) \) is defined as
  \[
  \hat{x}_a(t) = \sum_{k=-N_1}^{N_2} P_k(t)x[n + k]
  \]
  where \( P_k(t) \) are the Lagrange polynomials given by
  \[
  P_k(t) = \prod_{\ell=-N_1}^{-N_1} \left( \frac{t-t_\ell}{t_k-t_\ell} \right), \quad -N_1 \leq k \leq N_2
  \]

- Since
  \[
  P_k(t_r) = \begin{cases} 1, & k = r, \\ 0, & k \neq r, \end{cases} \quad -N_1 \leq r \leq N_2
  \]
  it follows from the previous 3 equations that
  \[
  \hat{x}_a(t_k) = x_a(t_k), \quad -N_1 \leq k \leq N_2
  \]

- From \( \hat{x}_a(t) = \sum_{k=-N_1}^{N_2} P_k(t)x[n + k] \)
  the value of \( x_a(t) \) at an arbitrary value \( t = t_0 + \alpha T_{in} \) is given by
  \[
  x_a(t') = \hat{x}_a(t_0 + \alpha T_{in}) = y[n] = \sum_{k=-N_1}^{N_2} P_k(\alpha)x[n + k]
  \]
  where
  \[
  P_k(\alpha) = P_k(0 + \alpha T_{in}) = \prod_{\ell=-N_1}^{-N_1} \left( \frac{\alpha - \ell}{k - \ell} \right)
  \]

- Example - Design a fractional-rate interpolator with an interpolation factor of 3/2 using a 3rd-order polynomial approximation with \( N_1 = 2 \) and \( N_2 = 1 \)
  - The output \( y[n] \) of the interpolator is thus computed using
    \[
    y[n] = P_{-2}(\alpha)x[n - 2] + P_{-1}(\alpha)x[n - 1]
    + P_0(\alpha)x[n] + P_1(\alpha)x[n + 1]
    \]

- Here, the Lagrange polynomials are given by
  \[
  \begin{align*}
  P_{-2}(\alpha) &= \frac{(\alpha+1)(\alpha-1)}{6} = \frac{1}{6}(-\alpha^3 + \alpha) \\
  P_{-1}(\alpha) &= \frac{(\alpha+2)(\alpha-1)}{2} = \frac{1}{2}(\alpha^3 + \alpha^2 - 2\alpha) \\
  P_0(\alpha) &= \frac{(\alpha+2)(\alpha+1)(\alpha-1)}{2} = \frac{1}{2}(\alpha^3 + 2\alpha^2 - \alpha - 2) \\
  P_1(\alpha) &= \frac{(\alpha+2)(\alpha+1)}{6} = \frac{1}{6}(\alpha^3 + 3\alpha^2 + \alpha)
  \end{align*}
  \]

- Figure below shows the locations of the samples of the input and the output for an interpolator with a conversion factor of 3/2
  - Locations of the output samples \( y[n] \), \( y[n+1] \), and \( y[n+2] \) in the input sample domain are marked with an arrow.
Lagrange Interpolation Algorithm

- From the figure on the previous slide it can be seen that the value of \( \alpha \) for computation of \( y[n] \), to be labeled \( \alpha_0 \), is 0
- Substituting this value of \( \alpha \) in the expressions for the Lagrange polynomial coefficients derived earlier we get
  \[
P_2(\alpha_0) = 0, \quad P_1(\alpha_0) = 0
\]
  \[
P_0(\alpha_0) = 1, \quad P_1(\alpha_0) = 0
\]

Lagrange Interpolation Algorithm

- The value of \( \alpha \) for computation of \( y[n+1] \), to be labeled \( \alpha_1 \), is 2/3
- Substituting this value of \( \alpha \) in the expressions for the Lagrange polynomial coefficients we get
  \[
P_2(\alpha_1) = 0.0617, \quad P_1(\alpha_1) = -0.2963
\]
  \[
P_0(\alpha_1) = 0.7407, \quad P_1(\alpha_1) = 0.4938
\]

Lagrange Interpolation Algorithm

- The value of \( \alpha \) for computation of \( y[n+2] \), to be labeled \( \alpha_2 \), is 4/3
- Substituting this value of \( \alpha \) in the expressions for the Lagrange polynomial coefficients we get
  \[
P_2(\alpha_2) = -0.1728, \quad P_1(\alpha_2) = 0.7407
\]
  \[
P_0(\alpha_2) = -1.2963, \quad P_1(\alpha_2) = 1.7284
\]

Lagrange Interpolation Algorithm

- The input-output relation of the interpolation filter can be compactly written as
  \[
\begin{bmatrix}
y[n]
y[n+1]
y[n+2]
\end{bmatrix}
= \begin{bmatrix}
x[n-2]
x[n-1]
x[n]
\end{bmatrix}
\begin{bmatrix}
P_2(\alpha_0) & P_1(\alpha_0) & P_0(\alpha_0) & R_3(\alpha_0)
\end{bmatrix}
\begin{bmatrix}
x[n-2]
x[n-1]
x[n]
\end{bmatrix}
\]

where \( H \) is the block coefficient matrix
- For the factor-of-3/2 interpolator, we have
  \[
H = \begin{bmatrix}
0.0617 & -0.2963 & 0.7407 & 0.4938
-0.1728 & 0.7407 & -1.2963 & 1.7284
\end{bmatrix}
\]

Lagrange Interpolation Algorithm

- It should be evident from an examination of
  \[
\begin{array}{cccc}
y[n+3]
y[n+4]
y[n+5]
\end{array}
= \begin{array}{cccc}
x[n]
x[n+1]
x[n+2]
\end{array}
\begin{array}{cccc}
P_2(\alpha_0) & P_1(\alpha_0) & P_0(\alpha_0) & R_3(\alpha_0)
\end{array}
\begin{array}{cccc}
x[n]
x[n+1]
x[n+2]
\end{array}
\]

that the filter coefficients to compute \( y[n+3], y[n+4], \) and \( y[n+5] \) are again given by the same block matrix \( H \)
Lagrange Interpolation Algorithm

• In practice, the overall system delay of the fractional rate interpolator will be 3 sample periods.
• Hence, the input-output relation of a practical interpolator will be

\[
\begin{bmatrix}
  y[n] \\
  y[n+1] \\
  y[n+2]
\end{bmatrix} = H \begin{bmatrix}
  x[n] \\
  x[n+1] \\
  x[n+2]
\end{bmatrix}
\]

49

Lagrange Interpolation Algorithm

• The next set of length-3 output vector is then computed using

\[
\begin{bmatrix}
  y[n+3] \\
  y[n+4] \\
  y[n+5]
\end{bmatrix} = H \begin{bmatrix}
  x[n+2] \\
  x[n+3] \\
  x[n+4]
\end{bmatrix}
\]

and so on.

50

Lagrange Interpolation Algorithm

• The desired interpolation filter is a time-varying filter.
• A realization of the interpolator is given below.

51

Lagrange Interpolation Algorithm

• A realization of the factor-of-3 interpolator in the form of a time-varying filter is shown below.

52

Lagrange Interpolation Algorithm

• The coefficients of the 5-th order time-varying FIR filter have a period of 3 and are assigned the values indicated below.

\[
\begin{array}{c|c|c|c|c|c}
\text{Time} & h_0[n] & h_1[n] & h_2[n] & h_3[n] & h_4[n] \\
\hline
n & n+1 & n+2 & n+3 & n+4 & n+5 \\
\hline
0 & P_0(a_0) & P_1(a_0) & P_2(a_0) & 0 & 0 \\
1 & P_0(a_0) & P_1(a_0) & P_2(a_0) & 0 & 0 \\
2 & P_0(a_0) & P_1(a_0) & P_2(a_0) & 0 & 0 \\
\end{array}
\]

53

Lagrange Interpolation Algorithm

• Because of the factor-of-2 down-sampling, the next set of input samples appearing at the input of the block filter H is \(x[n+2], x[n+3], x[n+4], \) and \(x[n+5]\).
**Lagrange Interpolation Algorithm**

- Substituting the expressions for the Lagrange polynomials in the output equation we arrive at
  \[
  y[n] = \alpha^3 \left( -\frac{1}{6} x[n-2] + \frac{1}{2} x[n-1] - \frac{1}{2} x[n] + \frac{1}{6} x[n+1] \right) \\
  + \alpha^2 \left( \frac{1}{2} x[n-1] - x[n] + \frac{1}{2} x[n+1] \right) \\
  + \alpha \left( \frac{1}{6} x[n-2] - x[n-1] + \frac{1}{2} x[n] + \frac{1}{3} x[n+1] \right) \\
  + x[n]
  \]

**Spline Interpolation**

- Here, a polynomial approximation \( \hat{x}_k(t) \) to \( x(t) \) is made using the B-spline functions as the basis
  \[ \text{The time instants } t_k, m \leq k \leq N + m, \text{ at which the samples } x(t_k) \text{ of the signal } x(t) \text{ are known, are called knots} \]
Spline Interpolation

• The polynomial approximation \( \hat{x}_a(t) \) to \( x_a(t) \) is given by

\[
\hat{x}_a(t) = \sum_{k=m}^{N+m} B_k^{(L)}(t)x_a(t_k)
\]

• The coefficients \( a_i \) in \( B_m^{(L)}(t) = \sum_{i=m}^{N+m} a_i \phi_i(t) \) are determined by imposing specific conditions at the knots \( t_m \) and \( t_{N+m} \).

Spline Interpolation

• It follows from the definition of the truncated power functions that \( B_m^{(L)}(t) = 0 \) for \( t \leq t_m \).

• An additional condition, \( B_m^{(L)}(t) = 0 \) for \( t \geq t_{N+m} \) is also imposed.

• Hence, for \( t \geq t_{N+m} \) we have

\[
\sum_{i=m}^{N+m} a_i (t - t_i)^L = 0
\]

Cubic B-Spline

• Here \( L = 3 \) and therefore \( N = 4 \).

• For notational convenience, we choose \( m = 0 \).

• In this case, \( \sum_{i=0}^{4} a_i (t - t_i)^3 = 0 \) in matrix form becomes

\[
\begin{bmatrix}
1 & 1 & 1 & 1 & 1 \\
t_0 & t_1 & t_2 & t_3 & t_4 \\
\frac{t_0^2}{2} & \frac{t_1^2}{2} & t_2^2 & t_3^2 & t_4^2 \\
\frac{t_0^3}{3} & \frac{t_1^3}{3} & t_2^3 & t_3^3 & t_4^3 \\
t_0 & t_1 & t_2 & t_3 & t_4
\end{bmatrix}
\begin{bmatrix}
a_0 \\
a_1 \\
a_2 \\
a_3 \\
a_4
\end{bmatrix} =
\begin{bmatrix}
0 \\
0 \\
0 \\
0
\end{bmatrix}
\]

Cubic B-Spline

• Considering \( a_4 \) to be the free parameter, we rewrite the matrix equation in Slide 59 as

\[
\begin{bmatrix}
1 & 1 & 1 & 1 \\
t_0 & t_1 & t_2 & t_3 \\
\frac{t_0^2}{2} & \frac{t_1^2}{2} & \frac{t_2^2}{2} & \frac{t_3^2}{2} \\
\frac{t_0^3}{3} & \frac{t_1^3}{3} & \frac{t_2^3}{3} & \frac{t_3^3}{3} \\
t_0 & t_1 & t_2 & t_3
\end{bmatrix}
\begin{bmatrix}
a_0 \\
a_1 \\
a_2 \\
a_3 \\
a_4
\end{bmatrix} =
\begin{bmatrix}
1 \\
t_4 \\
t_4^2 \\
t_4^3 \\
t_4
\end{bmatrix}
\]

• We can solve the above matrix equation for \( a_i \) using Cramer’s rule.
Cubic B-Spline

• For example,

\[
\begin{pmatrix}
1 & 1 & 1 & 1 \\
1 & t_4 & t_4^2 & t_4^3 \\
1 & t_1 & t_1^2 & t_1^3 \\
1 & t_2 & t_2^2 & t_2^3 \\
1 & t_3 & t_3^2 & t_3^3
\end{pmatrix}
\]

\[a_0 = -a_4\]

• The numerator and the denominator of the previous equation are determinants of Vandermonde matrices and have nonzero values if the knots \( t_i \) are distinct.

• It can be shown that

\[a_0 = -a_4 \frac{(t_4 - t_1)(t_4 - t_2)(t_4 - t_3)}{(t_0 - t_1)(t_0 - t_2)(t_0 - t_3)}\]

Cubic B-Spline

• Choosing the free parameter \( a_4 \) to be

\[a_4 = \frac{1}{(t_4 - t_0)(t_4 - t_1)(t_4 - t_2)(t_4 - t_3)}\]

we arrive at

\[a_0 = \frac{1}{(t_0 - t_1)(t_0 - t_2)(t_0 - t_3)(t_0 - t_4)}\]

• In a similar manner the expressions for the remaining 3 coefficients can be derived and are given in the next slide.

B-Spline

• In the general case, the coefficients are given by

\[a_i = \frac{(-1)^{L+1}}{N+m} \prod_{k=m+1}^{i} (t_i - t_k)\]

and the \( L \)th order B-spline function is given by

\[B_m^{(L)}(t) = (-1)^{L+1} \sum_{i=m}^{N+m} \frac{(t - t_i)^L}{\prod_{k=m+1}^{i} (t_i - t_k)}\]

Normalized B-Spline

• Since the maximum value of the B-spline decreases with increasing \( L \), it is a common practice to use instead, a normalized form given by

\[B_m^{(L)}(t) = (t_{N+m} - t_m)B_m^{(L)}(t)\]

for interpolation.

• In digital signal processing applications, the knots are uniformly spaced at sampling instants.
Second-Order Normalized B-Spline

- Here $L = 2$ and hence, $N = 3$
- The knots are at $t_i = i$, $m \leq i \leq m + 3$
- As a result

$$a_{m+1} = -\frac{1}{(m+1-m-1)(m+1-m-2)(m+1-m-3)} = \frac{1}{6}$$
$$a_{m+2} = -\frac{1}{(m+2-m-1)(m+2-m-2)(m+2-m-3)} = \frac{1}{2}$$
$$a_{m+3} = -\frac{1}{(m+3-m-1)(m+3-m-2)(m+3-m-3)} = \frac{1}{6}$$

- The expression for the second-order B-spline is given below:

$$\beta_{m}^{(2)}(t) = (m+3-m)\beta_{m}^{(2)}(t) = \begin{cases} 0, & m \leq i \leq m+1 \\ \frac{1}{6}(t-i)^2 + \frac{1}{2}(t-i)^3 & m+1 \leq i \leq m+2 \\ -\frac{1}{6}(t-i)^2 + \frac{1}{2}(t-i)^3 & m+2 \leq i \leq m+3 \\ 0, & i \geq m+3 \end{cases}$$

- A plot of $\beta_{m}^{(2)}(t)$ and the corresponding power functions for several values of $m$ are shown in the next slide.

Spline Interpolation

- The interpolation formula is obtained by forming a linear combination of the normalized B-splines weighted by the known values of the function $x_a(t)$ at the knots $t_k = n + k$
- The interpolated value at the time instant $t' = t_0 + \alpha T_{in}$ is given by

$$\hat{x}_a(t') = \hat{x}_a(t_0 + \alpha T_{in})$$
$$y[n] = \sum_{k=m}^{L+m+1} \beta_k^{(L)}(t_0 + \alpha T_{in}) x[n + k]$$

- It should be noted that, unlike the Lagrange interpolation algorithm, in the case of spline interpolation, $\hat{x}_a(t_k) \neq x_a(t_k)$.
- We illustrate next the development of the interpolation formula using the normalized second-order B-spline.
Interpolation Using Second-Order B-Spline

• The interpolation process is illustrated below:

\[ y[n] = \sum_{k=-1}^{1} \beta_k^2(\alpha)x[n+k] \]

where
\[ \beta_{-1}^2(\alpha) = \frac{\alpha^2}{2} - \alpha + \frac{1}{2} \]
\[ \beta_0^2(\alpha) = -\alpha^2 + \alpha + \frac{1}{2} \]
\[ \beta_1^2(\alpha) = \frac{\alpha^2}{2} \]
\[ \beta_2^2(\alpha) = 0 \]

• The equation in the previous slide can be rewritten as:
\[ y[n] = \left( \frac{1}{2} x[n-1] + \frac{1}{2} x[n] \right) + \alpha x[n-1] + \frac{1}{2} x[n+1] \]
leading to the Farrow structure shown on the next slide:

\[ H_0(z) = \frac{1}{2} z^{-1} - 1 + \frac{1}{2} z \]
\[ H_1(z) = -z^{-1} + 1 \]
\[ H_2(z) = \frac{1}{2} z^{-1} + \frac{1}{2} \]

Interpolation Using Second-Order B-Spline

• As can be seen from the above figure, the position \( t' = 1 + \alpha \) of the desired value \( y[n] \) of \( x_d(t) \) is between the knots \( t = 1 \) and \( t = 2 \)
• Here we thus have:
\[ y[n] = \sum_{k=-1}^{1} \beta_k^2(\alpha)x[n+k] \]
Arbitrary-Rate Sampling Rate Converter

Practical Considerations

• A direct design of a fractional-rate sampling rate converter in most applications is not practical

• This is due to two main reasons:
  – length of the time-varying filter needed is usually very large
  – real-time computation of the corresponding filter coefficients is nearly impossible

As a result, the fractional-rate sampling rate converter is almost realized in a hybrid form as indicated below for the case of an interpolator

The digital sampling rate converter can be implemented in a multistage form to reduce the computational complexity