Laboratory 1
Wavelet Transform

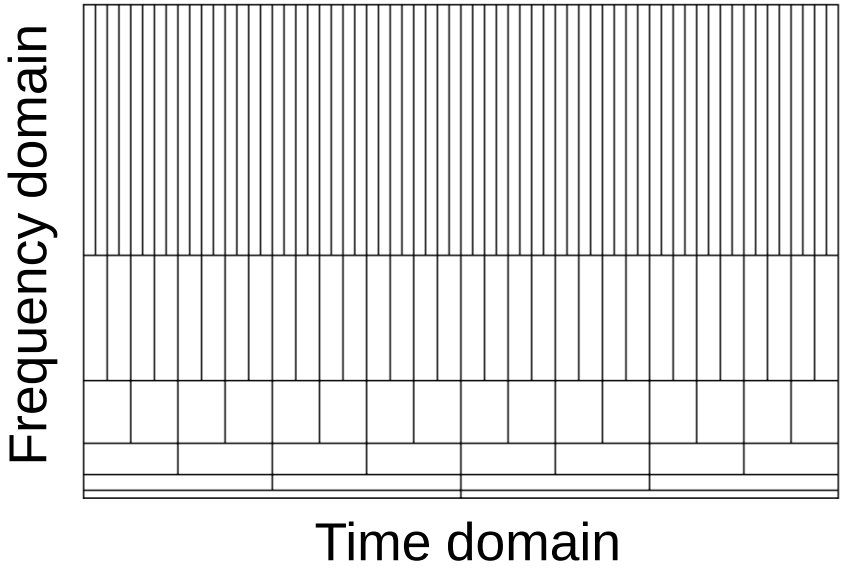
Federico Mason
federico.mason@unipd.it

Dyadic Wavelet Transform

In the dyadic wavelet transform, we **discretize** the wavelet coefficient as

$$W_x[j,n] = \sum_{m=0}^{N-1} x[m] \frac{1}{\sqrt{2^j}} \psi^* \left(\frac{T_s \cdot m - 2^j \cdot T_s \cdot n}{2^j} \right), \qquad n, j \in \mathbb{Z}^+$$

- At any scale j, the signal is segmented into frames of 2^j samples, with a bandwidth proportional to $\approx 1/(T_\psi \cdot 2^j)$
- At the following scale $j+1$, we obtain two sub-frames of 2^{j+1} samples, with bandwidth proportional to $\approx 1/(T_\psi \cdot 2^{j+1})$



Dyadic Wavelet Transform

In the laboratory, we considered an Haar wavelet with support equal to $T_s = T_\psi$

- This implies that the **dyadic** wavelet atom associated with scale $j = 1$ is associated with a support of $T_\psi \cdot 2^j = 2 \cdot T_s$ seconds and $T_\psi \cdot \frac{2^j}{T_s} = 2$ samples

The Wavelet transform $W_x[j, n]$ of $x[\cdot]$ at sample n and scale $j = 1$ is defined as

$$W_x[1, n] = \sum_{m=0}^{N-1} x[m] \frac{1}{\sqrt{2}} \psi^* \left(\frac{T_s \cdot m - 2 \cdot T_s \cdot n}{2} \right), \qquad n \in \mathbb{Z}^+$$

where $\psi^* \left(\frac{T_s \cdot m - 2 \cdot T_s \cdot n}{2} \right)$ has non-zero values only when $\frac{T_s \cdot m - 2 \cdot T_s \cdot n}{2} \in \left[-{}^{T_s}/_2 , +{}^{T_s}/_2 \right[$

- This implies that m must be equal to $2n$ or $2n - 1$

Dyadic Wavelet Transform

In the laboratory, we considered an Haar wavelet with support equal to $T_s = T_\psi$

- This implies that the **dyadic** wavelet atom associated with a generic scale j is associated with a support of $T_\psi \cdot 2^j = 2^j \cdot T_s$ seconds and $T_\psi \cdot \frac{2^j}{T_s} = 2^j$ samples

The Wavelet transform $W_x[j,n]$ of $x[\cdot]$ at sample n and scale j is defined as

$$W_x[j,n] = \sum_{m=0}^{N-1} x[m] \frac{1}{\sqrt{2^j}} \psi^* \left(\frac{T_s \cdot m - 2^j \cdot T_s \cdot n}{2^j} \right), \qquad n \in \mathbb{Z}^+$$

where $\psi^* \left(\frac{T_s \cdot m - 2^j \cdot T_s \cdot n}{2^j} \right)$ has non-zero values only when $\frac{T_s \cdot m - 2^j \cdot T_s \cdot n}{2^j} \in \left[-{}^{T_s}/_2 , +{}^{T_s}/_2 \right[$

- This implies that m must be in $\{2^j n - 2^{j-1}, \dots, 2^j n, \dots, 2^j n + 2^{j-1} - 1\}$

Dyadic Decomposition

Given the dyadic WT of $x[\cdot]$ for the scale $j \in \{1, 2, 3, \dots, J\}$, we can approximate $x[n]$ as

$$x(n) \approx \frac{\log_e 2}{C_\psi} \sum_{j=1}^{J} \frac{1}{2^j} \sum_{m=\lceil n/2^j \rceil} W_x[j,m]\psi_{j,m}[n] + \frac{1}{2^J} \sum_{m=\lceil n/2^J \rceil} \mathcal{L}_x[J,m]\phi_{J,m}[n]$$

In the case of Haar wavelet, $C_\psi = \log_e 2$

This equation is implemented by the method ***get_reconstructed_signal_v1***

```
def get_reconstructed_signal_v1(signal_length: int,
                                max_j: int,
                                detail_coefficients: np.ndarray,
                                approx_coefficients: np.ndarray,
                                sampling_frequency: int,
                                sampling_period: float):

  reconstructed_signal = np.zeros(signal_length)

  j_values = np.arange(1, max_j+1)  # Consider all the scale from 1 to max_j

  coefficient_masks = np.zeros((len(j_values), 2, int(signal_length / 2)))  # Variable to monitor the number of wavelet coefficients used for the reconstruction
```

Dyadic Decomposition

The first part of the method implements the first part of the equation, i.e., add to each signal sample $x[n]$ the details associated with all the scales from $j = 1$ to $j = J$

```
for sample_index in range(signal_length):
  for j_index, j in enumerate(j_values):
    m = np.floor((sample_index - 2 ** (j-1)) / (2**j)) + 1  # Select the time shift associated with the sample
    atom_domain, atom_wavelet = get_haar_dyadic_wavelet(j, m, sampling_period) # Get the wavelet atom associated with (j,m)
    if atom_domain[0] > 0:
      # We fill with zeros the portion of the signal domain that is not covered by the wavelet
      atom_wavelet = np.concatenate((np.zeros(int(round(sampling_frequency * atom_domain[0]))), atom_wavelet))
    else:
      # We do not consider the portion of the wavelet domain that is lower than 0
      atom_wavelet = atom_wavelet[int(np.argmin(np.abs(atom_domain))):]
    m_values = np.arange(int(signal_length / (2**j)))  # Time shifts associated with scale j
    if m in m_values:
      m_index = np.argwhere(m_values == m)
      # We add the detail coefficient, weighted by the wavelet functions, to reconstruct the signal
      reconstructed_signal[sample_index] += detail_coefficients[j_index, m_index] * atom_wavelet[sample_index] / (2 ** j)
      coefficient_masks[j_index, 0, m_index] = 1
```

$$x[n] \mathrel{+}= \frac{\log_e 2}{C_\psi} \sum_{j=1}^{J} \frac{1}{2^j} \sum_{m=\lfloor n/2^j \rfloor} W_x[j,m]\psi_{j,m}[n]$$

Dyadic Decomposition

The first part of the method implements the first part of the equation, i.e., add to each signal sample $x[n]$ the details associated with all the scales from $j = 1$ to $j = J$

```
for sample_index in range(signal_length):
  for j_index, j in enumerate(j_values):
    m = np.floor((sample_index - 2 ** (j-1)) / (2**j)) + 1  # Select the time shift associated with the sample
    atom_domain, atom_wavelet = get_haar_dyadic_wavelet(j, m, sampling_period) # Get the wavelet atom associated with (j,m)
    if atom_domain[0] > 0:
      # We fill with zeros the portion of the signal domain that is not covered by the wavelet
      atom_wavelet = np.concatenate((np.zeros(int(round(sampling_frequency * atom_domain[0]))), atom_wavelet))
    else:
      # We do not consider the portion of the wavelet domain that is lower than 0
      atom_wavelet = atom_wavelet[int(np.argmin(np.abs(atom_domain))):]
    m_values = np.arange(int(signal_length / (2**j)))  # Time shifts associated with scale j
    if m in m_values:
      m_index = np.argwhere(m_values == m)
      # We add the detail coefficient, weighted by the wavelet functions, to reconstruct the signal
      reconstructed_signal[sample_index] += detail_coefficients[j_index, m_index] * atom_wavelet[sample_index] / (2 ** j)
      coefficient_masks[j_index, 0, m_index] = 1
```

$$x[n] \mathrel{+}= \frac{\log_e 2}{C_\psi} \sum_{j=1}^{J} \frac{1}{2^j} \sum_{m=\lfloor n/2^j \rfloor} W_x[j,m]\psi_{j,m}[n]$$

Dyadic Decomposition

The first part of the method implements the first part of the equation, i.e., add to each signal sample $x[n]$ the details associated with all the scales from $j = 1$ to $j = J$

```
for sample_index in range(signal_length):
  for j_index, j in enumerate(j_values):
    m = np.floor((sample_index - 2 ** (j-1)) / (2**j)) + 1  # Select the time shift associated with the sample
    atom_domain, atom_wavelet = get_haar_dyadic_wavelet(j, m, sampling_period) # Get the wavelet atom associated with (j,m)
    if atom_domain[0] > 0:
      # We fill with zeros the portion of the signal domain that is not covered by the wavelet
      atom_wavelet = np.concatenate((np.zeros(int(round(sampling_frequency * atom_domain[0]))), atom_wavelet))
    else:
      # We do not consider the portion of the wavelet domain that is lower than 0
      atom_wavelet = atom_wavelet[int(np.argmin(np.abs(atom_domain))):]
    m_values = np.arange(int(signal_length / (2**j)))  # Time shifts associated with scale j
    if m in m_values:
      m_index = np.argwhere(m_values == m)
      # We add the detail coefficient, weighted by the wavelet functions, to reconstruct the signal
      reconstructed_signal[sample_index] += detail_coefficients[j_index, m_index] * atom_wavelet[sample_index] / (2 ** j)
      coefficient_masks[j_index, 0, m_index] = 1
```

The only scope of this is to align the Wavelet domain with the signal domain, i.e., ensure that the first sample of the Wavelet atom is associated with time $t = 0$

Dyadic Decomposition

The second part of the method implements the second part of the equation, i.e., add to each signal sample $x[n]$ the approximation associated with the maximum scale $j = J$

```
  # Only for the max j value
  if m in m_values:
    m_index = np.argwhere(m_values == m)
    atom_domain, atom_scaling_function = get_haar_dyadic_scaling_function(j, m, sampling_period) # Get the scaling function associated with (j,m)
    if atom_domain[0] > 0:
      # We fill with zeros the portion of the signal domain that is not covered by the wavelet
      atom_scaling_function = np.concatenate((np.zeros(int(round(sampling_frequency * atom_domain[0]))), atom_scaling_function))
    else:
      # We do not consider the portion of the wavelet domain that is lower than 0
      atom_scaling_function  = atom_scaling_function[int(np.argmin(np.abs(atom_domain))):]
    # We add the approximation coefficient associated with the maximum scale, weighted by the scaling functions, to reconstruct the signal
    reconstructed_signal[sample_index] += approx_coefficients[j_index, m_index] * atom_scaling_function[sample_index] / (2 ** j)
    coefficient_masks[j_index, 1, m_index] = 1
# We observe that np.sum(coefficient_masks) is the total number of coefficients used
# and, thus, represents the size of the reconstructed signal
return reconstructed_signal, np.sum(coefficient_masks)
```

$$x[n] \mathrel{+}= \frac{1}{2^J} \sum_{m=\lfloor n/2^J \rfloor} \mathcal{L}_x[J,m]\phi_{J,m}[n]$$

Dyadic Decomposition

The second part of the method implements the second part of the equation, i.e., add to each signal sample $x[n]$ the approximation associated with the maximum scale $j = J$

```
  # Only for the max j value
  if m in m_values:
    m_index = np.argwhere(m_values == m)
    atom_domain, atom_scaling_function = get_haar_dyadic_scaling_function(j, m, sampling_period) # Get the scaling function associated with (j,m)
    if atom_domain[0] > 0:
      # We fill with zeros the portion of the signal domain that is not covered by the wavelet
      atom_scaling_function = np.concatenate((np.zeros(int(round(sampling_frequency * atom_domain[0]))), atom_scaling_function))
    else:
      # We do not consider the portion of the wavelet domain that is lower than 0
      atom_scaling_function  = atom_scaling_function[int(np.argmin(np.abs(atom_domain))):]
    # We add the approximation coefficient associated with the maximum scale, weighted by the scaling functions, to reconstruct the signal
    reconstructed_signal[sample_index] += approx_coefficients[j_index, m_index] * atom_scaling_function[sample_index] / (2 ** j)
    coefficient_masks[j_index, 1, m_index] = 1
# We observe that np.sum(coefficient_masks) is the total number of coefficients used
# and, thus, represents the size of the reconstructed signal
return reconstructed_signal, np.sum(coefficient_masks)
```

$$x[n] \mathrel{+}= \frac{1}{2^J} \sum_{m=\lfloor n/2^J \rfloor} \mathcal{L}_x[J,m]\,\phi_{J,m}[n]$$

Dyadic Decomposition

The second part of the method implements the second part of the equation, i.e., add to each signal sample $x[n]$ the approximation associated with the maximum scale $j = J$

```
  # Only for the max j value
  if m in m_values:
    m_index = np.argwhere(m_values == m)
    atom_domain, atom_scaling_function = get_haar_dyadic_scaling_function(j, m, sampling_period) # Get the scaling function associated with (j,m)
    if atom_domain[0] > 0:
      # We fill with zeros the portion of the signal domain that is not covered by the wavelet
      atom_scaling_function = np.concatenate((np.zeros(int(round(sampling_frequency * atom_domain[0]))), atom_scaling_function))
    else:
      # We do not consider the portion of the wavelet domain that is lower than 0
      atom_scaling_function  = atom_scaling_function[int(np.argmin(np.abs(atom_domain))):]
    # We add the approximation coefficient associated with the maximum scale, weighted by the scaling functions, to reconstruct the signal
    reconstructed_signal[sample_index] += approx_coefficients[j_index, m_index] * atom_scaling_function[sample_index] / (2 ** j)
    coefficient_masks[j_index, 1, m_index] = 1
# We observe that np.sum(coefficient_masks) is the total number of coefficients used
# and, thus, represents the size of the reconstructed signal
return reconstructed_signal, np.sum(coefficient_masks)
```

The only scope of this is to align the Wavelet domain with the signal domain, i.e., ensure that the first sample of the Wavelet atom is associated with time $t = 0$