Applications of digital processors

SPECTRAL ANALYSIS

on signal processors

Author: Grzegorz Szwoch

Gdańsk University of Technology, Department of Multimedia Systems

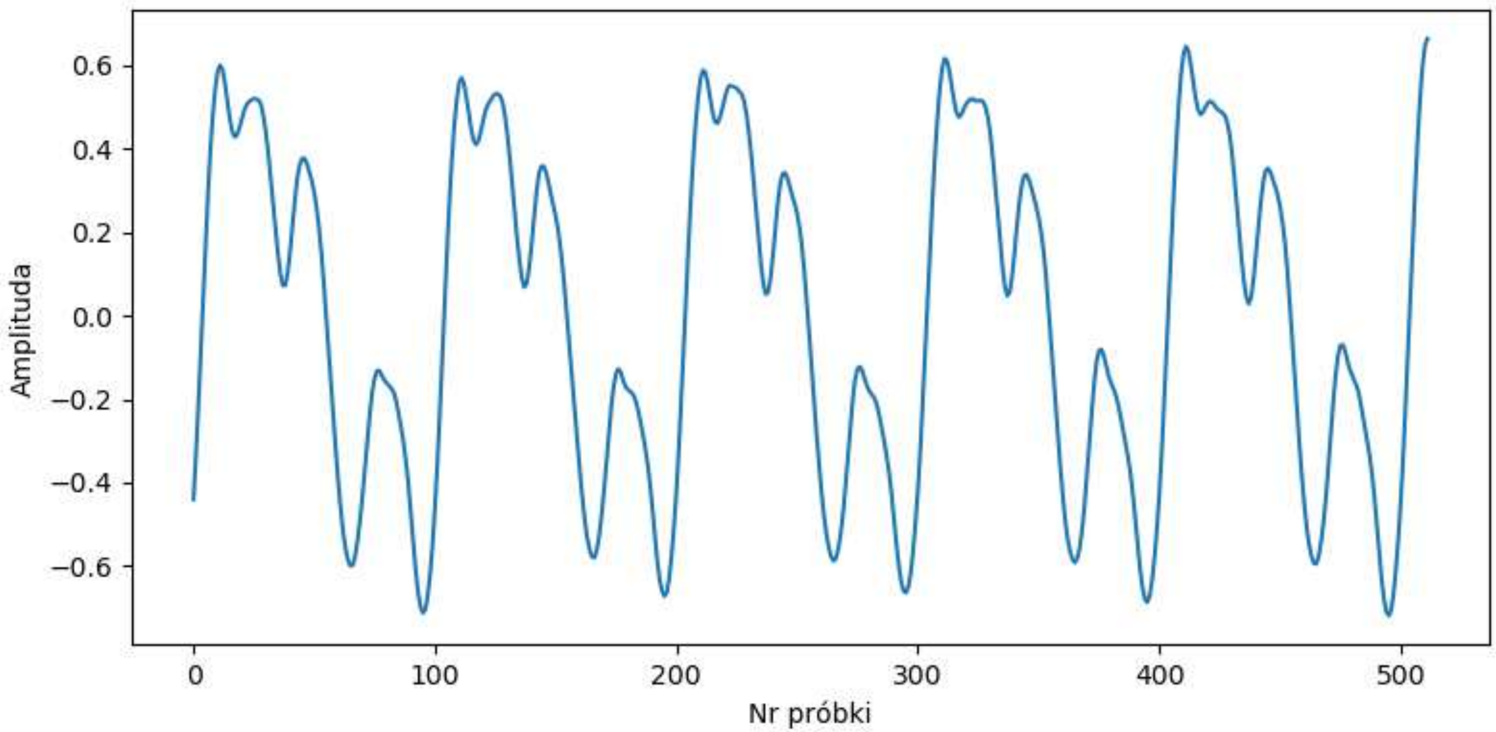
Introduction

- Samples describe values of a digital signal in the time domain.
- Many digital signal processing operations must be performed in the frequency domain. We only want to process selected frequency components, not the whole signal.
- Spectral analysis – determining spectral (frequency) components of a signal.

Example

Time plot of a clarinet sound recording.

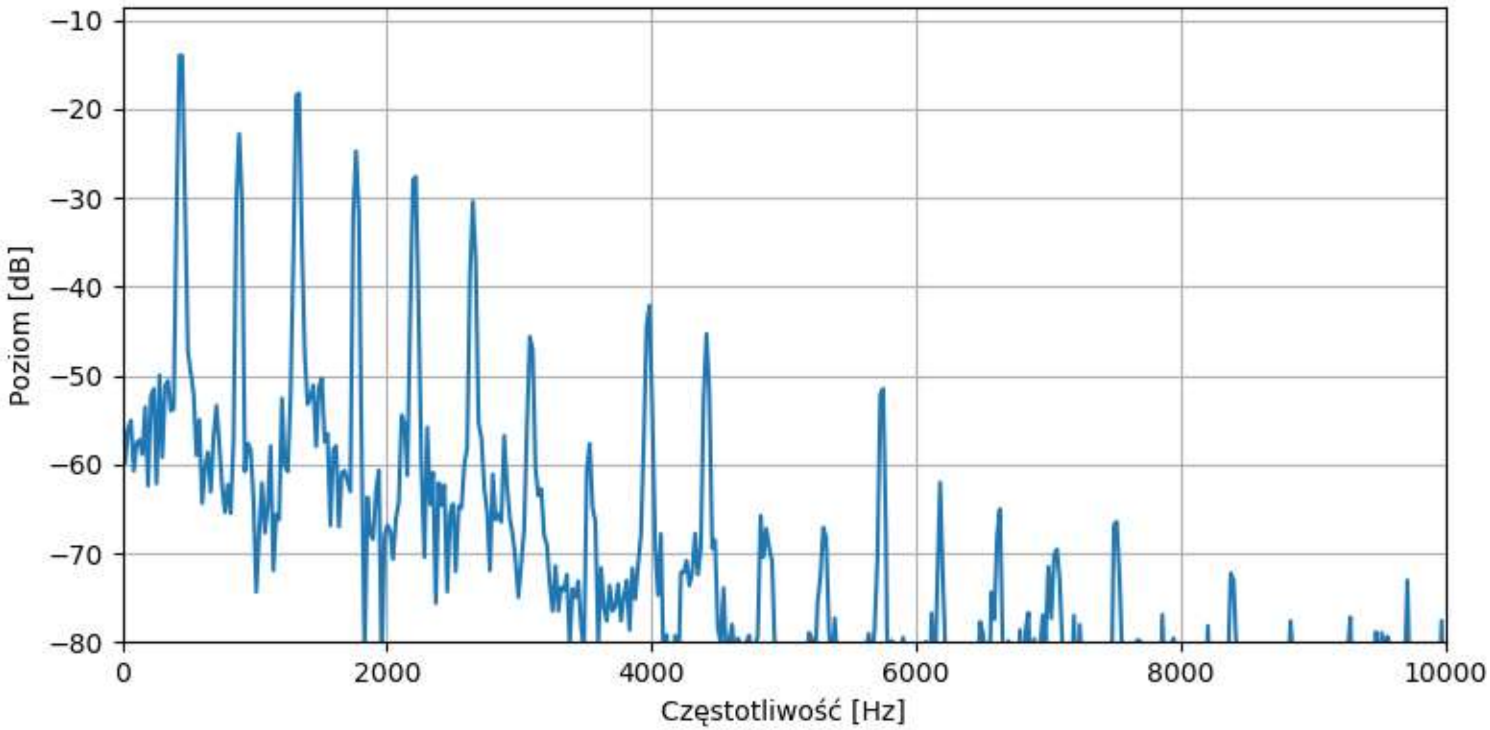
We don’t know from this plot what the signal structure is.



Example

The result of spectral analysis of the clarinet sound.

- We can see the signal structure – a sum of harmonics.
- We can determine the sound pitch – it is related to frequency of the first spectral peak.



Fourier transform

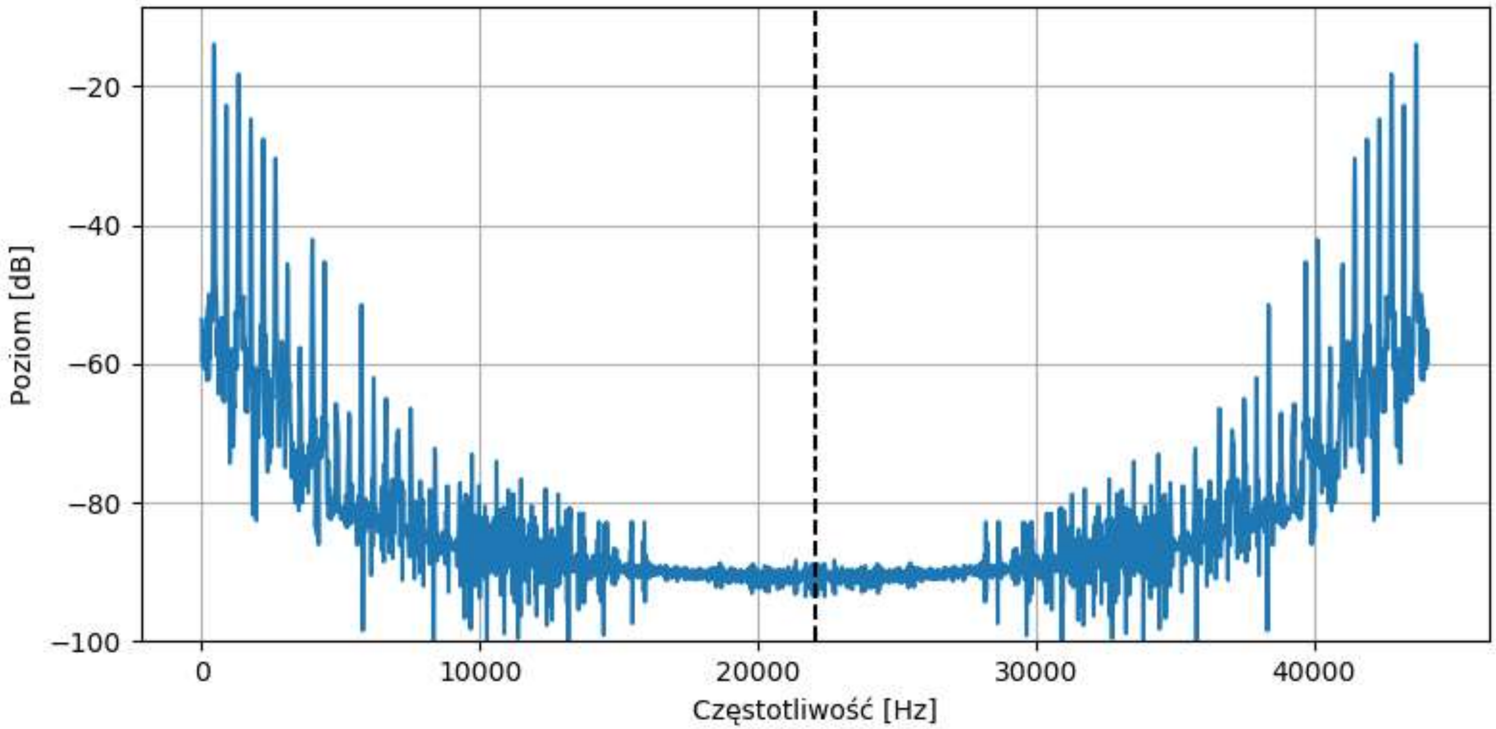
- The Fourier transform converts *N* signal samples into *N* samples of the signal spectrum.
- The result of the transform is the signal spectrum (an analogy to the light spectrum).
- The inverse Fourier transform converts spectrum samples back into signal samples.
- A signal may be transformed into the spectrum, process it and transform back into the processed signal.
  This is spectral processing.

Spectrum of a real signal

- Fourier transform works for both real and complex signals.
- The spectrum is always a set of complex numbers.
- Usually, we process real (not complex) signals.
- N samples of the spectrum of a real signal:
  - the first value: the direct component, a sum of samples
  - values 2 to $N/2$: samples of the signal spectrum
  - value at $N/2 + 1$: a Nyquist component, should be zero
  - values $N/2 + 2$ to N: a mirror reflection of the first half of the spectrum (a Hermitian symmetry).
- A spectrum obtained from N samples has $(N/2 + 1)$ unique values.

Spectrum of a real sound

We only need ($N/2 + 1$) values out of N spectral samples.



Amplitude spectrum

- Spectrum $X(f)$ contains complex values.
- Usually, we are interested in the magnitude spectrum $A(f)$, the absolute value of the complex spectrum:

$$A(f) = \left|X(f)\right|$$

- Power spectrum is the absolute value of the spectrum, squared:

$$P(f) = \left|X(f)\right|^2$$

- Often, we use a logarithmic spectrum, expressed in decibels (dB):

$$A(f) = 10\log_{10}\left|X(f)\right|$$

$$P(f) = 10\log_{10}\left|X(f)\right|^2 = 20\log_{10}\left|X(f)\right|$$

Magnitude spectrum

- To obtain the amplitude of spectral components, we must divide the absolute value of the spectrum by the number of samples *N*, and multiply by two, as the signal energy is divided into two mirrored halves.
- Amplitude of a spectral component at index *n*:

$$A[n] = \frac{2}{N}\left|X[n]\right|$$

- The first (DC) and the Nyquist component do not have a pair, they should not be multiplied by two.
- The direct component divided by *N* equals to the mean of the signal within the analyzed section.

Frequencies of the signal samples

- From *N* signal samples, we obtain *N* spectral samples.
- The spectrum covers the frequency range $[0, f_S]$.
- The *n*-th spectral component is at the frequency:

$$f[n] = n\frac{f_S}{N}$$

- Spacing between spectral values is equal to the sampling frequency divided by the number of samples.
- This is a frequency resolution of spectral analysis:

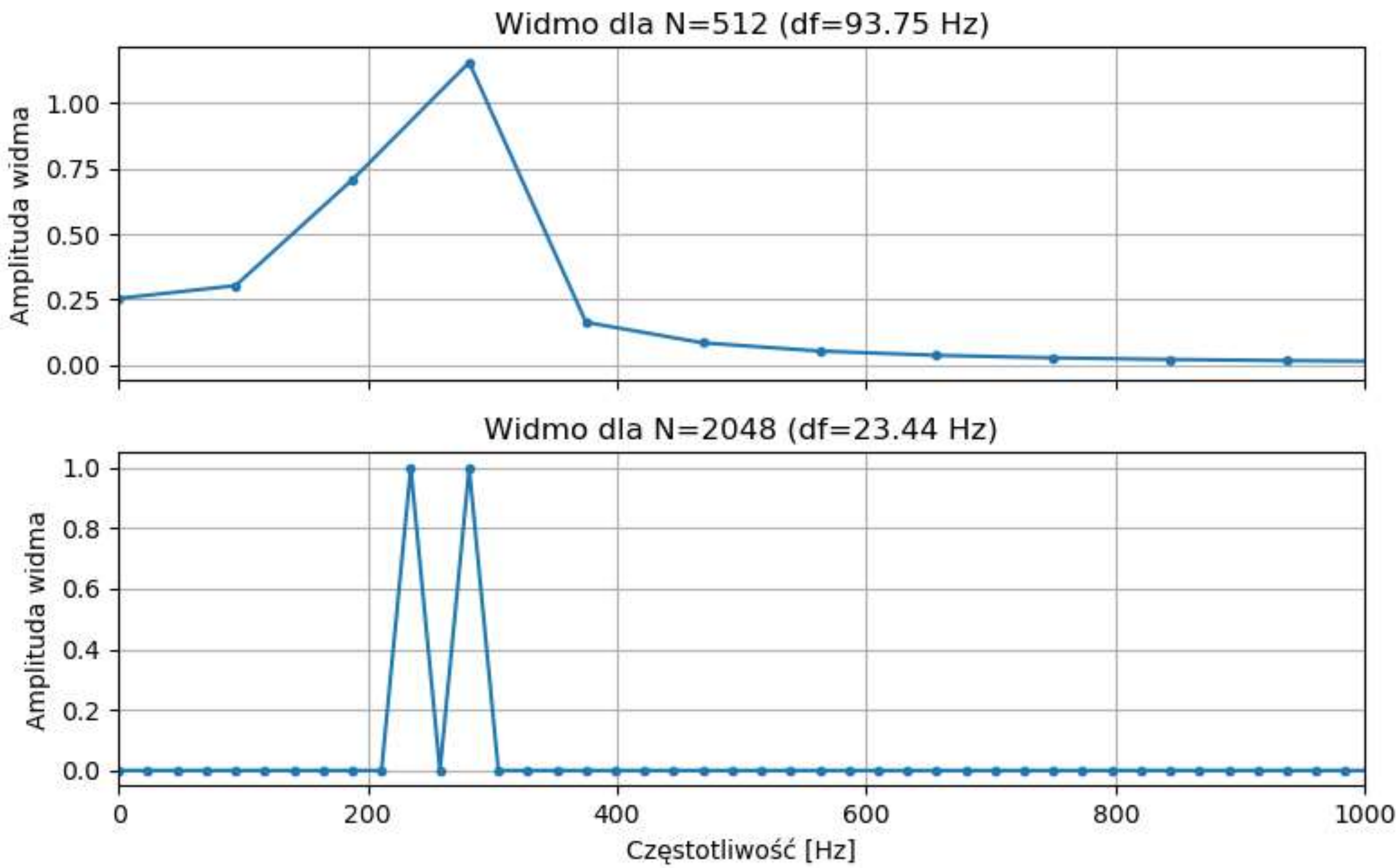
$$df = \frac{f_S}{N}$$

Frequency resolution

- What does frequency resolution mean ($df = f_S / N$):
  - two spectral components cannot be distinguished if they are spaced by less than *df* (they fall into the same spectral bin),
  - inaccuracy (error) of determining the frequency of a spectral component is max. $\pm df/2$.
- Larger number of samples (N) increases the resolution.
- Resolution may be improved by zero padding the signal before the transform. We don't get more data (the values are interpolated), but we obtain better resolution.

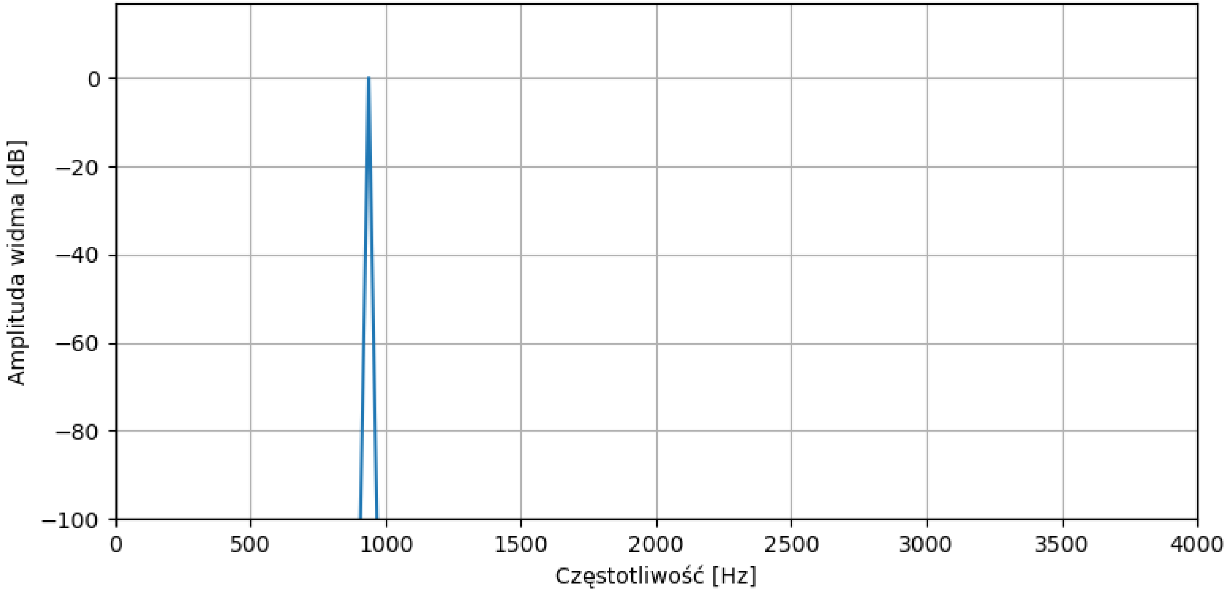
Frequency resolution

Example: sum of two sines $f1$ = 234.375 and f2 = 281.25 Hz



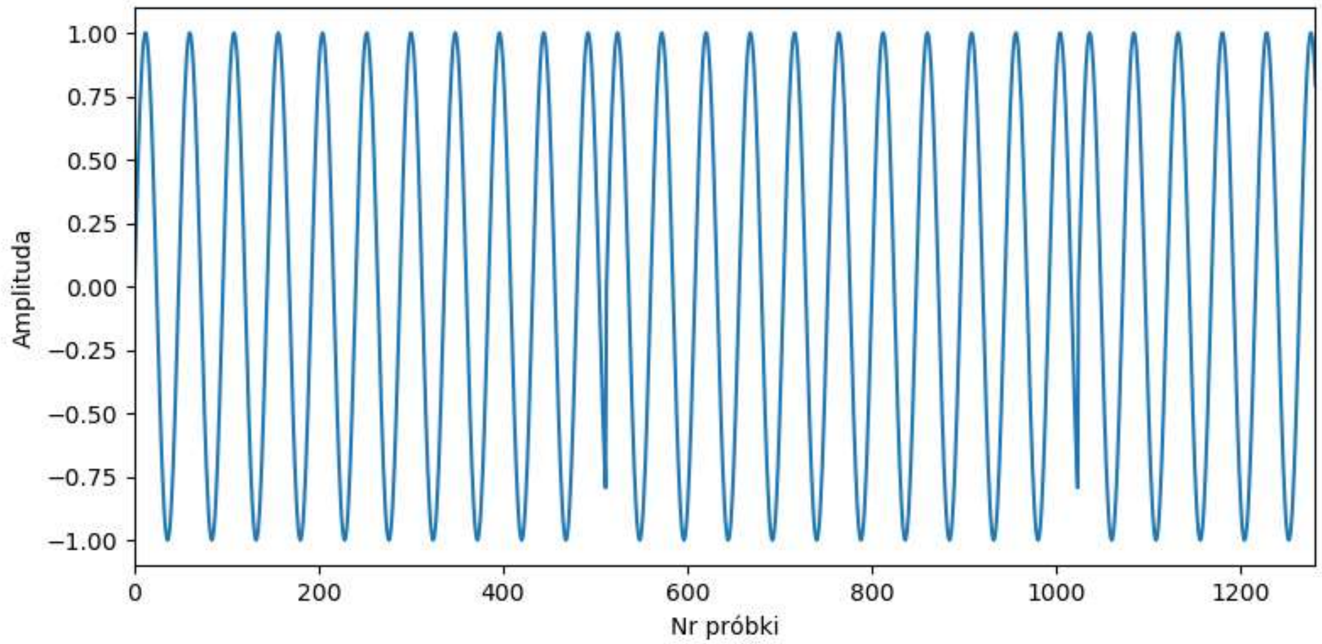
Periodicity of the signal

- Fourier transform assumes that the signal is periodic.
- A signal section used for the transform is assumed to be the signal period (or its multiple).
- Example for a signal in which the length of the window is a multiple of the signal period (f = 937.5 Hz; N = 512):



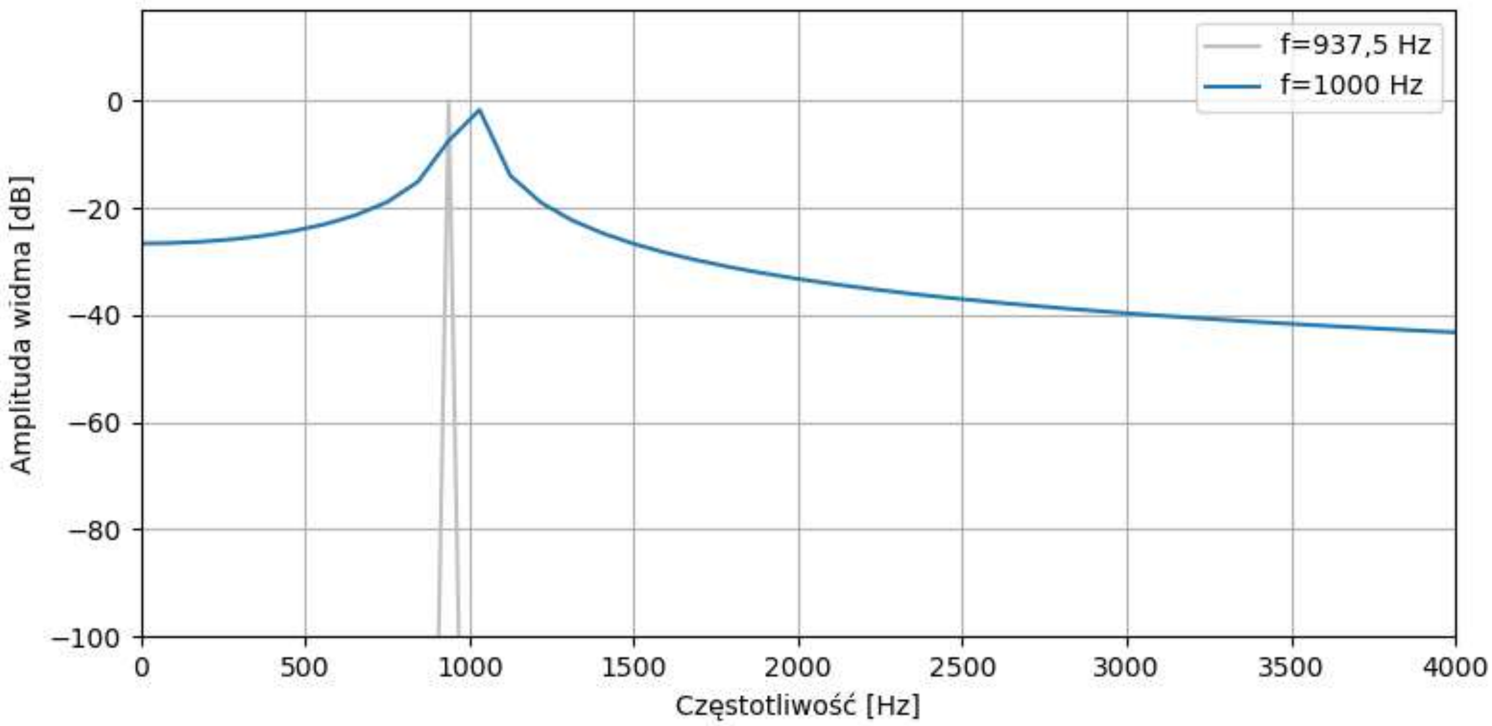
Periodicity of the signal

- What happens if the analysis window length is not a multiple of the signal period?
- The result is a transform of a signal that is a "looped" analysis window.
- Example for f = 1000 Hz, N = 512:



Spectral leakage

How does the spectrum of such a signal look like?

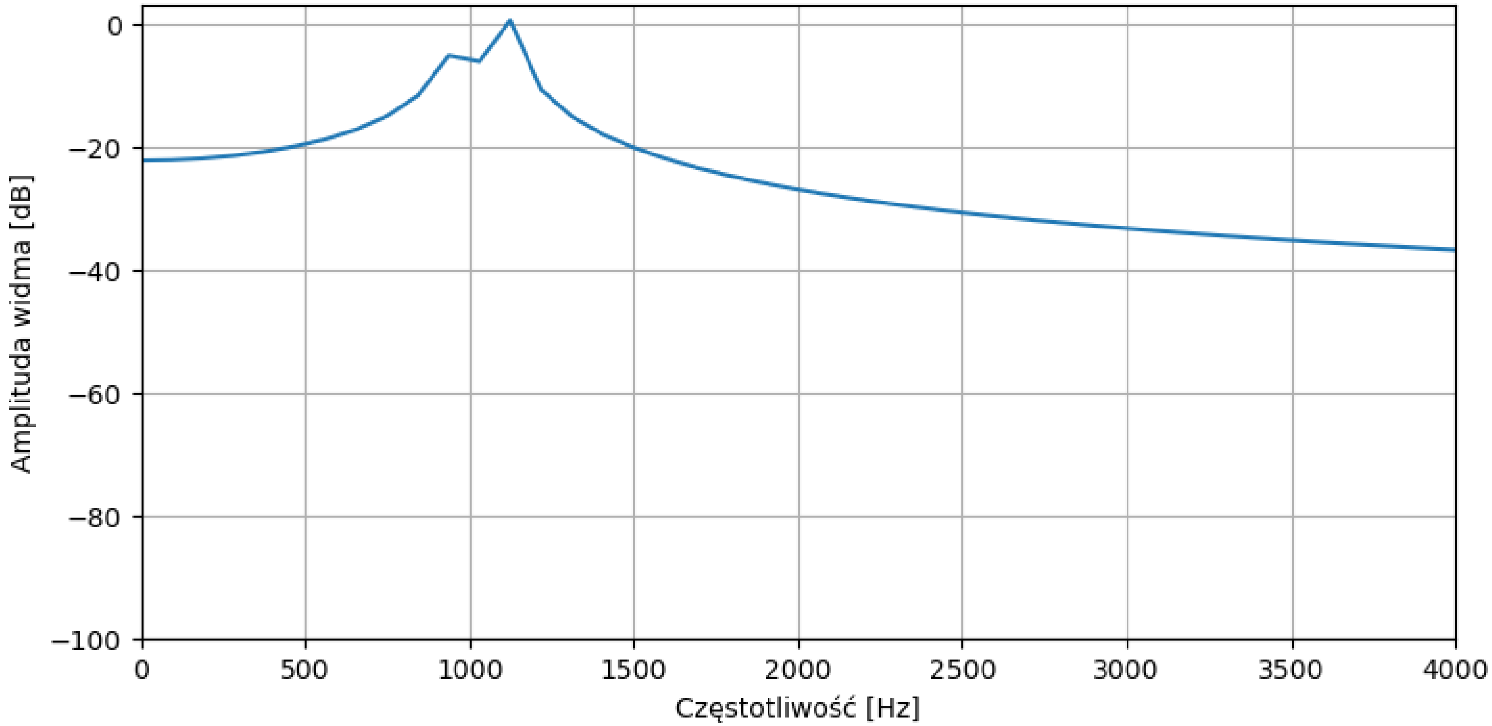


This is spectral leakage – the spectral energy "leaks" to the adjacent bins.

Spectral leakage

Example: sum of sines 1000 Hz & 1100 Hz (N = 512).

Spectral leakage hides the shape of individual peaks.

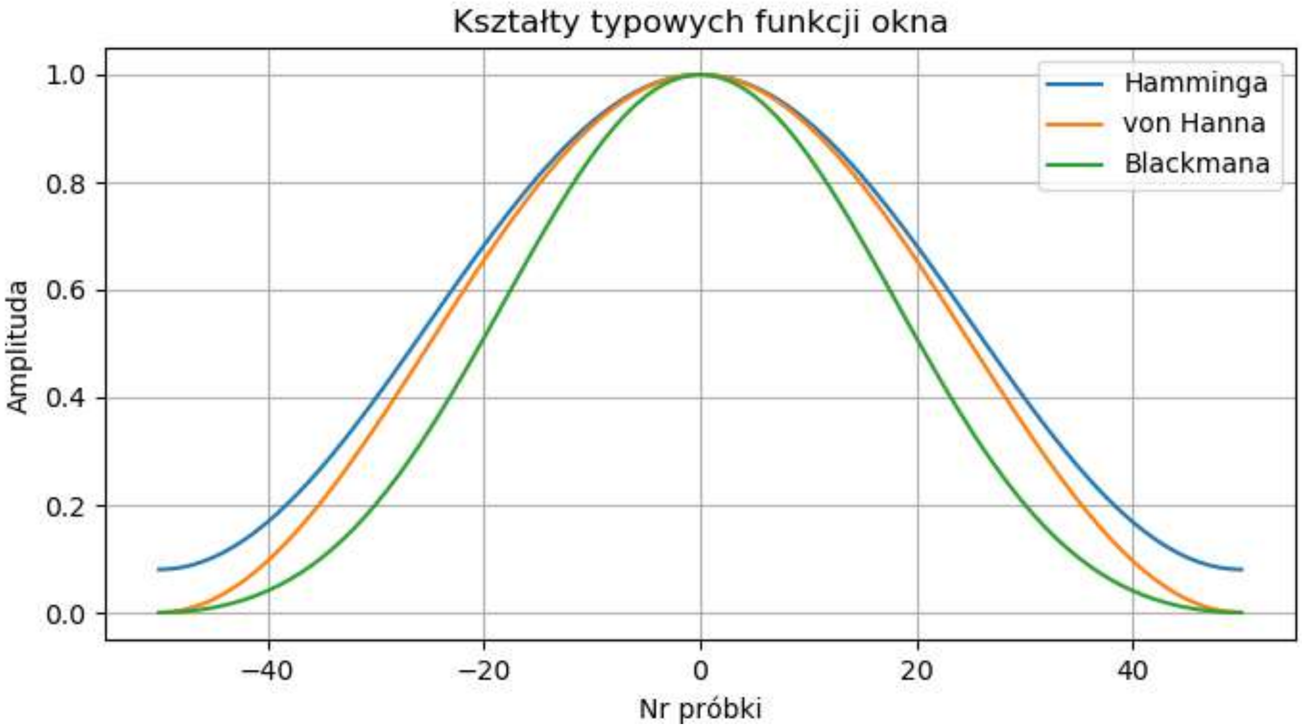


Analysis windows

- Spectral leakage is a result of discontinuity at the analysis window when it is looped.
- Leakage may be suppressed if the signal is multiplied by a window function before the transform.
- The window function suppresses the samples at the edges.
- The result of applying a window:
  - it reduces energy leaks to the adjacent spectral bins,
  - but it also widens the peaks – the leakage is confined to a narrower range.

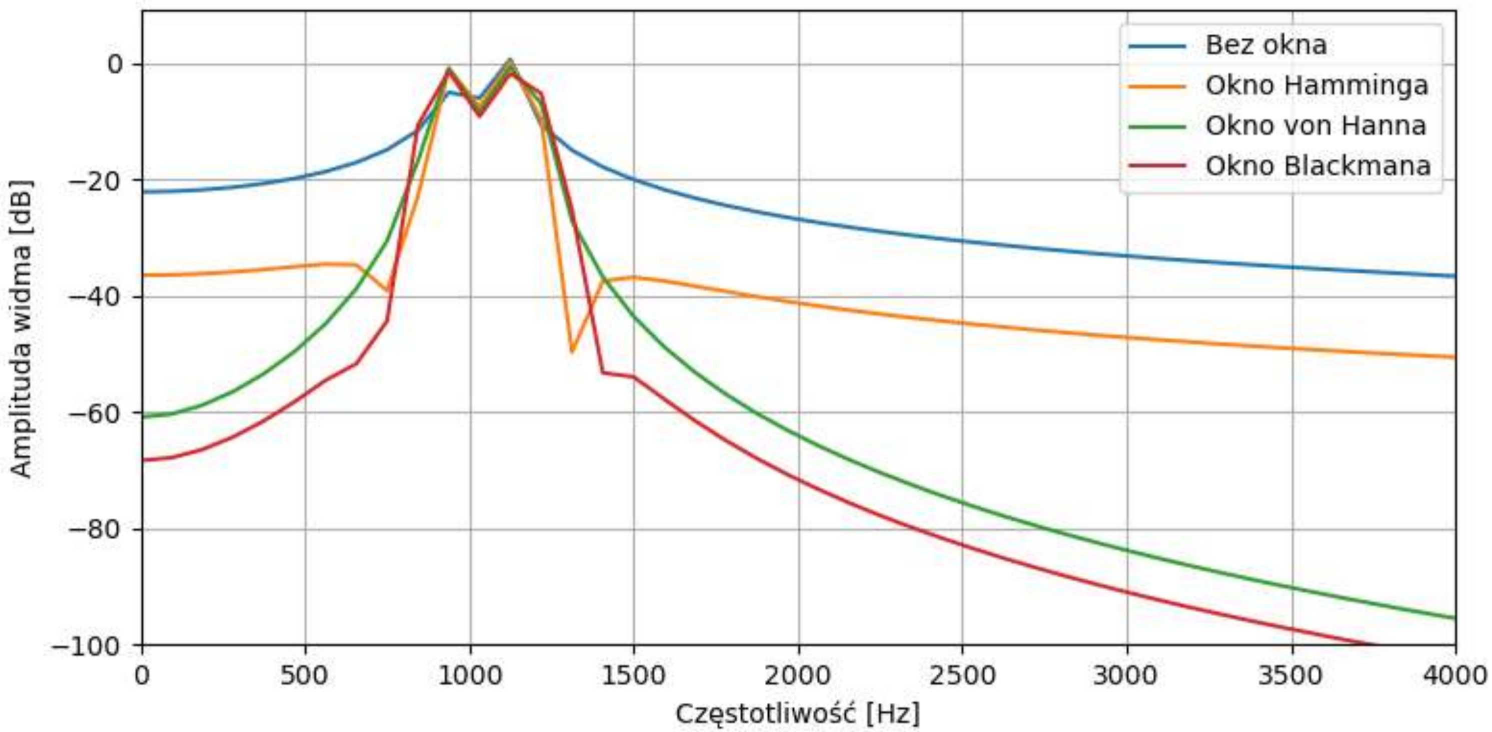
Analysis windows

The windows that are most frequently used for spectral analysis:



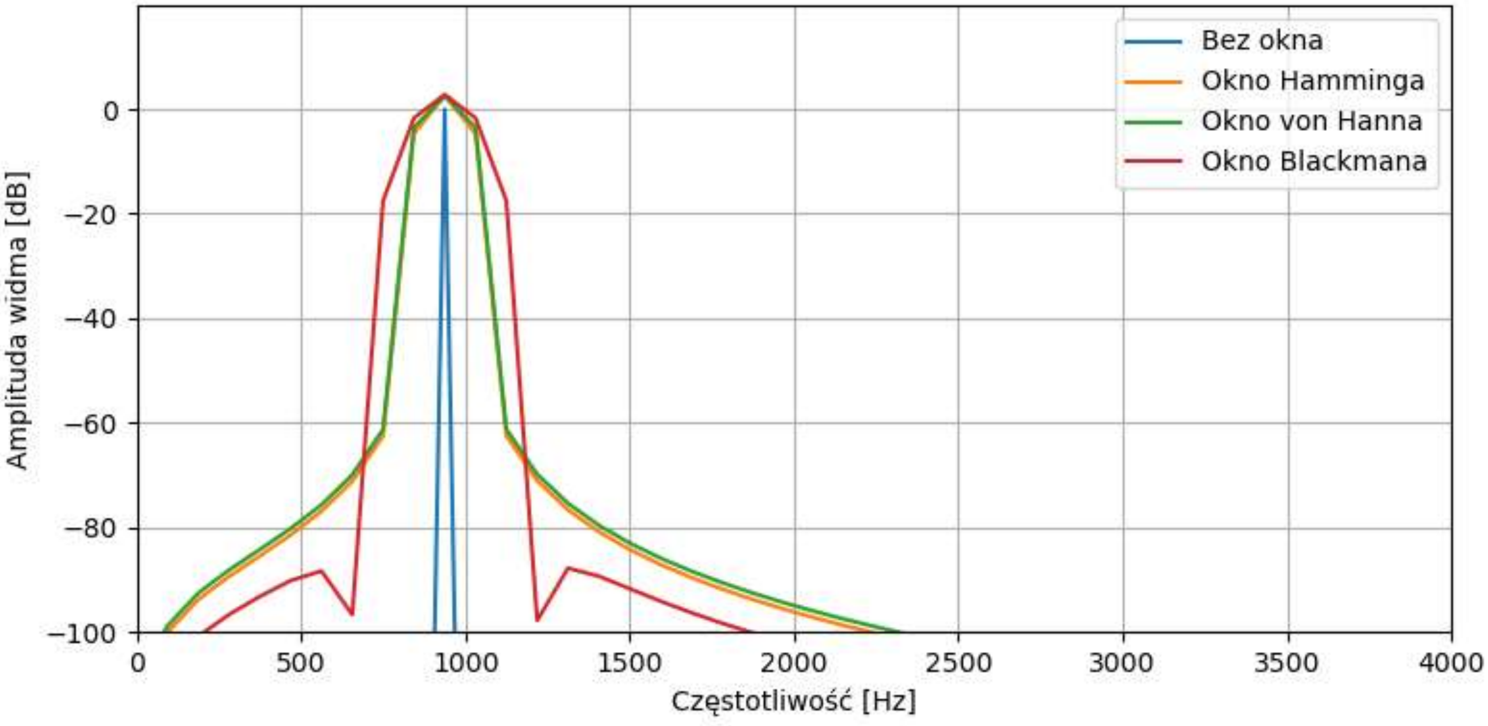
Effect of a window function

Window applied to the analysis in the case of large leakage – the result is improved (the leakage diminishes).



Effect of a window function

Effect of applying a window when there was no leakage
- widening of the peak is visible.



Remarks about windows

- There is no “best” window. Hamming and von Hann are the most frequently used ones.
- Blackman window is useful when we need to suppress the leakage, at the cost of widening the peaks.
- We use the window function when we analyze the spectrum, e.g. we search for the maxima.
- We do not use the window if we process the signal in frequency domain and then we get back to time domain.
- Amplitude normalization when window *w* is used:

$$A[n] = \frac{2}{\sum w_i} \left| X[n] \right|$$

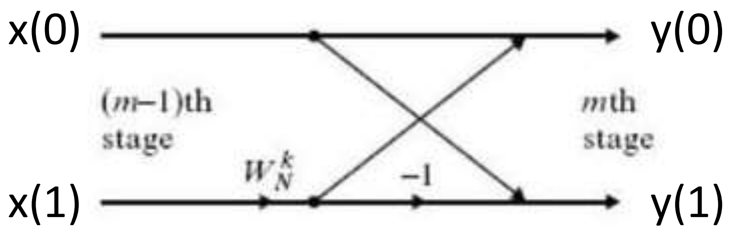
Computing a Fourier transform

There are two ways to compute a Fourier transform

1. From the definition of discrete Fourier transform (DFT):
   - works for all signals,
   - requires many multiplication and addition operations.
2. With Cooley-Tukey algorithm, also called fast Fourier transform (FFT):
   - reduced number of operations,
   - limitations related to the window length,
   - used e.g. in digital signal processors.

FFT

Fourier transform of two samples:



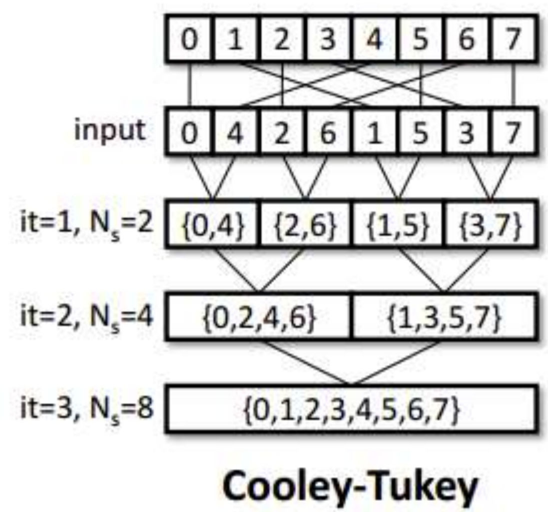
$$y(0) = x(0) + x(1) \cdot W_n^k$$
$$y(1) = x(0) - x(1) \cdot W_n^k$$

- We need to perform: one multiplication, one addition and one subtraction.
- This is a butterfly structure.
- It is a base element of a radix-2 FFT algorithm.
- FFT of length 2^N may be computed in *N* stages, using the butterfly structure.

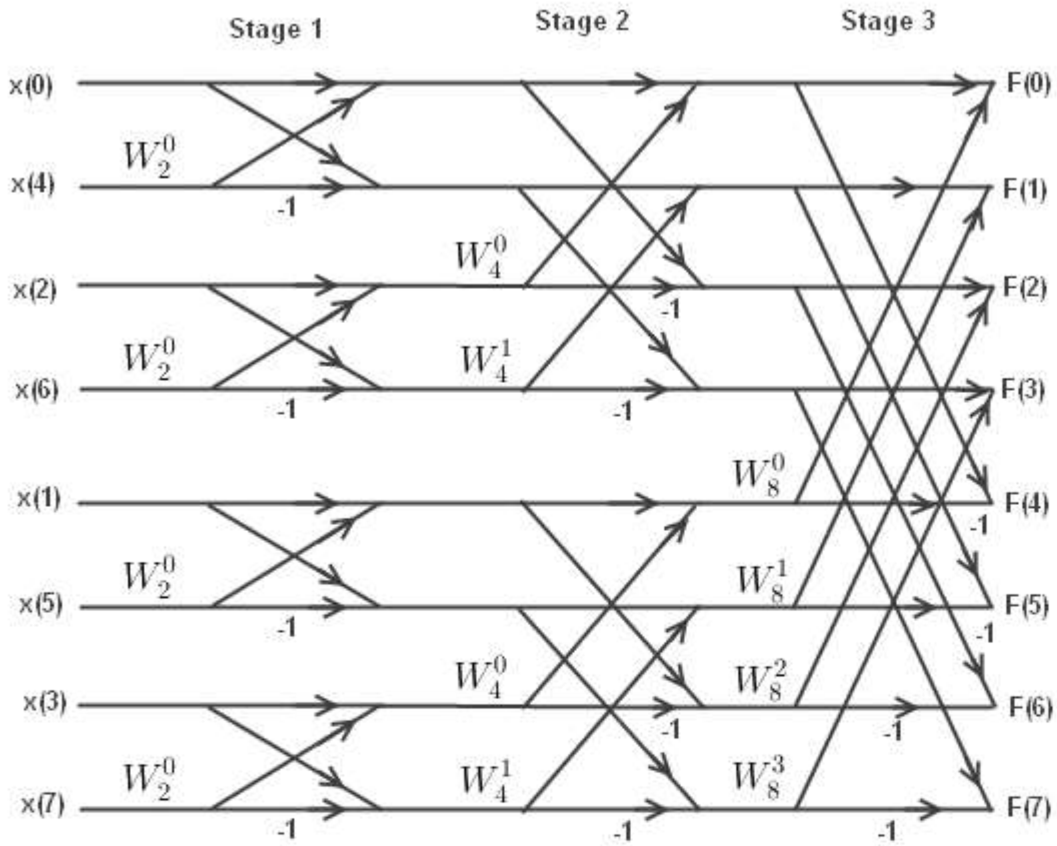
FFT

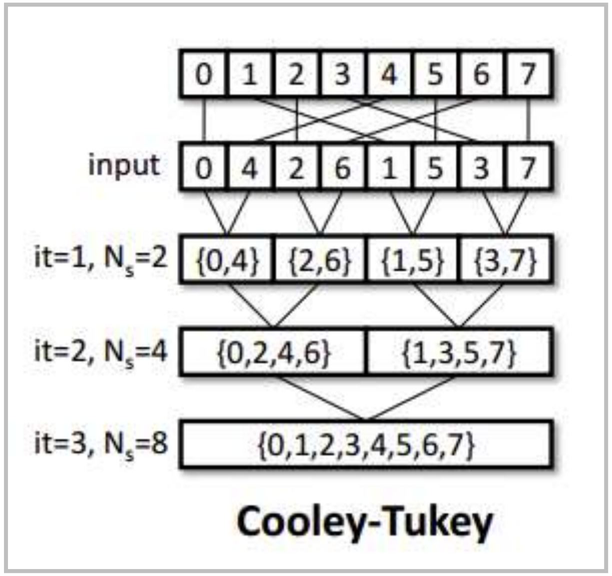
A radix-2 Cooley-Tukey FFT algorithm:

- The length of the signal must be a power of two: 2^N.
- We divide the signal into two parts, assigning samples to each part alternatively.
- We repeat this procedure until we get length 2 sequences.
- We compute the “butterflies”.
- Then we compose the results into new butterflies.
- This is repeated until the whole transform is computed.



FFT – example for N = 8 = 2^3





FFT - twiddle factors

- The coefficients W have form:

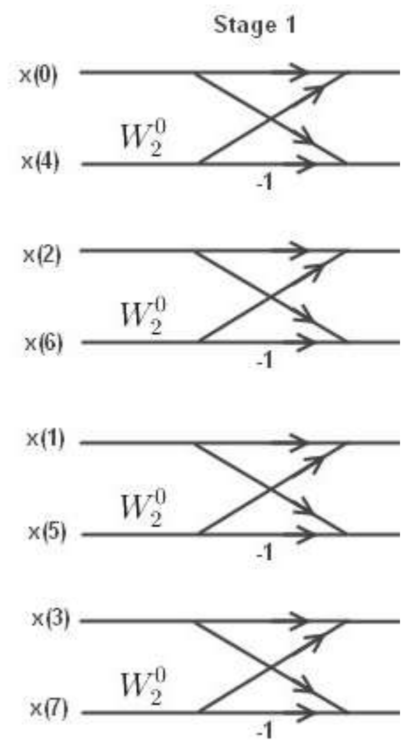
$$W_N^k = e^{-j2\pi k/N} = \cos\left(\frac{2\pi k}{N}\right) - j\sin\left(\frac{2\pi k}{N}\right)$$

- They are called twiddle factors.
- N is the transform length at a given stage.
- At the i-th stage we need 2^{i-1} coefficients.
- Twiddle factors are always the same for a defined transform length. Therefore, they are usually precomputed and kept in a table in memory.

Bit reversal

- Before the transform, the samples must be set in the correct order for the first stage.
- It can be done by reversing the bit order of the indices in the binary notation (bit reversal)

0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7



Transform length

- A classic radix-2 FFT algorithm requires that the transform length is a power of two. It is recommended to use this convention.
- The most frequently used transform lengths:
  512, 1024, 2048.
- If we don't have enough samples, we can pad them with zeros to the nearest power of two. We shouldn't do this unless there are no more samples available.

Modern FFT algorithms

FFT libraries used in practice (e.g. FFTW):

- Implementations of “butterflies” for various radices, e.g. 2, 3, 5, 7, 11, 13, 17, 19 (low prime numbers).
- The required transform length is decomposed into prime factors, e.g.: $2016 = 2^5 \cdot 3^2 \cdot 7$.
- The signal is divided into sections and FFT of radix-2/3/7 are computed, then the results are merged (a split radix algorithm).
- If one of the parts has a length which is a large prime number, a slower DFT is computed. This should be avoided.

Comparison DFT and FFT efficiency

Taken from: Mark McKeown, FFT Implementation on the TMS320VC5505, TMS320C5505, and TMS320C5515 DSPs (SPRABB6A)

FFT Length	Direct DFT Computation		Radix-2 FFT	
	Complex Multiplications	Complex Additions	Complex Multiplications	Complex Additions
128	16,384	16,256	448	896
256	65,536	65,280	1,024	2,048
512	262,144	264,632	2,304	4,608
1024	1,048,576	1,047,552	5,120	10,240

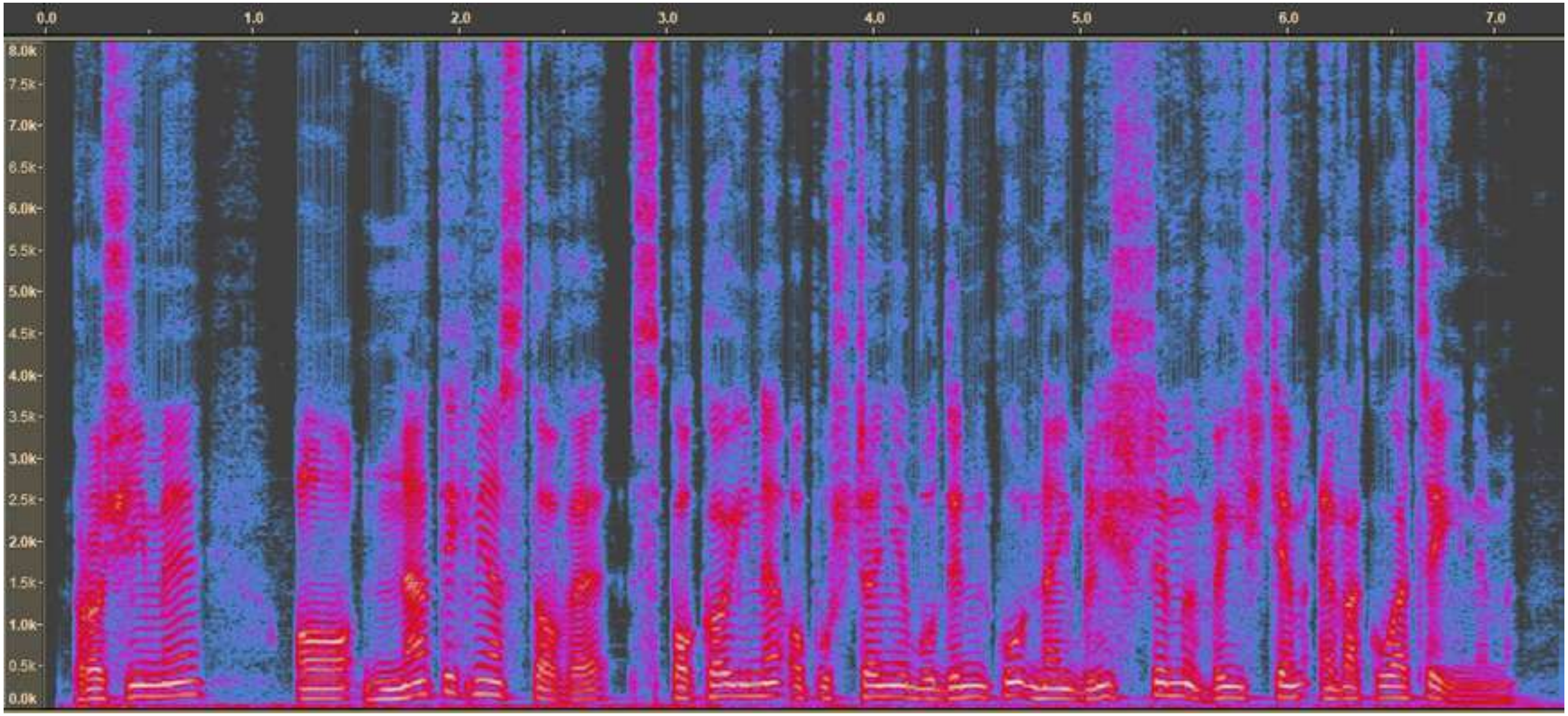
For radix-2 N=1024: about five thousands of complex multiplication operations, compared with over a million multiplications for the DFT (200× more).

Analysis of a continuous signal

- A Fourier transform works on blocks of samples.
- If we analyze a continuous signal, we need to divide it into blocks (windows). For each block, we compute FFT.
- This is called a short-term Fourier transform (STFT).
- Each spectrum “averages” the signal within the window over time.
- We lose short-term events inside the block.

Analysis of a continuous signal - STFT

STFT result in a form of a spectrogram: time (horizontal) vs. frequency (vertical) vs. spectral level (color).



Temporal resolution of STFT

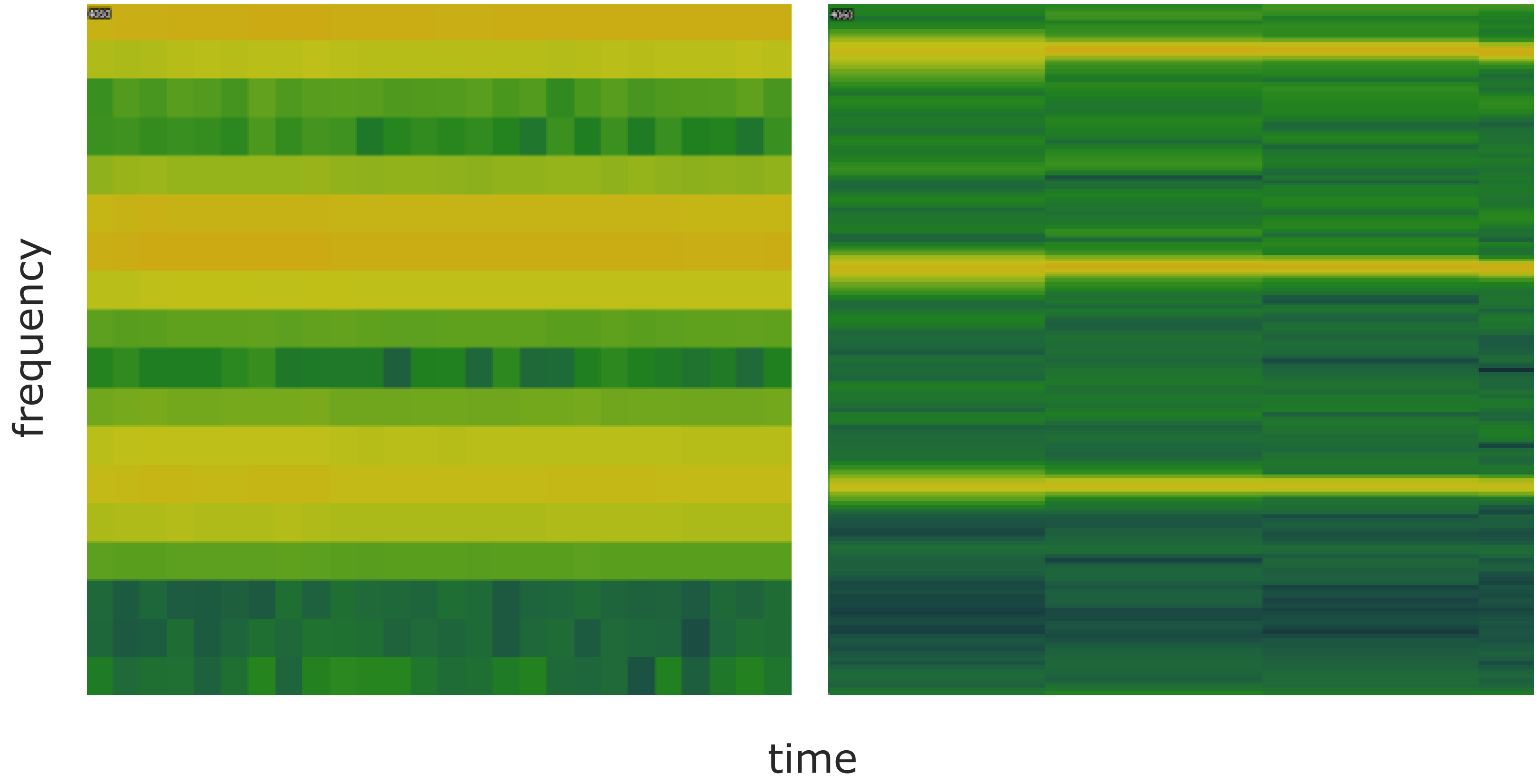
- Temporal resolution depends on the window length:

$$dt = \frac{N}{f_S} = \frac{1}{df}$$

- It is a reciprocal of frequency resolution.
- Interpretation: a minimum time difference between two events in the signal that can be distinguished in STFT.
- We can’t have good temporal resolution and good frequency resolution of STFT at the same time.

Temporal resolution of STFT

Comparison of windows of length 512 and 4096 samples.



Overlapping

- Overlapping is achieved by moving the analysis window by less than the window length (some samples are used more than once).
- We use overlapping to:
  - increase the effective temporal resolution,
  - reduce the effect of applying a window function.
- Usually:
  - for Hamming and von Hann window, we shift the window by ½ of its length,
  - for Blackman window, we shift it by ¼ of its length.

FFT on digital signal processors

- Architecture and commands of digital signal processors allow for fast FFT execution.
- Some DSPs have coprocessors for FFT computation.
- The DSP maker usually provides optimized FFT procedures in Assembler, we should use them.
- Often, they are radix-2 implementations.
- We can write our own FFT implementation, bit it’s not easy to obtain a faster algorithm than the one already tested.

FFT on C5535 DSP

- The C5535 fixed point DSP (used in the course project) has a coprocessor for FFT computations (HWAFFT).
- A hardware FFT implementation for length: 8, 16, 32, 64, 128, 512, 1024.
- Uses complex signal values. If we process a real signal, we must insert zeros for the imaginary parts.
- A special procedure for bit reversal exists.
- The twiddle factors are precomputed and kept in memory.
- Two FFT stages can be computed in a single run.

FFT on C5535 DSP

- Functions for FFT and IFFT are available from C code.
- It’s easier to use functions from DSPLIB.
  Documentation: SPRU422J

Functions	Description
void cfft (DATA *x, ushort nx, type)	Radix-2 complex forward FFT – MACRO
void cfft32 (LDATA *x, ushort nx, type);	32-bit forward complex FFT
void cifft (DATA *x, ushort nx, type)	Radix-2 complex inverse FFT – MACRO
void cifft32 (LDATA *x, ushort nx, type);	32-bit inverse complex FFT
void cbrev (DATA *x, DATA *r, ushort n)	Complex bit-reverse function
void cbrev32 (LDATA *a, LDATA *r, ushort)	32-bit complex bit reverse
void rfft (DATA *x, ushort nx, type)	Radix-2 real forward FFT – MACRO
void rifft (DATA *x, ushort nx, type)	Radix-2 real inverse FFT – MACRO
void rfft32 (LDATA *x, ushort nx, type)	Forward 32-bit Real FFT (in-place)
void rifft32 (LDATA *x, ushort nx, type)	Inverse 32-bit Real FFT (in-place)

Complex spectrum representation

- Spectral values are complex numbers.
- Real and imaginary parts are written separately, one after another:
  Re(0), Im(0), Re(1), Im(1), Re(2), Im(2), …
- Each part is represented as Q15 or Q31.
- Function *cfft* requires a complex signal. If we have a real signal, we must insert zeros in between real values.
- For IFFT, we must write a complex spectrum as above.

FFT of a real signal

Functions *rfft* and *irfft* use a trick:

- they treat a real signal as a complex one,
- they compute FFT of length $N/2$,
- then they transform the result to obtain the correct one.

Therefore, we can compute FFT of maximum length 2048.

Details: Robert Matusiak, Implementing Fast Fourier Transform Algorithms of Real-Valued Sequences With the TMS320 DSP Platform (SPRA291)

WARNING: all FFT functions in DSPLIB operate in place, i.e., they overwrite the input buffer!

FFT of a real signal

- Spectrum of a real signal is symmetric.
- From N signal values, the *rfft* function computes $N/2$ complex values of the spectrum, represented with N numbers (real part, imaginary part).
- The first two values are real: the direct component and the Nyquist component.
- Spectrum representation:
  Re(0), Re(N/2), Re(1), Im(1), Re(2), Im(2), …

Range overflow

- On fixed point DSPs, there is a risk of range overflow when FFT stages are computed.
- DSPLIB functions work in two modes.
- SCALE mode:
  - results after each stage are divided by 2,
  - no overflow if the input values < 1.
- NOSCALE mode:
  - no scaling,
  - no overflow only if the input values < $(1/N)$, for FFT of length 2^N.

FFT of a real signal

- For a real signal, *rfft* function:

```
void rfft (DATA *x, ushort nx, type);
```

- Arguments:
  - *x* – pointer to a sample buffer (will be overwritten!), type DATA (= *short*).
  - *nx* – buffer length (the number of samples).
  - *type* – scaling mode, we use *SCALE*.

```
rfft(bufor, 2048, SCALE);
```

Project configuration for FFT

FFT functions from DSP library have the following requirements (example for N = 2048).

- Memory configuration in *.cmd* file:

```
.fftcode        >  SARAM0
.data:twiddle   >  SARAM1, align(2048)
.input          >  DARAM0, align(4)
```

- Buffer declaration in C code:

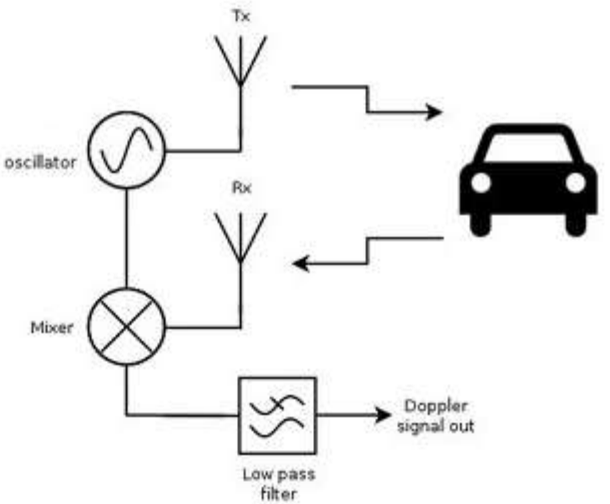
```
#define N 2048
#pragma DATA_SECTION (bufor_fft, ".input")
DATA bufor_fft[N];
```

- The name ".input" is an example, any name can be used.

A practical project - a Doppler radar

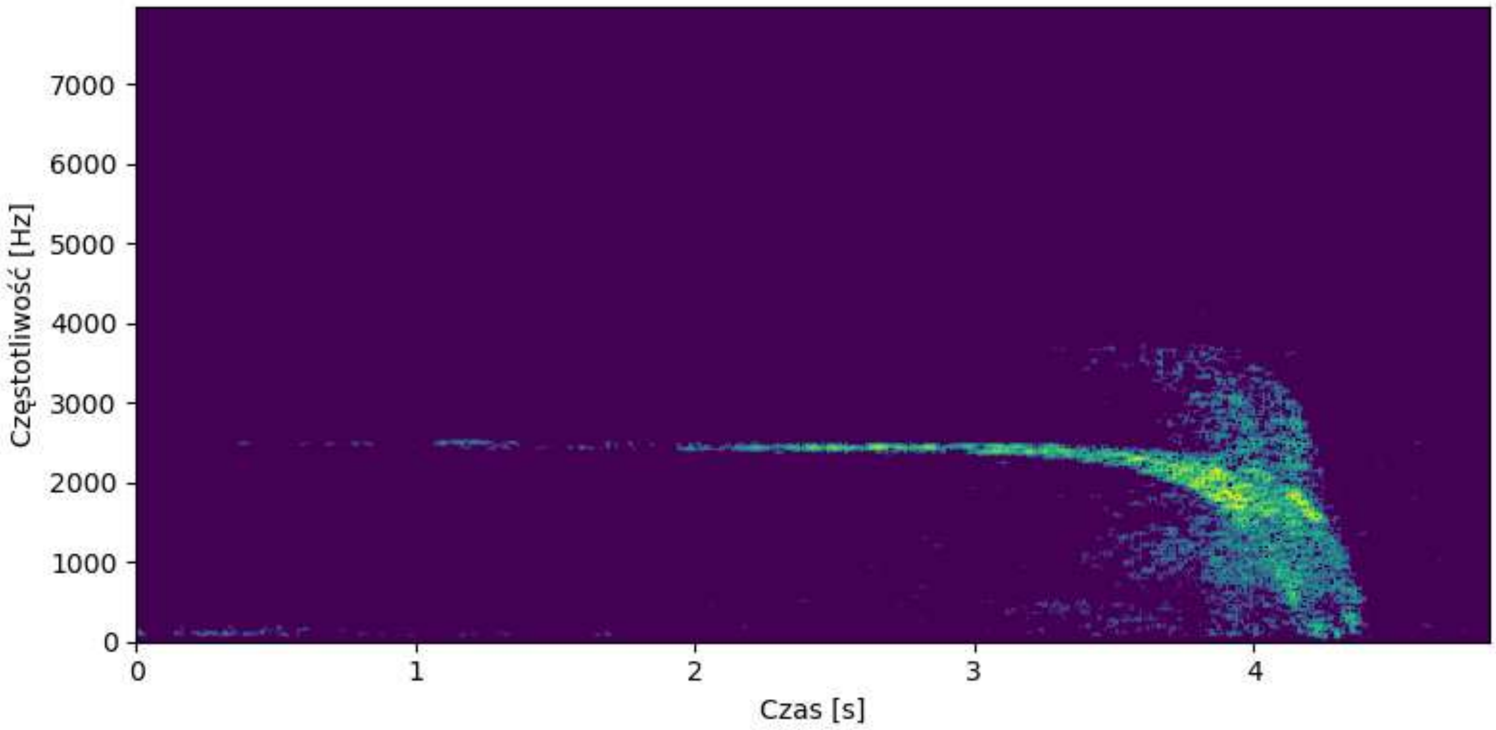
We use a DSP to analyze the signal from a microwave Doppler radar sensor.

- The emitter send an electromagnetic wave
  – a sine of frequency 24.125 GHz.
- The receiver gets the wave reflected from an object.
- Due to the Doppler effect,
  the reflected wave has different
  frequency than the emitted one.
- The frequency shift depends
  on the object speed.



Spectrogram of a sensor signal

The difference signal – a spectrogram from a passing vehicle



Signal analysis

The algorithm works as follows:

- the signal is analyzed in blocks of 2048 samples, with 50% overlap,
- the incoming signal samples are written into a circular buffer,
- when the buffer is full:
  - the buffer is multiplied by a Hamming window,
  - FFT is computed,
  - power spectrum is computed,
  - we look for spectral maxima (peaks),
  - a vehicle speed is computed from the frequency of a spectral peak.

Buffering of signal samples

- The incoming signal samples are written into a circular buffer of length 2048 (*short* numbers in Q15 format).
- The window is moved by 1024 samples. When we get 1024 new samples, then:
  - we loop over the buffer, from the oldest to the latest sample,
  - we multiply samples by values of the Hamming window (using _*smpy* function),
  - we store the result in a linear buffer.

Window function

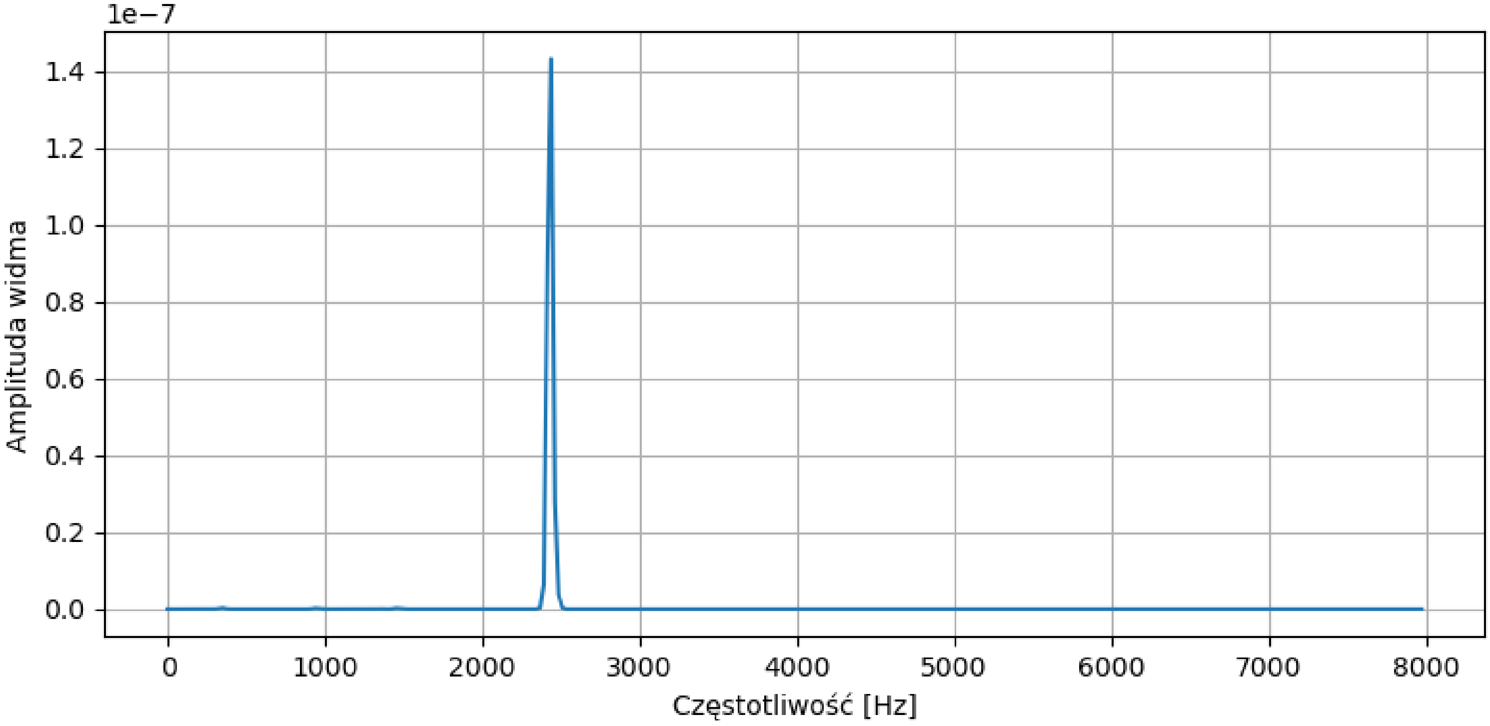
- There is no need to compute Hamming window each time. The window values are constant for a given length.
- We compute the window values using a software.
- The values are converted to Q15.
- They are stored in a table (*const short*) in C code.
- Warning: the maximum value of the window is 1. We cannot write 1 in Q15! We have to scale the window, multiplying by 32767 instead of 32768.

Power spectrum

- We compute the complex spectrum with *rfft*.
- We compute the power spectrum:
  - iterate over pairs (Re, Im),
  - compute a square of each part (*_smpy*),
  - sum up the squared real and imaginary part,
  - write the result to the buffer (we can use the same buffer).
- For amplitude spectrum, the square root of the result should be computed (*sqrt_16* function from DSPLIB).

Spectrum of a single block

Next, we analyze the spectrum, looking for the peaks.



Peak finding

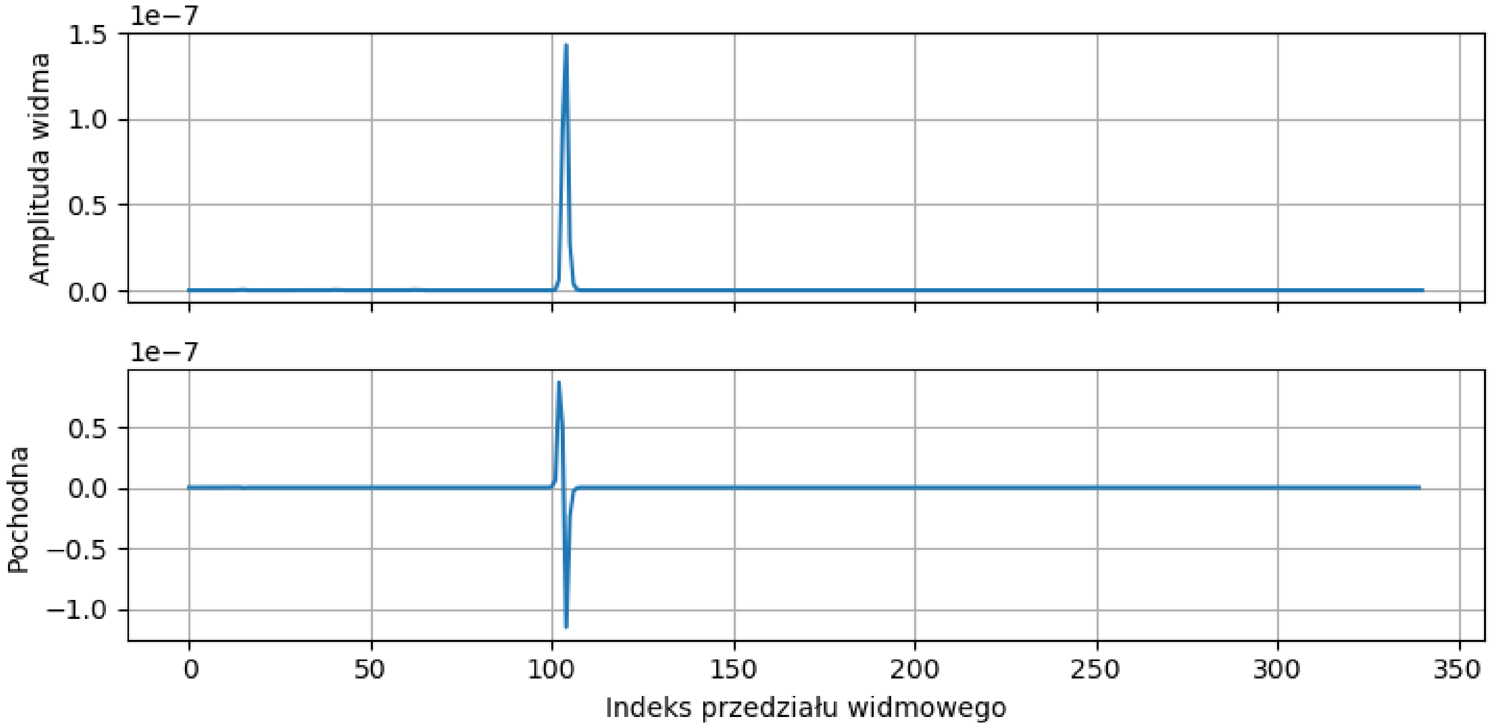
There are many methods of peak finding. An example:

- we compute a derivative of the spectrum: from each spectral value, we subtract the previous one,
- if the derivative crosses zero going down, it means the maximum is at that position,
- we also need to check the spectral amplitude to reduce the noise effect,
- we can also add other conditions, e.g. the maximum width of the peak.

Peak finding

Power spectrum and its derivative.

A peak is found at index 104.



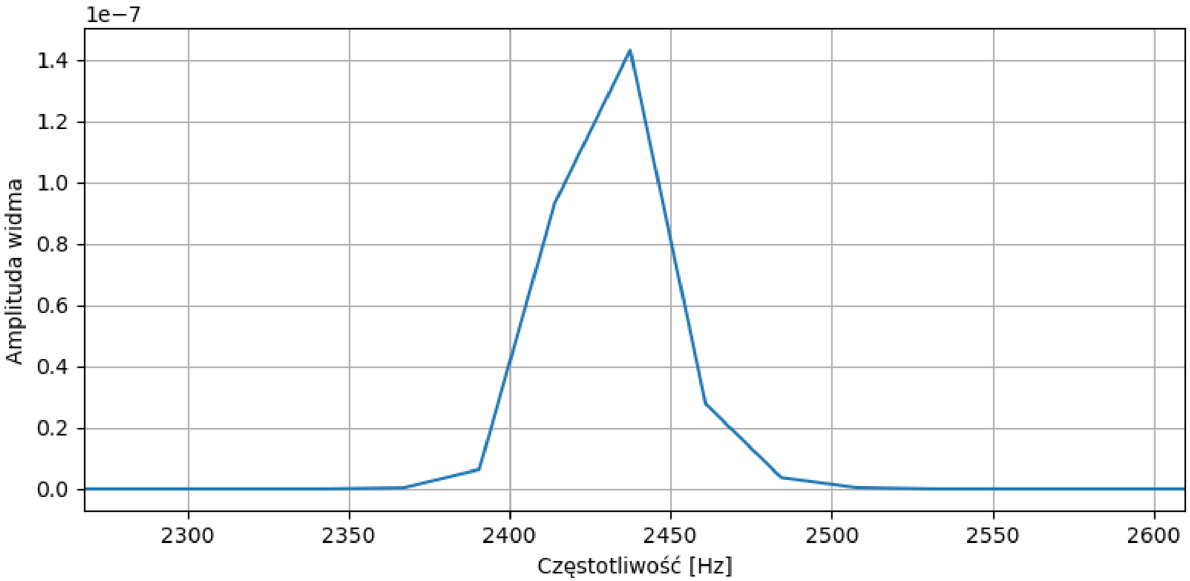
Velocity calculation

All that is remaining is to compute the speed.

- Relation between frequency and speed:
  $v \approx 0.02234 \cdot f$ (from the Doppler equation)
- Relation between the spectral index n and the frequency f, assuming $f_S = 48$ kHz: $f = 23.4375 \cdot n$
- So: $v \approx 0.52425 \cdot n$ [km/h]
- In Q15: $v \approx 17179 \cdot n$
- We compute: 104 * 17179 = 1786616
- In a decimal notation:
  1786616 / 32768 ≈ 54.52 km/h

Accuracy of speed measurement

- Remember that the accuracy of frequency calculation depends on the frequency resolution. For N = 2048, the maximum error is 11.72 Hz, c.a. 0.26 km/h.
- The index of the peak may not indicate the real peak.



Accuracy of peak finding

- We can improve the peak finding accuracy, but this method requires a division, so this algorithm is rather for floating-point processors.
- Three points determine a parabola.
- We match a parabola to the spectral peak and its two neighbors. Their values are: *a*, *b*, *c*.
- The parabola peak position is given by:

$$m = n + \frac{1}{2}\frac{a-c}{a-2b+c}$$

- In our example: m = 103.8; v = 54.42 (was 54.52).

Źródło: https://ccrma.stanford.edu/~jos/sasp/Quadratic_Interpolation_Spectral_Peaks.html

Accuracy of peak finding

The result of parabola matching and finding its maximum (×):