7 Frequenzraum

Markus Lippitz
20. Mai 2022

Ziele Sie können Julia benutzen, um Signale im Frequenzraum zu *analysieren*.

- Fast Fourier Transform FFT
- Frequenzachse
- Abtastrate, Länge und 'zero padding'

Weitere Aufgaben

- Signale transformieren und im Fourier-Raum filtern

Literatur Butz Kap. 4, Horowitz / Hill, Kap. 1.08, 7.20, 15.18

Diskrete FT: eine periodische Zahlenfolge und deren Fourier-Transformierte Zahlenfolge

Insbesondere wenn man mit einem Computer Messwerte erfasst und auswertet, dann kennt man die gemessene Funktion $f(t)$ weder auf einer kontinuierlichen Achse t, sondern nur zu diskreten Zeiten $t_k = k\,\Delta t$, noch kennt man die Funktion von $t = -\infty$ bis $t = +\infty$. Als Ausgangspunkt hat man also nur eine Zahlenfolge f_k endlicher Länge.

Weil wir die Zahlenfolge außerhalb des gemessenen Intervalls nicht kennen machen wir die Annahme, dass sie periodisch ist. Bei N gemessenen Werte ist die Periodendauer also $T = N\Delta t$. Der Einfachheit halber definieren wir auch $f_k = f_{k+N}$ und somit $f_{-k} = f_{N-k}$ mit $k = 0, 1, \ldots, N-1$. Damit wird die Fourier-Transformation

$$F_j = \frac{1}{N} \sum_{k=0}^{N-1} f_k \, e^{-k\,j\,2\pi i/N}$$

und die inverse Fourier-Transformation

$$f_k = \sum_{j=0}^{N-1} F_j \, e^{+k\,j\,2\pi i/N}$$

Diese Definition nach Butz ist wieder so, dass F_0 dem Mittelwert entspricht. Wegen $f_{-k} = f_{N-k}$ liegen die positiven Frequenzen mit steigender Frequenz in der ersten Hälfte von F_j. Danach kommen die negativen Frequenzen, beginnend bei der 'negativsten' Frequenz steigend mit zur letzten Frequenz vor der Frequenz Null. Die maximal darstellbare Frequenz ist also die Nyquist-Frequenz

$$f_{\text{Nyquist}} = \frac{1}{2\Delta t}$$

bzw. Nyquist-Kreis-Frequenz

$$\Omega_{\text{Nyquist}} = \frac{\pi}{\Delta t}$$

Diese Frequenz ist also gerade so, dass wir zwei Samples pro Periode der Oszillation messen. Schnellere Oszillationen bzw. weniger Samples pro Periode sind nicht darstellbar. Schon bei f_{Nyquist} ist der Imaginärteil immer Null, weil wir den Sinus gerade immer in Nulldurchgang sampeln.

Es ist je nach geradem oder ungeraden N ein klein wenig aufwändig, die jeweiligen Frequenzen zu berechnen. Wir benutzen hier und im folgenden das Paket FFTW, das dies für uns erledigt.

Hier als Beispiel die Frequenzen bei $\Delta t = 1$ und 5 bzw 6 Elementen

```
1 using FFTW
```

```
AbstractFFTs.Frequencies{Float64}: [0.0, 0.2, 0.4, -0.4, -0.2]
```

```
1 fftfreq(5)
```

```
AbstractFFTs.Frequencies{Float64}: [0.0, 0.166667, 0.333333, -0.5, -0.333333, -0.1666
```

```
1 fftfreq(6)
```

FFTW

In Julia können wir das Paket FFTW benutzen. Dabei wechselt allerdings der Vorfaktor $1/N$ von der Hin- zur Rück-Transformation, also

$$F_j = \sum_{k=0}^{N-1} f_k \, e^{-k\,j\,2\pi i/N}$$

und die inverse Fourier-Transformation

$$f_k = \frac{1}{N} \sum_{j=0}^{N-1} F_j \, e^{+k\,j\,2\pi i/N}$$

In Gleichungen benutze ich (und Butz) mathematische Indizes (startend von Null). Julia zählt ab Eins, also ausnahmsweise hier zwei Gleichungen mit Julia-Indizes

$$F_{\tilde{j}} = \sum_{\tilde{k}=1}^{N} f_{\tilde{k}} \, e^{-(\tilde{k}-1)\,(\tilde{j}-1)\,2\pi i/N}$$

und die inverse Fourier-Transformation

$$f_{\tilde{k}} = \frac{1}{N} \sum_{\tilde{j}=1}^{N} F_{\tilde{j}} \, e^{+(\tilde{k}-1)\,(\tilde{j}-1)\,2\pi i/N}$$

Die Fourier-Transformierte einer Konstanten ist die Delta-Funktion, also ist nur das erste Element, also das bei $f = 0$ von Null verscheiden und gleich der Summe der Werte im Zeitraum

```
1×4 Matrix{ComplexF64}:
 4.0+0.0im  0.0+0.0im  0.0+0.0im  0.0+0.0im
```

```
1 fft([1 1 1 1])
```

Die inverse FFT geht auch und beinhaltet hier das $1/N$

```
1×4 Matrix{ComplexF64}:
 0.25+0.0im  0.25+0.0im  0.25+0.0im  0.25+0.0im
```

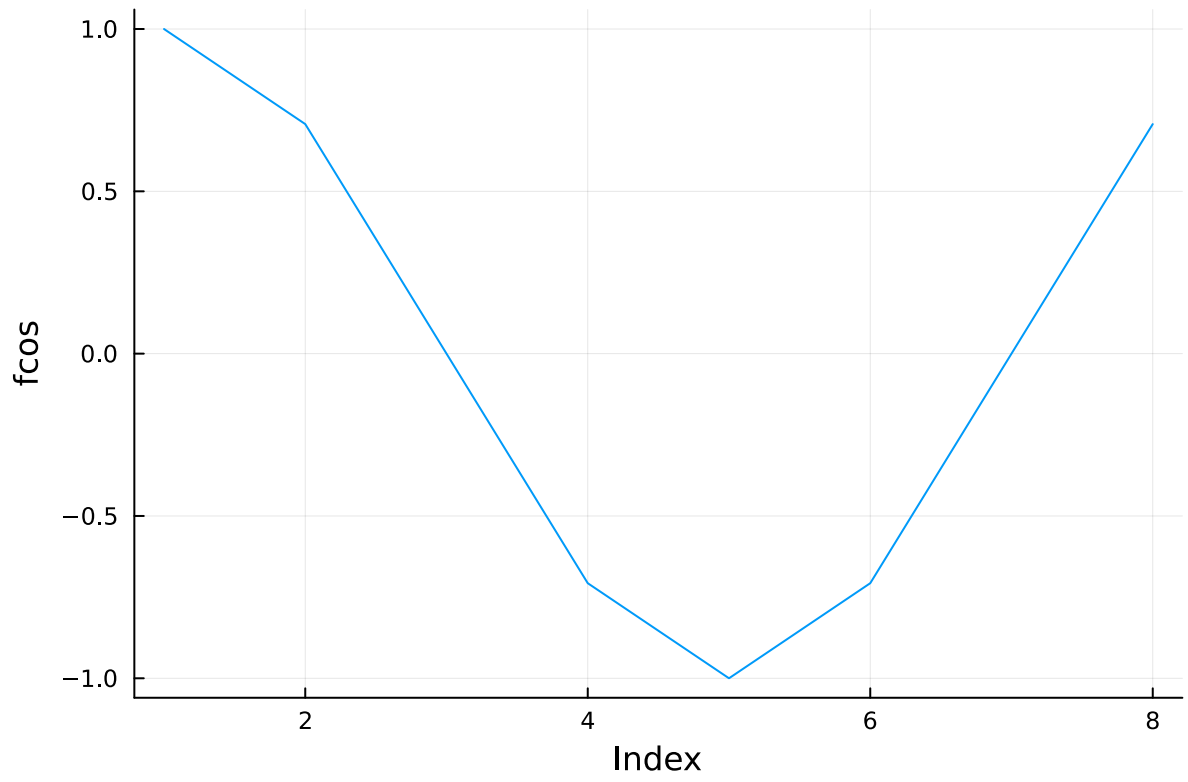
```
1 ifft([1 0 0 0])
```

Wrapping & fftshift

Im nächsten Beispiel Fourier-transformieren wir einen Kosinus

```
f_cos =
▸ [1.0, 0.707107, 6.12323e-17, -0.707107, -1.0, -0.707107, -1.83697e-16, 0.707107]
```

```
1 f_cos = cos.((0:7)./8 .* 2pi)
```

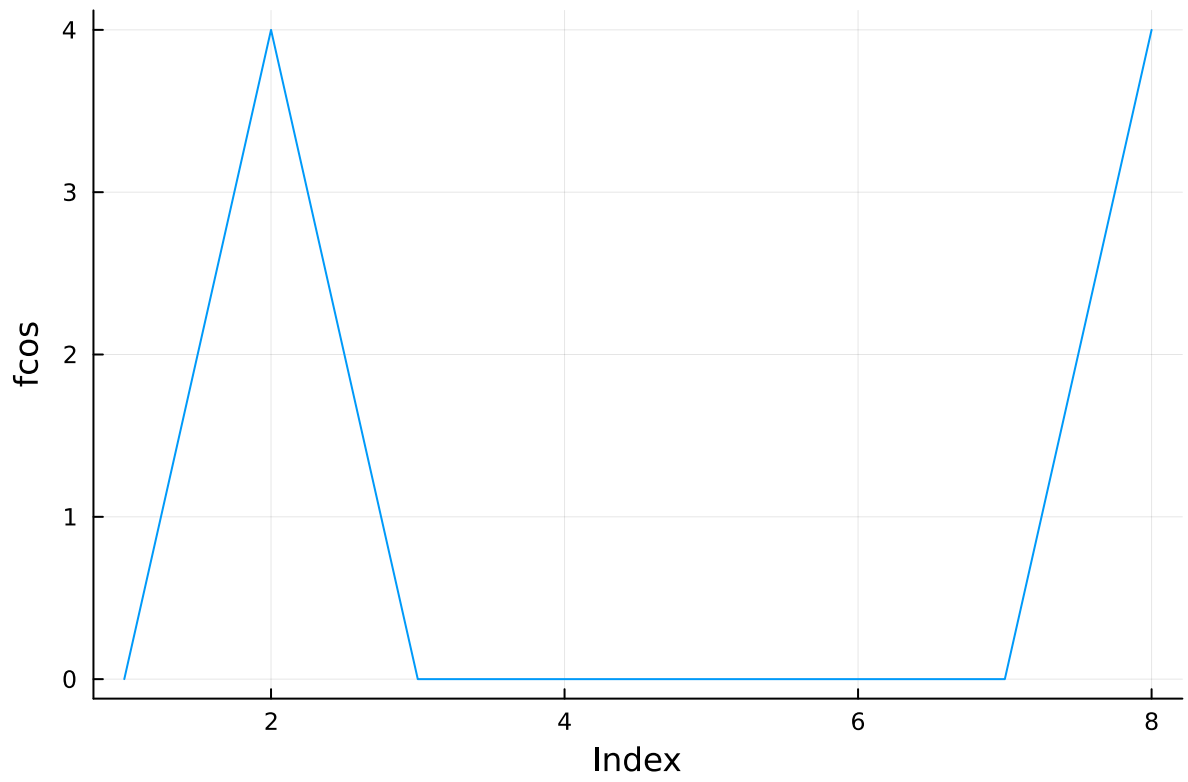



```
1 aside(embed_display(plot(f_cos, leg=false, xlabel="Index", ylabel="fcos")))
```

In diesem Fall ist die FT rein reelwertig und wir ignorieren den Imaginärteil an der numerischen Rauschgrenze.

```
F_cos =
▸ [-3.44509e-16, 4.0, 1.22465e-16, -4.44089e-16, 9.95799e-17, -4.44089e-16, 1.22465e-16,
```

```
1 F_cos = real.(fft(f_cos))
```




```
aside(embed_display(plot(F_cos, leg=false, xlabel="Index", ylabel="fcos")))
```

Von Null verschieden sind die Werte beim Julia-Index 2 und 8, mathematisch also 1 und 7. Generell gilt bei reellen Ausgangswerten

$$F_{N-j} = F_j^\star$$

Hier also $F_1 = F_7$

Es müssen zwei Werte von Null verschieden sein, weil

$$\cos(x) = \frac{1}{2}\left(e^{ix} + e^{-ix}\right)$$

Die Position dieser zwei von Null verschiedenen Werte ist eine Folge der Definition der F_k: zunächst kommen alle positiven Frequenzen und dann alle negativen, oder

```
▸ AbstractFFTs.Frequencies{Float64}: [0.0, 0.125, 0.25, 0.375, -0.5, -0.375, -0.25, -0.1
```

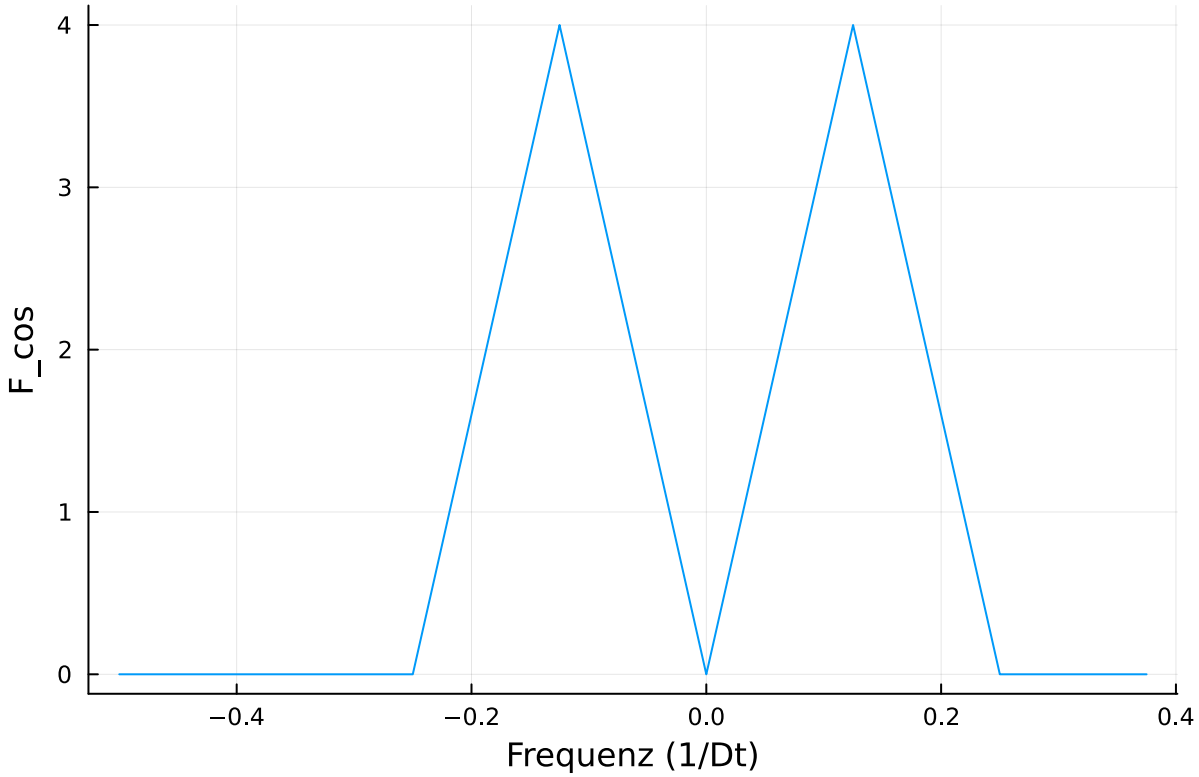
```
fftfreq(length(f_cos))
```

Zur Darstellung schöner ist es oft, wenn die Frequenz Null nicht am Rand sondern in der Mitte zwischen den positiven und negativen Frequenzen ist. Dies bewirkt fftshift bzw. rückwärts ifftshift

```
-0.5:0.125:0.375
```

```
1 fftshift(fftfreq(length(f_cos)))
```

Damit können wir alles über der Frequenzachse darstellen




```
1 plot(fftshift(fftfreq(length(f_cos))), fftshift(real(fft(f_cos))),
  xlabel="Frequenz (1/Dt)", ylabel="F_cos", leg=false)
```

Selbsttest

Ersetzen Sie den Cosinus durch einen Sinus in diesem Beispiel und erklären Sie das Ergebnis.

Sampling-Theorem

Wir brauchen mindestens zwei Samples pro Periode, um eine Funktion durch ihre Fourier-Koeffizienten darstellen zu können. Die Frequenzen müssen also unterhalb der Nyquist-Frequenz f_{Nyquist} liegen mit

$$f_{\mathrm{Nyquist}} = \frac{1}{2\Delta t} \quad .$$

Das Sampling-Theorem besagt, dass dies dann aber auch ausreichend ist.

Sei $f(t)$ eine bandbreiten-begrenzte Funktion, also $F(\omega)$ nur im Intervall $|\omega| \leq \Omega_{\mathrm{Nyquist}}$ von Null verschieden. Dann gilt das Sampling-Theorem (Beweis siehe Butz Kap.4.4)

$$f(t) = \sum_{k=-\infty}^{\infty} f(k\Delta t)\,\mathrm{sinc}\left(\Omega_{\mathrm{Nyquist}} \cdot [t - k\Delta t]\right)$$

Es reicht also aus, f alle Δt zu samplen. An den Zeiten dazwischen ist f durch die (unendlich lange) Summe der benachbarten Werte mal sinc vollständig beschrieben.

In der Messtechnik müssen wir also nur beispielsweise durch einen Filter sicherstellen, dass alle Frequenzen eines Signals unter $\Omega_{\mathrm{Nyquist}}$ liegen, und unsere digitale Erfassung des Signals ist identisch mit dem Signal selbst.

Wenn wir aber zu selten samplen, bzw. doch höhere Frequenzen vorhanden sind, dann werden diese zu hohen Frequenz-Komponenten an der dann niedrigeren Nyquist-Frequenz gespiegelt und landen bei scheinbar niedrigeren Frequenzen. Dieses 'aliasing' verfälscht dann das Signal.

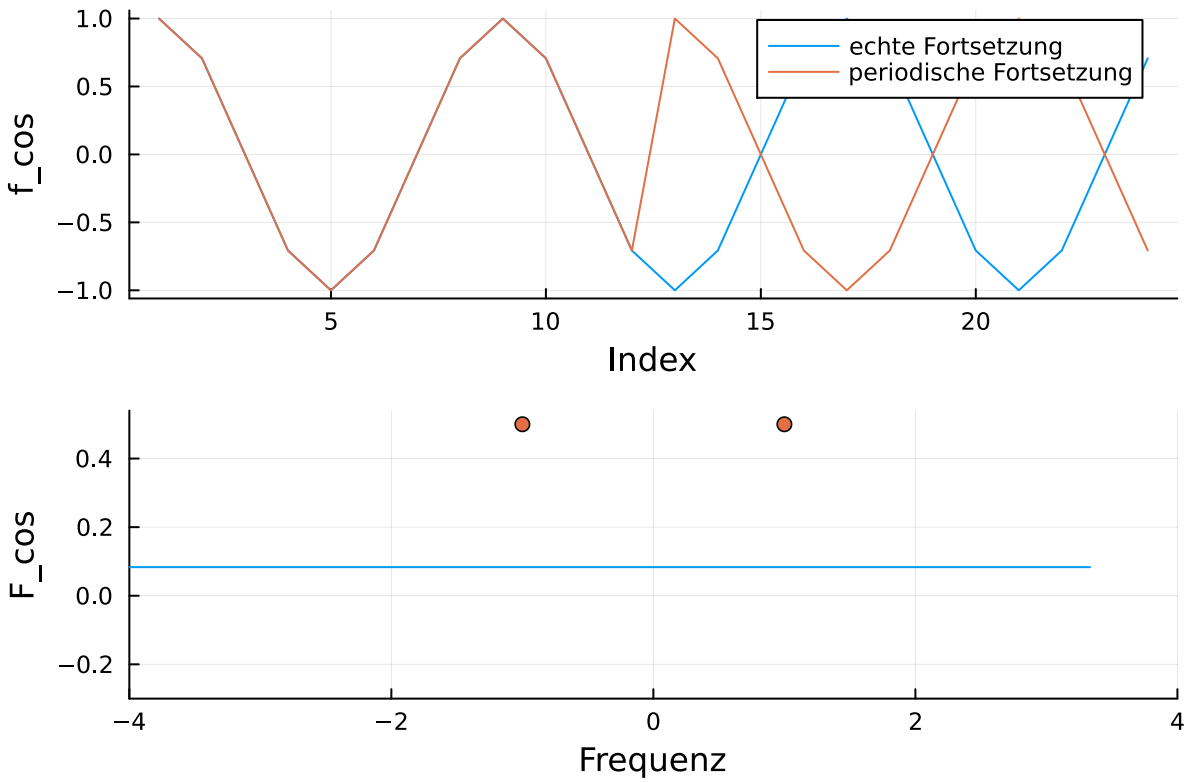
Zero padding

Wir hatten ganz oben angefangen mit einer periodischen Zahlenfolge und deren Fourier-Transformation. Die Länge der Zahlenfolge war in den Beispielen immer so gewählt, dass dies gerade einem ganzzahligen Vielfachen der Periodendauer entsprach. Das geht aber natürlich in der Praxis nicht. Wir kennen in Zweifelsfall die Periodendauer des Signals nicht, oder nicht genau genug. Oder es sind sogar mehrere Signale mit unterschiedlicher Frequenz von Interesse.

Das Problem ist dann der Abschneide-Fehler, der zu Artefakten in der Fourier-Transformation führt.

Stellen Sie in diesem Beispiel n_{sample} so ein, dass der Fehler minimal wird!

Datenpunkte im Sample n_{sample} 12




```
let
    ts = 1/8
    time = (0:n_sample-1) .* ts;
    f_cos = cos.(2pi * time)

    time2 = (0:2 *n_sample-1) .* ts;
    f_cos2 = cos.(2pi * time2)

    plot(f_cos2, layout=(2,1), label="echte Fortsetzung")
    plot!([f_cos; f_cos], xlabel="Index", ylabel="f_cos", label="periodische
    Fortsetzung")
    plot!(fftshift(fftfreq(length(f_cos),1/ts)),
        fftshift(real.(fft(f_cos)))./n_sample,
        yrange=(-0.3,0.54), xrange=(-4,4), xlabel="Frequenz", ylabel="F_cos",
        leg=false,layout=(2,1), subplot=2)
    scatter!([-1, 1], [0.5, 0.5], subplot=2)
end
```

Der Ausweg ist **zero-padding**. Sei unsere eigentlich gemessene Signalfolge $f(t)$, die wir im Intervall $[-T, T]$ kennen. Nun tun wir so, als hätten wir statt dessen gemessen

$$g(t) = f(t) \cdot w(t)$$

mit der Fensterfunktion $w(t)$

$$w(t) = 1 \quad \text{für} \quad -T < t < T \quad \text{sonst} = 0$$

Damit können wir $g(t)$ über beliebig lange Zeiten 'messen', weil es ja quasi immer Null ist. Die Fourier-Transformierte ist aber

$$G(\omega) = F(\omega) \otimes W(\omega)$$

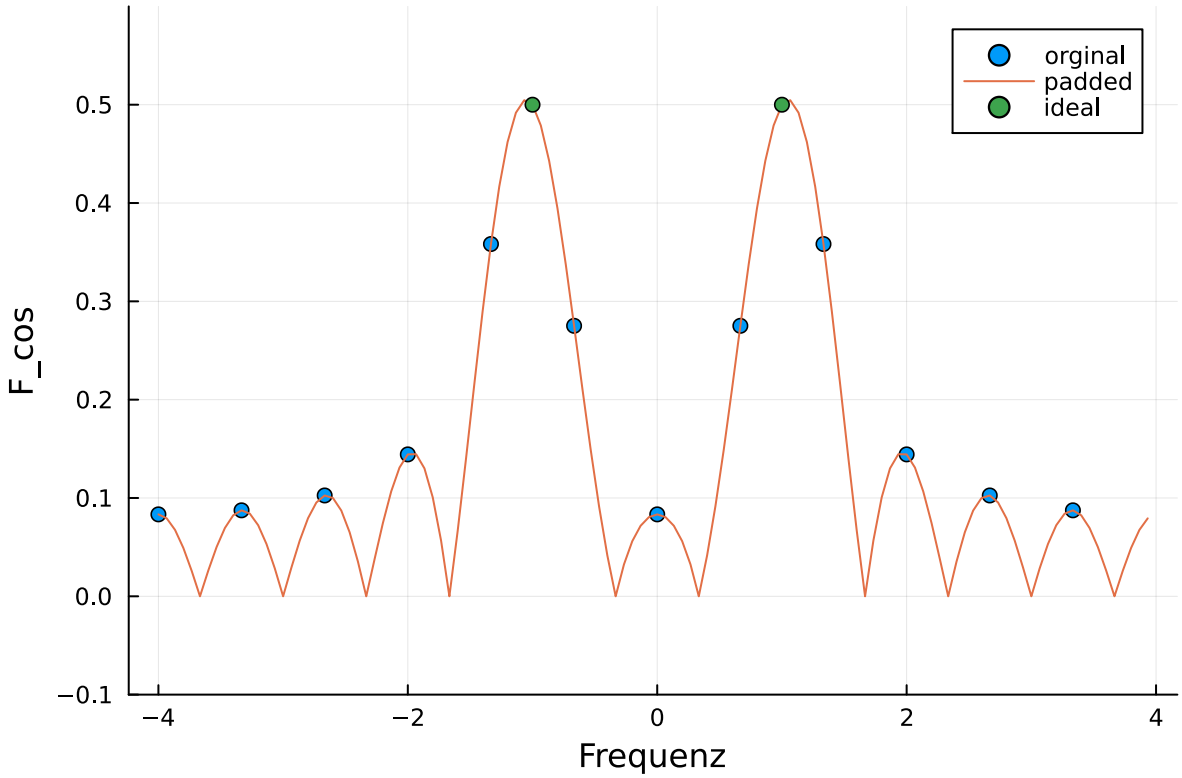
mit

$$W(\omega) = 2T \, \frac{\sin \omega T}{\omega T} = 2T \, \mathrm{sinc}(\omega T)$$

Wir verlängern also unseren Datensatz zu beiden Seiten mit Nullen. Der Effekt ist, dass wir die eigentliche Fourier-Transformation unseres Datensatzes falten mit einem sinc, dessen charakteristische Breite durch die eigentliche Messdauer bestimmt ist. Die Frequenzauflösung steigt dadurch also nicht. Vielmehr geschieht eine Art Interpolation im Fourier-Raum, die gerade die Artefakte des Abschneide-Fehlers beseitigt.

Wir betrachten den gleichen Datensatz wie oben, nur 'verlängern' wir ihn auf die 10-fache Länge. Dadurch hat der Abschneidefehler weniger Einfluss und der Peak liegt im Frequenzraum immer bei 1 Hz. Das erzeugt aber natürlich nicht mehr Auflösung. Nahe beieinander liegende Peaks können durch zero-padding nicht getrennt werden, nur die Position eines Peaks besser bestimmt werden.

Datenpunkte im Sample 12




```
let
    ts = 1/8
    time = (0:n_sample2-1) .* ts;
    f_cos = cos.(2pi * time)
    F_cos = fft(f_cos)

    padded = [ f_cos; zeros(9 .* size(f_cos))]
    F_padded = fft(padded)

    scatter(fftshift(fftfreq(length(F_cos), 1/ts)), fftshift(abs.
    (F_cos))./n_sample2,
        yrange=(-0.1,0.6), xlabel="Frequenz", ylabel="F_cos", label="orginal")
    plot!(fftshift(fftfreq(length(F_padded), 1/ts)), fftshift(abs.
    (F_padded))./n_sample2, label="padded")
    scatter!([-1, 1], [0.5, 0.5], label="ideal")
end
```

Windowing

Die Oszillationen im Spektrum im letzten Beispiel sind immer noch Artefakte. Eigentlich würde man ja zwei Delta-Funktionen bei ± 1Hz erwarten. Sie sind eine Konsequenz des Rechteck-Fensters $w(t)$, das zum sinc in Frequenzraum führt. Das Rechteck-Fenster ist in dem Sinne natürlich, dass wir immer zu einen bestimmten Zeitpunkt anfangen und aufhören, zu messen.

Andere Fensterfunktionen sind aber unter Umständen besser. Sie unterscheiden sich in der Breite des Peaks und im Abfall der Flanken. Leider muss man aber das eine gegen das andere einhandeln. Interessante Parameter sind die Breite des zentralen Peaks im Frequenzraum, gemessen als -3dB-Bandbreite, sowie die Seitenbandenunterdrückung in dB oder deren Abfall in dB/Oktave.

dB = Dezibel = 10 log_10 x

Typische Fensterfunktionen sind im Bereich $|x| = |t/T| < 1/2$

$$\text{Kosinus} = \cos \pi x$$

$$\text{Dreieck} = 1 - 2|x|$$

$$\text{Hanning} = \cos^2 \pi x$$

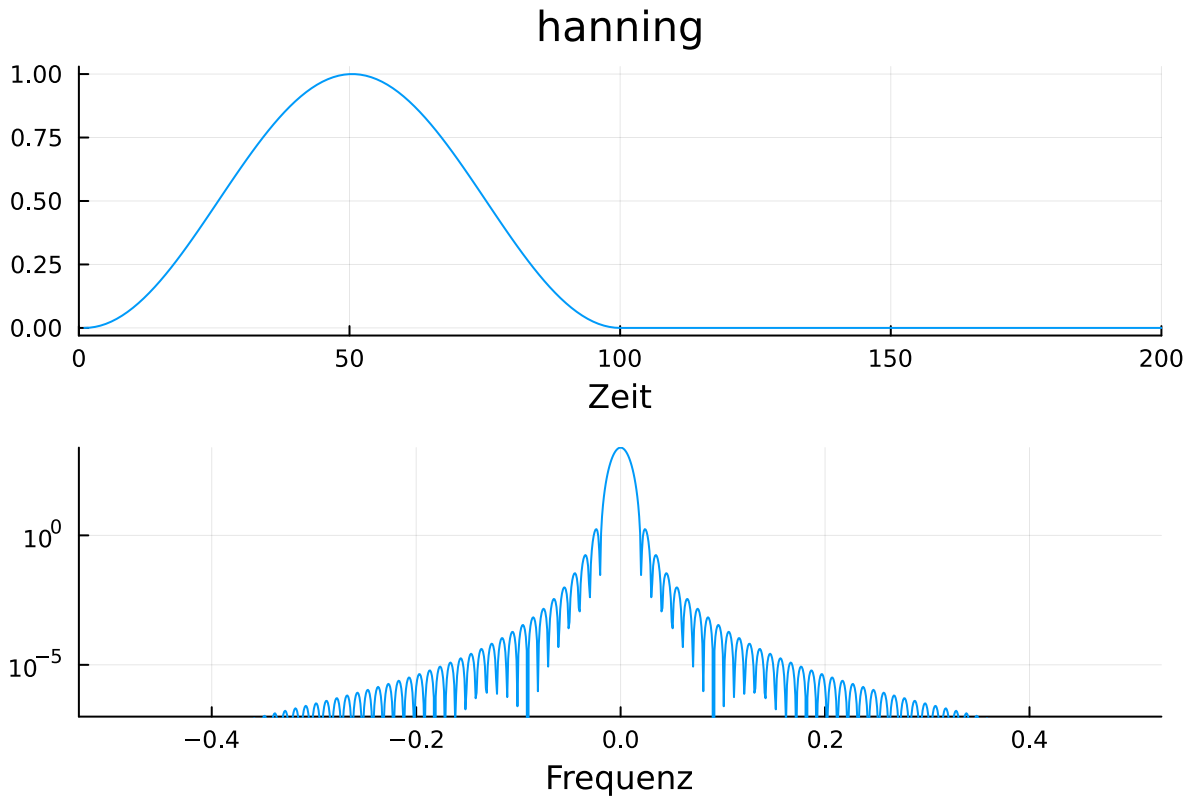
$$\text{Hamming} = a + (1-a)\cos^2 \pi x \quad \text{in DSP.jl mit } a = 0.54$$

$$\text{Gauss} = \exp\left(-\frac{1}{2}\frac{x^2}{\sigma^2}\right)$$

$$\text{Kaiser-Bessel} = \frac{I_0(\pi\alpha\sqrt{1-4x^2})}{I_0(\pi\alpha)} \quad \text{mit der modifizierten Bessel-Funktion } I_0$$

```
1 using DSP
```

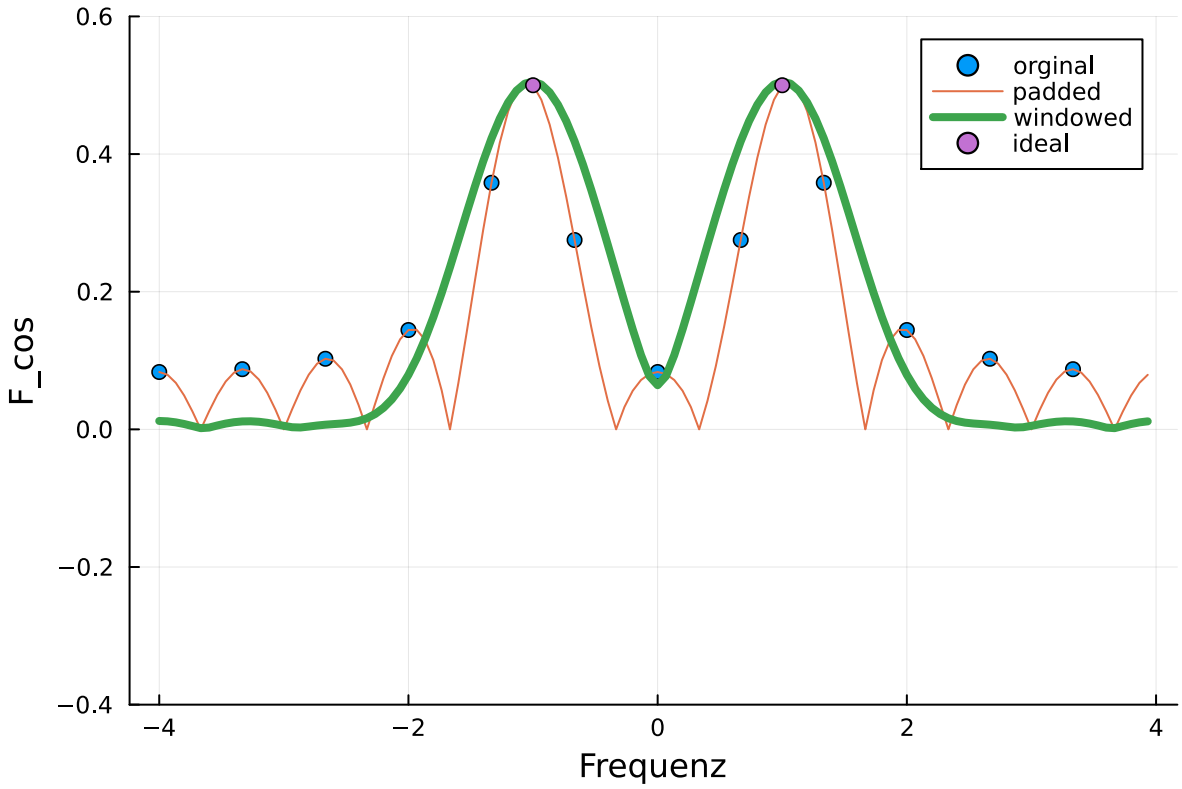
Fensterfunktion hanning , Parameter σ bei Gauss bzw. 1/α bei Kaiser 0.5




```
let

    if  (window=="gaussian")
        expr = "DSP.Windows.$(window)(100, $(win_p); padding=1000)"
    else
        if (window=="kaiser")
            expr = "DSP.Windows.$(window)(100, $(1/win_p); padding=1000)"
        else
            expr = "DSP.Windows.$(window)(100; padding=1000)"
        end
    end
    w = eval(Meta.parse(expr))

    x = fftshift(fftfreq(length(w)))
    fftabs(y) = fftshift(abs.(fft(y).^2))

    plot(w, layout=(2,1), title=window, xrange=(0,200), subplot=1, leg=false,
    xlabel="Zeit")
    plot!(x, fftabs(w), yaxis = (:log10,  (1e-7,Inf)), subplot=2, leg=false,
    xlabel="Frequenz")
end
```

Mit einem Fenster verkleinert man zwar die gemessenen Werte, macht die Fourier-Transformation aber glatter, weil der Übergang zum zero-Padding glatter wird. Damit erkennt man die Peaks bei ± 1Hz auch schon bei sehr wenig gesampelten Punkten.

Datenpunkte im Sample 12



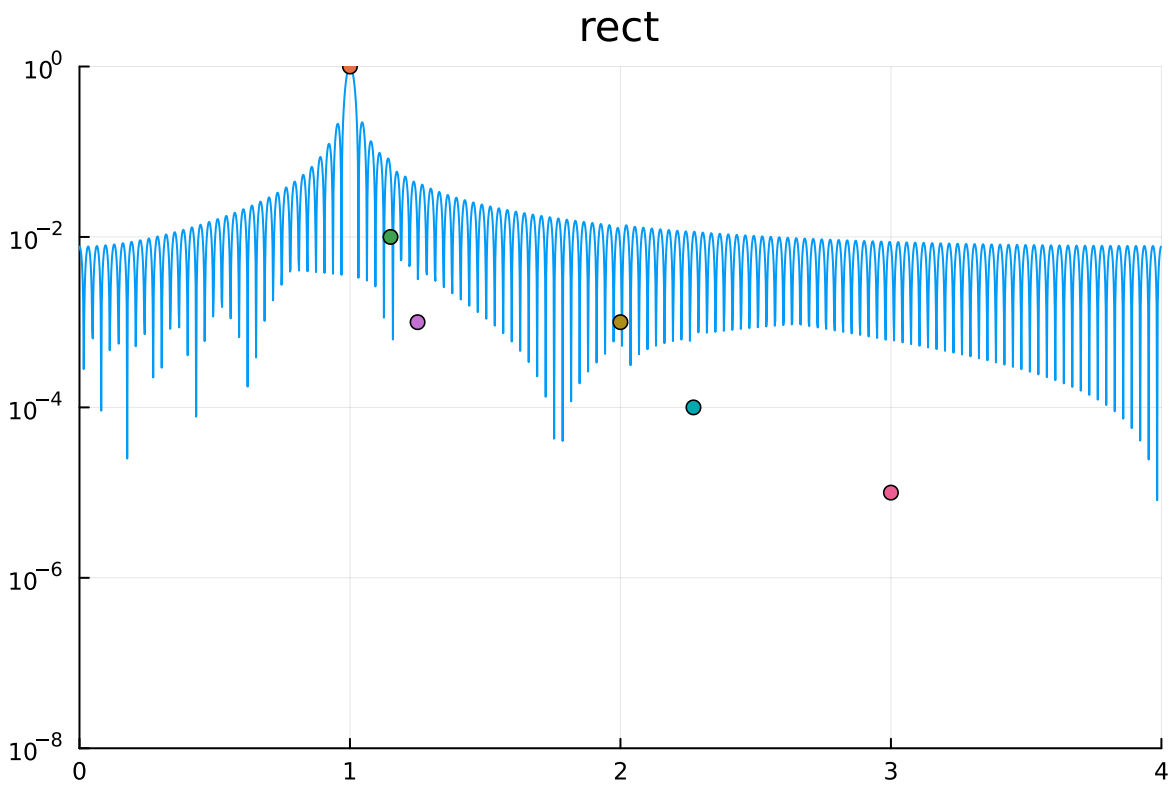

```
let
    ts = 1/8
    time = (0:n_sample3-1) .* ts;
    f_cos = cos.(2pi * time)
    F_cos = fft(f_cos)

    padded = [ f_cos ; zeros(9 .* size(f_cos))]
    F_padded = fft(padded)

    window = DSP.Windows.hamming(length(f_cos))
    window = window .* 2
    windowed = [ f_cos .* window; zeros(9 .* size(f_cos))]
    F_windowed = fft(windowed)

    scatter(fftshift(fftfreq(length(F_cos), 1/ts)), fftshift(abs.
    (F_cos))./n_sample3,
        yrange=(-0.4,0.6), xlabel="Frequenz", ylabel="F_cos", label="orginal")
    plot!(fftshift(fftfreq(length(F_padded), 1/ts)), fftshift(abs.
    (F_padded))./n_sample3, label="padded")
    plot!(fftshift(fftfreq(length(F_windowed), 1/ts)),
        fftshift(abs.(F_windowed))./n_sample3, label="windowed", linewidth = 4 )
    scatter!([-1, 1], [0.5, 0.5], label="ideal")
end
```

Beispiel

aus Butz, Kapitel 3.10

Wir betrachten eine Summe aus 6 Kosinus-Funktionen mit teils sehr unterschiedlichen Amplituden A_l und Frequenzen f_l. Wir samplen 256 Datenpunkte im Abstand von $1/8$ der längsten Periode, also nur $8/3$ Datenpunkte pro Oszillation der höchsten vorkommenden Frequenz. Diese ist um 5 Größenordnungen schwächer als die niedrigste Frequenz. Trotzdem findet man diesen Peak durch ein passendes Fenster und zero-padding.

Fensterfunktion rect , Parameter σ bei Gauss bzw. 1/α bei Kaiser 0.5




```
let
    fl = [1 1.15   1.25   2     2.27      3]
    Al = [1 1e-2   1e-3  1e-3    1e-4   1e-5]

    dt = 1/8
    time = (-127:127).*dt;
    signal = [ sum(Al .* cos.(2pi .* fl .* t)) for t in time];

    if (window2=="gaussian")
        expr = "DSP.Windows.$(window2)($(length(signal)),$(win_p2))"
    else
        if (window2=="kaiser")
            expr = "DSP.Windows.$(window2)($(length(signal)),$(1/win_p2))"
        else
            expr = "DSP.Windows.$(window2)($(length(signal)))"
        end
    end

    window = eval(Meta.parse(expr))
    windowed = [(signal.* window); zeros(15 .* size(signal))]
    F_windowed = abs.(fft(windowed))

    plot(fftshift(fftfreq(length(F_windowed), 1/dt)),
         fftshift(F_windowed)./maximum(F_windowed),
        legend=false, yaxis=(:log, (1e-8, Inf)), xrange=(0, 4)   , title=window2)
    scatter!(fl, Al)
end
```

Spielwiese: 2D FFT von Bildern

```
1 using StatsBase
```

```
1 using  Colors, ImageShow, ImageIO
```

Erlauben Sie ihrem Webbrowser, auf die Kamera zuzugreifen und nehmen Sie dann einen Schnappschuss auf! Zeigen Sie Ihrer Kamera einfache Muster und vergleichen Sie die Fourier-Transformierte mit Ihren Erwartungen.

```
1 @bind webcam_data camera_input()
```

```
1 snapshot = process_raw_camera_data(webcam_data)
```

```
1 spatial_data = Real.(Gray.(snapshot));
```

```
1 freq_data = abs.(fftshift(fft(spatial_data .- mean(spatial_data) )));
```

```
1 ft_snapshot = Gray.(freq_data ./ maximum(freq_data))
```

```
1 using PlutoUI
```

```
1 using Plots
```

Inhalt


```
TableOfContents(title="Inhalt")
```

```
aside(x) = PlutoUI.ExperimentalLayout.aside(x);
```

Webcam routines from
https://computationalthinking.mit.edu/Spring21/notebooks/week1/images.html

```
process_raw_camera_data (generic function with 1 method)
```

```
camera_input (generic function with 1 method)
```