HEILBRONN UNIVERSITY
OF APPLIED SCIENCES

Faculty of Engineering (TE)
Automotive Systems Engineering

Project for Digital Signal Processing

Project Documentation
Master Automotive Systems Engineering
by

Jakob Kurz & Mattis Tom Ritter
(Matriculation-Nr. 210262 & 210265)

presented in Wintersemester 2024 / -25 at

Hochschule Heilbronn

Reviewer:	Prof. Dr. Volker Stahl
Levy:	31. December 2024

Table of Contents

List of Abbreviations

DFT Discrete Fourier Transform

FFT Fast Fourier Transform

List of Figures

1. Basic Implementations

1.1 DFT vs. FFT

1.1.1 Implementation Overview

The DFT and FFT implementations are provided in `dft.py` and `fft.py`, respectively:

- **DFT**: Computes the Fourier Transform with $O(N^2)$ complexity using a direct summation formula.
- **FFT**:
  - **Recursive FFT**: Implements a divide-and-conquer approach, reducing complexity to $O(N \log N)$.
  - **Iterative FFT**: Uses iterative butterfly operations for further optimization, avoiding recursion overhead.

1.1.2 Verification of Similarities

The Python script `test_dft_fft.py` validates the numerical equivalence of the DFT and FFT implementations:

- A sinusoidal test signal is processed by all three methods (DFT, recursive FFT, and iterative FFT).
- Results are compared element-wise within a tolerance of 10^{-9}.
- Tests confirm that all implementations produce consistent outputs, differing only by negligible numerical errors.

The visualization script `vis_dft_fft_simmiliar.py` further illustrates the similarities:

- **Frequency Domain Comparison (Fig. 1.1 Left)**: All methods accurately identify the signal's frequency components.
- **Time Domain Reconstruction (Fig. 1.1 Right)**: All methods reconstruct the original signal with minimal deviations.

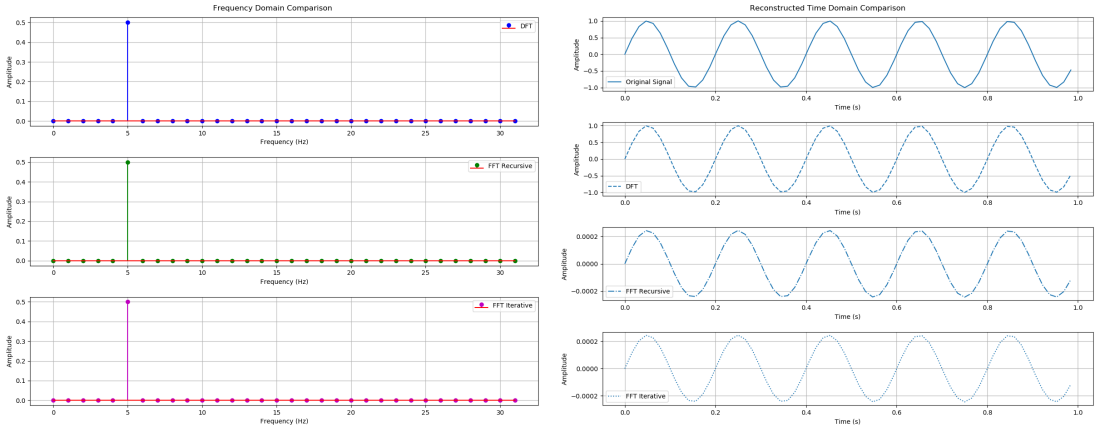

Figure 1.1: Frequency domain comparison (Left) and Time-domain reconstruction validation (Right)

1.1.3 Comparison of Differences

To highlight differences, the script `vis_dft_fft_different.py` visualizes computational time and memory usage for the DFT and FFT.

Computational Time and Memory Comparison

Figure 1.2 shows the execution time and memory usage for a fixed signal length:

- The DFT requires significantly more time due to its $O(N^2)$ complexity, while both FFT implementations achieve $O(N \log N)$.
- The iterative FFT is slightly faster than the recursive FFT.
- Memory usage for the FFT is lower than the DFT, with the iterative FFT requiring the least memory.

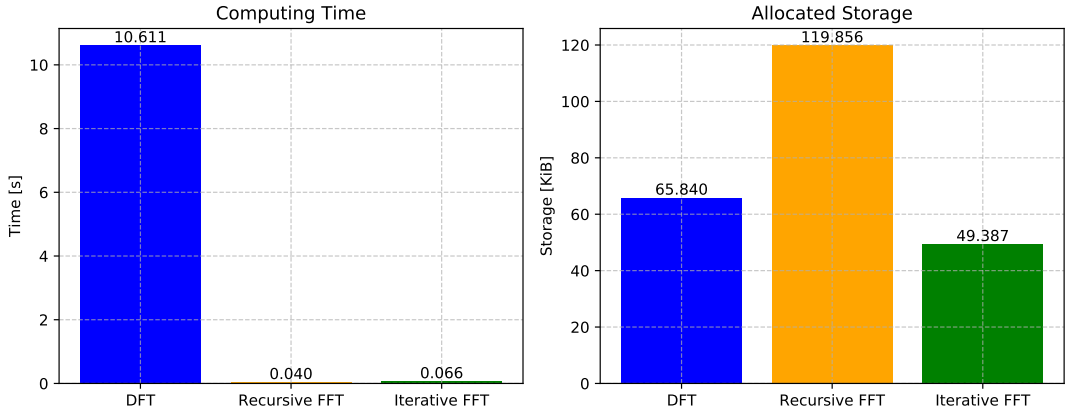

Figure 1.2: (Left) Computational time for fixed signal length. (Right) Memory usage comparison.

Scaling with Signal Length

Figure 1.3 examines execution time and memory usage as signal length increases:

- Execution time for the DFT grows quadratically, while the FFT scales logarithmically.
- Memory usage for the DFT increases linearly, whereas both FFT methods maintain near-constant memory requirements.
- The iterative FFT consistently outperforms the recursive FFT in both time and memory usage.

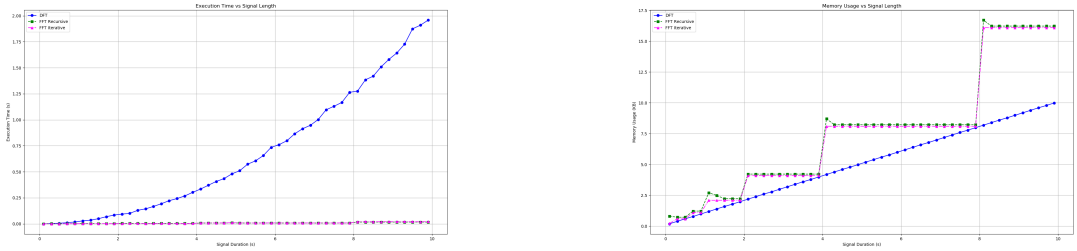

Figure 1.3: (Left) Execution time vs. signal length. (Right) Memory usage vs. signal length.

1.1.4 Conclusion

The DFT and FFT produce equivalent results in transforming time-domain signals into the frequency domain. However, the FFT offers significant advantages in computational efficiency and memory usage, making it more suitable for large-scale applications. The provided Python scripts and visualizations highlight these differences, offering a practical comparison between the methods.

1.2 Modulation and Demodulation (Non- and Quadrature)

1.2.1 Implementation Overview

Modulation and demodulation are essential processes for signal transformation during transmission and recovery. This implementation includes non-quadrature and quadrature approaches, developed in the Python script `modulation.py`. The test script `test_modulation.py` validates functionality, while `visualize_modulation.py` provides visual insights into the signal transformations.

1.2.2 Purpose and Testing

The implementation covers:

- **Modulation:** Encoding a signal onto a carrier wave.
- **Demodulation:** Recovering the original signal, verifying expected scaling effects.
- **Quadrature Modulation:** Utilizing orthogonal components ($f(t)$ and $g(t)$) for efficient dual-channel transmission.

Tests in `test_modulation.py` confirm that:

- Modulated signals exhibit the expected transformations.
- Demodulated signals accurately reconstruct the originals.
- Quadrature modulation and demodulation handle both channels correctly.

1.2.3 Visualization of Modulation Processes

Figure 1.4 illustrates non-quadrature and quadrature modulation/demodulation. The left panel shows the original, modulated, and demodulated signals for the non-quadrature case, where phase shifts and amplitude scaling are evident. The right panel depicts quadrature modulation and demodulation, showcasing the successful separation and reconstruction of two orthogonal signal components. Both visualizations confirm the correctness and reliability of the implementations.

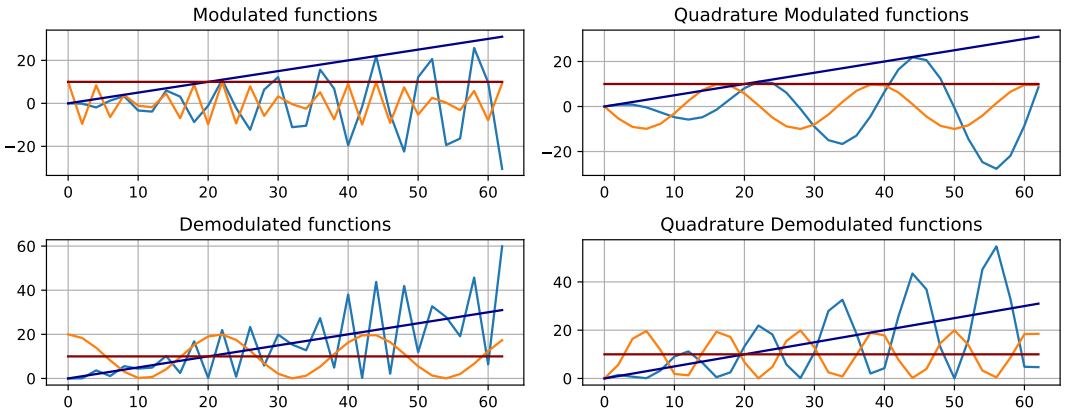

Figure 1.4: (Left) Non-quadrature modulation and demodulation. (Right) Quadrature modulation and demodulation.

2. Low-Pass Filter

The filter is implemented by convolving the input signal with the *sinc* function. By filtering an impulse signal, the filter can be obtained. Figure 2.1 (left) shows the filter with and without a Hamming window.

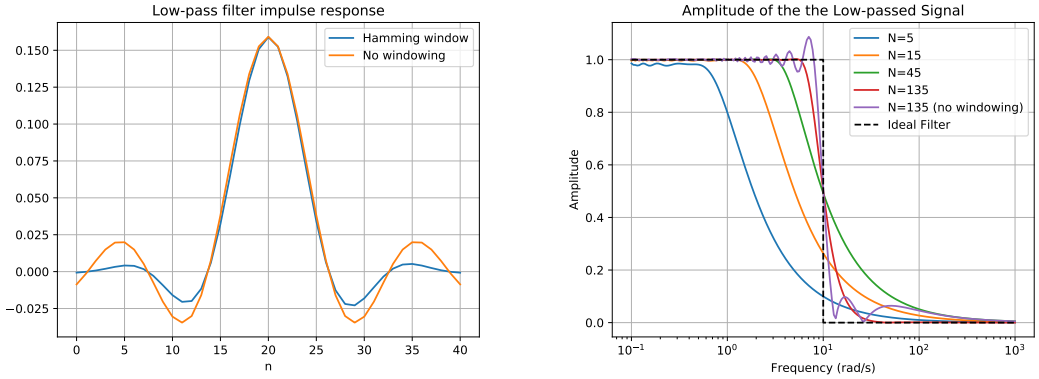

Figure 2.1: Impulse response of the low-pass filter with N=20 (left). Amplitude of the low-passed signal with different filter lengths (right).

The frequency response of the low-pass filter is determined by filtering cosine waves with increasing frequencies and calculating the highest amplitude of the output signal. This is shown for different filter lengths in Figure 2.1 (right). Here, the cut-off frequency is 10 rad/s. It can be observed that the steepness of the transition band increases with the filter length. If no Hamming window is applied, there are ripples in the passband and the stopband.

The performance of the low-pass filter highly depends on the implementation of the convolution. Table 2.1 shows the timing comparison when using fast convolution and convolution in time-domain. For the given parameters, the fast convolution is on average more than 2.4 times faster then in time-domain.

	Fast Convolution	**Time-Domain Convolution**
Average time	0.2997 s	0.7241 s
Standard deviation	0.0397 s	0.0603 s
Min time	0.2598 s	0.6684 s
Max time	0.5171 s	1.269 s
Median time	0.2864 s	0.715 s

Table 2.1: Timing comparison for Low-Pass Filter using fast convolution and time-domain convolution. Function length = 20000, Filter length = 50, Number of tests = 200.

3. Overall system

The overall system recreates the transmission of two signals over a common medium using modulation. On the sender side, the signals are low-pass filtered, modulated and added. The sum of both modulated signals is then transmitted to the receiver, where the signal is demodulated and low-pass filtered again.

For testing the overall system, the following signals are used. The first signal is a sum of cosine waves (Equation 1) and the second signal is a sawtooth wave (Equation 2).

$$f_1(t) = 3 + cos(t+1) + 2cos(3t+2) - 5cos(4t-1) + cos(13t) \tag{3.1}$$

$$f_2(t) = 3 \cdot (t \mod \pi) \tag{3.2}$$

Both signals are sampled 32 times in the interval from 0 to 2π. Each step of the transmission is shown in Figure 3.1, where the left side depicts the operations in time-domain and the right side in frequency-domain, by showing the magnitude of the first half of the Fourier coefficients.

The cut-off frequency is choosen to be 7 times the base frequency of the cosine wave. As seen in the second row, the low-pass filter removes the high frequency components. Furthermore the sampling rate is increased after the low-pass filter, because modulation would not be possible without violating the sampling theorem. Then the signals are modulated, so that their frequency bands do not overlap. After adding, demodulation and low-pass filtering, the reconstructed signals are nearly equal to the original, low-passed signals.

An alternative approach is to use quadrature modulation. This is shown in Figure 3.2, where the signals are modulated with the same modulation frequency, but with a cosine and a sine wave, respectively.

When a overlap of the frequency bands is introduced, the signals cannot be reconstructed properly. This is shown in Figure 3.3, where the signals contain noise of the other signal.

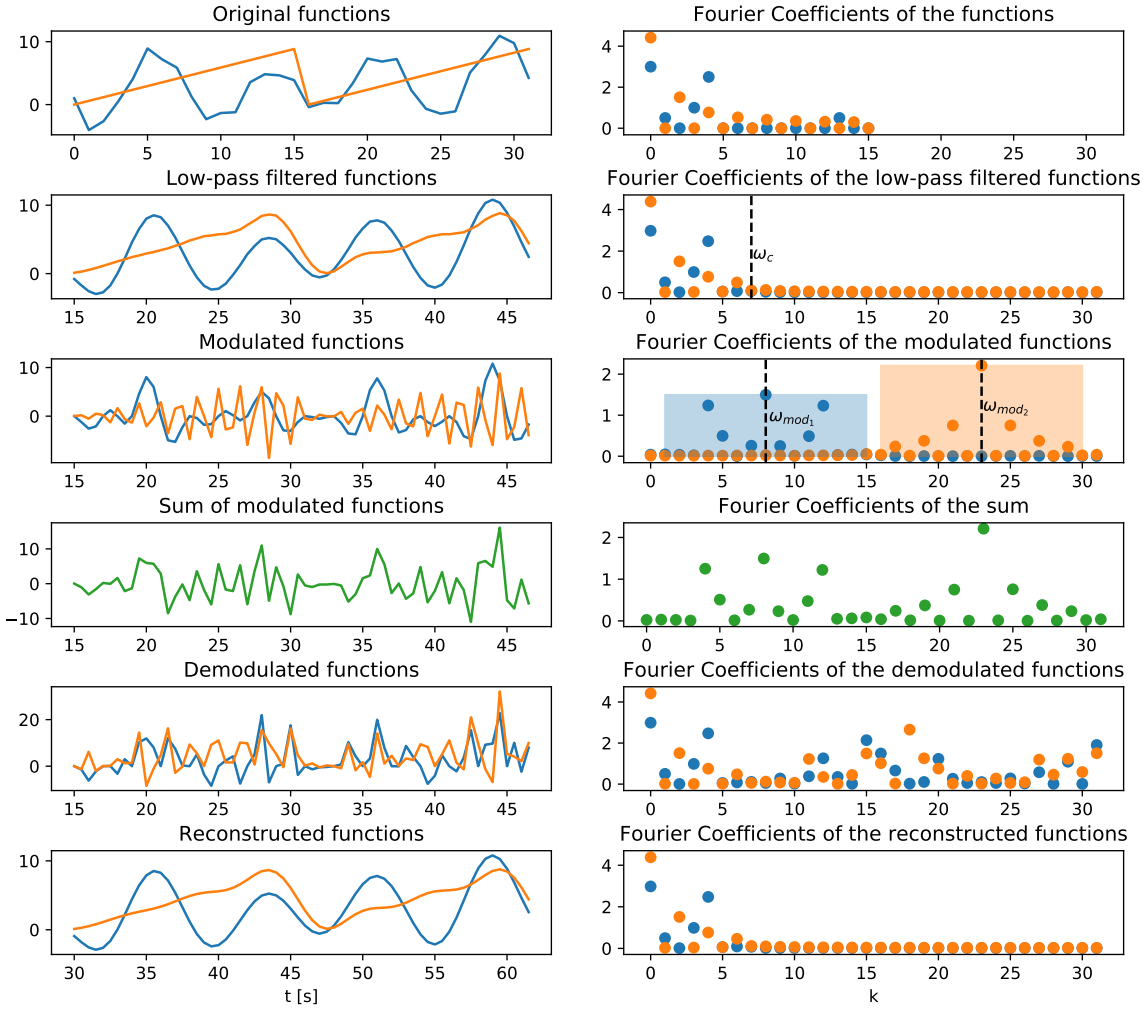

Figure 3.1: Overall sytsem.

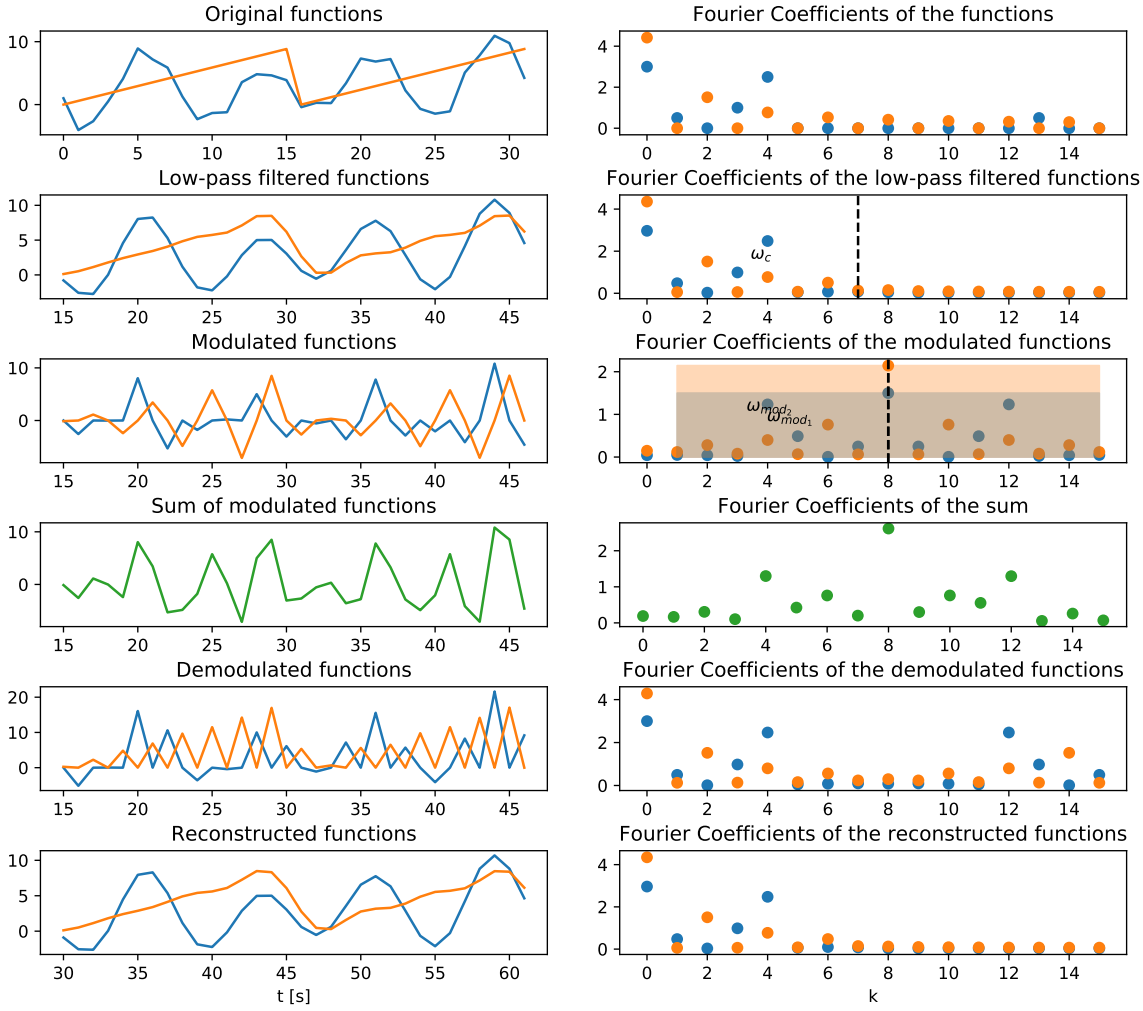

Figure 3.2: Overall sytsem with quadrature modulation.

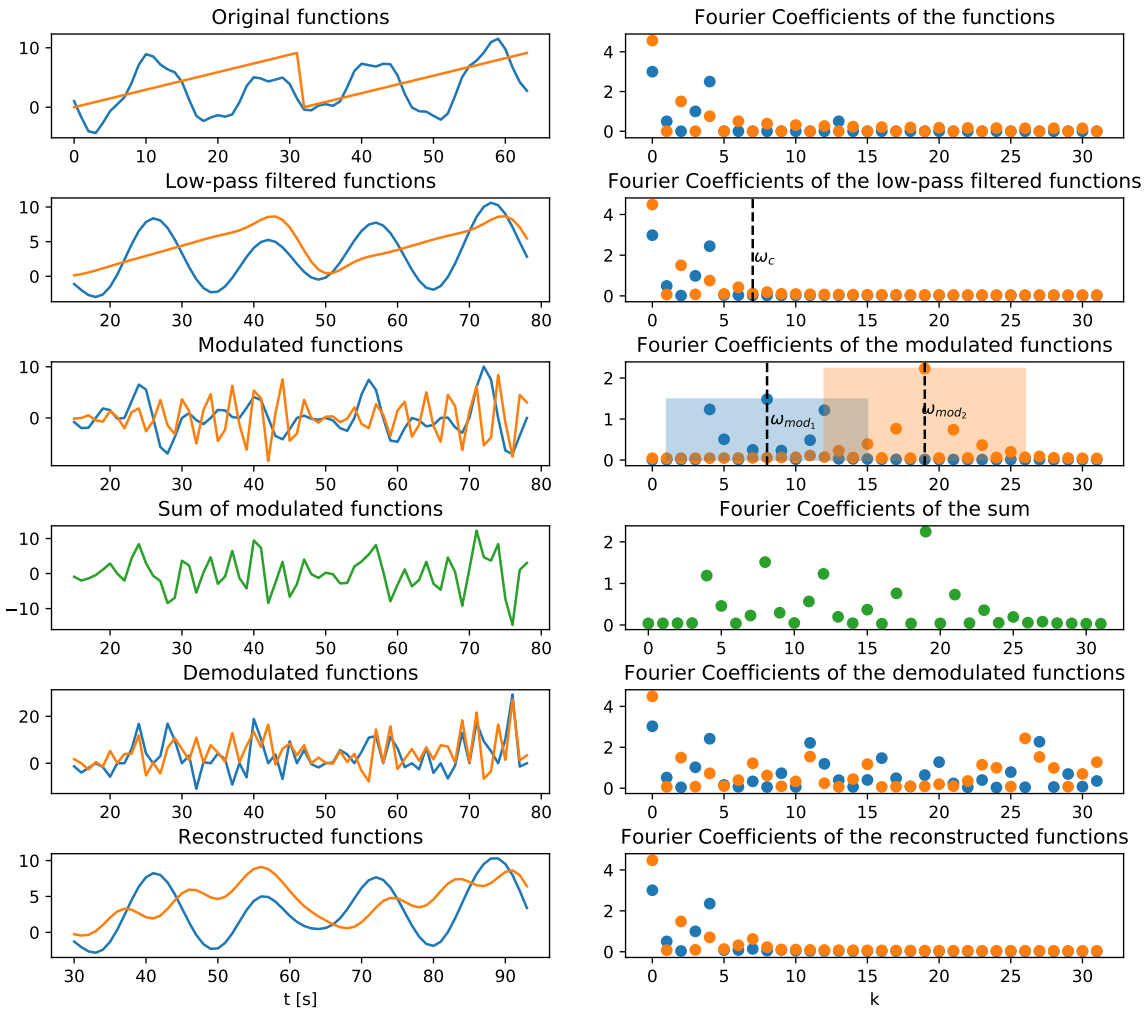

Figure 3.3: Overall sytsem with overlapping frequency bands.