

Skript Digitale Signalverarbeitung

Sebastian Semper – FG Elektrische Messtechnik und Signalverarbeitung – EMS

3. Februar 2025

Inhaltsverzeichnis

Abbildungsverzeichnis	iii
Codeschnipselverzeichnis	iv
Linkverzeichnis	iv
1 Theoretische Grundlagen	1
1.1 Komplexe Zahlen	1
1.2 Signale	1
1.2.1 Definition und Typen	1
1.2.2 Signale als Vektoren	2
1.2.3 Transformation von Signalen	2
1.2.4 Zufällige Signale	3
1.2.5 Spezielle Signale	4
1.2.6 Beispiel: LTI-Systeme	5
2 Abtastung von Signalen	6
2.1 Frequenz von Signalen	6
2.1.1 Zeit-Kontinuierliche Sinussignale	6
2.1.2 Zeit-Diskrete Sinussignale	8
2.2 Zeit-Kontinuierliche Komplexe Sinussignale	9
2.3 Zeit-Diskrete Komplexe Sinussignale	10
2.4 Abtastung von Sinussignalen	12
2.5 Das Samplingtheorem	15
3 Diskrete Signale und Systeme	21
3.1 Diskrete Signale	21
3.1.1 Energie Diskreter Signale	21
3.1.2 Periodische Signale	23
3.1.3 Symmetrie von Signalen	24
3.2 Diskrete Systeme	24
3.3 Diskrete LTI-Systeme	30

3.3.1	Faltungsformel	30
3.3.2	Eigenschaften von LTI-Systemen	34
3.3.3	Rekursive IIR-Systeme	35
3.3.4	Approximation von IIR-Systemen durch FIR-Systeme	37
3.4	<i>z</i> -Transformation	39
3.4.1	Eigenschaften	41
3.4.2	Anwendung bei diskreten LTI-Systemen	42
4	Fourier-Transformation	46
4.1	Fourier-Transformation kontinuierlicher Signale	46
4.1.1	Fourier-Transformation kontinuierlicher periodischer Signale	46
4.1.2	Leistungsdichte-Spektrum periodischer Signale	50
4.1.3	Fourier-Transformation von kontinuierlichen aperiodischen Signalen	53
4.1.4	Leistungsdichte-Spektrum aperiodischer Signale	54
4.2	Fourier-Transformation diskreter Signale	56
4.2.1	Fourier-Transformation diskreter periodischer Signale	56
4.2.2	Leistungsdichte-Spektrum diskreter periodischer Signale	57
4.2.3	Fourier-Transformation diskreter aperiodischer Signale	58
4.2.4	Leistungsdichte-Spektrum diskreter aperiodischer Signale	61
4.3	Eigenschaften der Fourier-Transformationen	61
4.3.1	Beziehung der Fourier-Transformationen und der <i>z</i> -Transformation	61
4.3.2	Weitere Eigenschaften der DFT	64
5	Anwendung der Disreten Fourier-Transformation	68
5.1	Spektralanalyse mit der Short Time Fourier Transformation	68
5.2	Interpolation mittels DFT	75
5.2.1	Theoretische Grundlagen	75
5.2.2	Interpolation von Antennen-Richtcharakteristiken	76
5.3	Schnelle Korrelation für Radar-Signalverarbeitung	81
5.3.1	Zyklische Korrelationen	81
5.3.2	Erzeugung der Sendesequenz	84
5.3.3	MLBS als Sendesequenz	85
6	Signalverarbeitung mit B-Splines	88
6.1	B-Splines als Polynome	88
6.2	Kubische B-Spline Interpolation	90
6.3	Verbindung zur Nyquist-Sampling-Theorie	93
Abkürzungsverzeichnis		95
Literaturverzeichnis		96

Abbildungsverzeichnis

1	$x_a(t) = A \cos(2\pi F t + \theta)$, Quelle: [1]	7
2	Uniforme Abtastung einer Signals. Quelle: [1]	13
3	zeigt zwei verschiedene systemtheoretische Interpretation von linearen Systemen. Quelle: [1]	30
4	Veranschaulichung der Entstehung der ROC aus den Konvergenzkriterien. Quelle: [1]	40
5	Impulsantwort $h[n] = a^n u[n]$ für verschiedene Werte von a , Quelle: [1]	44
6	Mehr Versionen von Listing 15; Oben: $x(t) = \exp(-25(t - 0.5)^2) \cos(16\pi t)$; Unten: $x(t) = \sin(20\pi t)$;	49
7	Schematische Darstellung einer MIMO Kanalmessung. Grafik aus [5].	76
8	Links Oben: Darstellung der Messpositionen für eine Antenne, oder eine Antennenstruktur, welche sich im Ursprung des abgebildeten Koordinatensystems befindet. Rechts Oben: Darstellung der winkelabhängigen Amplitude eines einzelnen Patch-Elements. Unten: Messpunkte für die Abtastung der Funktion a . Grafiken aus [8, 4].	78
9	Messaufbau zur Kalibrierung eines Antennenarrays. Links: Drehsteller für die Positionierung der AUT. Rechts: Weiterer Blickwinkel mit Referenzantenne.	79
10	Links: Periodifiziertes 2D Array $a[n_\varphi, n_\theta]$ der gemessenen Amplituden eines einzelnen Patch-Elements. Rechts: Der Betrag der zugehörigen EADF, wobei hier $\mu_1 = k_\varphi$ und $\mu_2 = k_\theta$. Grafik aus [8]	80
11	Lineares Feedback Shift Register, Quelle [1]	84
12	Kubische B-Splines für Abtastung an den Werten $n = 0, \dots, 12$	89
13	Kubische B-Spline-Interpolation für Abtastung an den Werten $n = 0, \dots, 12$	92

Codeschnipselverzeichnis

1	Berechnung und Darstellung von (2.1.1)	8
2	Berechnung und Darstellung von (2.1.2)	9
3	Berechnung und Darstellung von (2.3.1) für $f_0 = 1/16$ und $f_0 = 1/(5e)$	11
4	Visualisierung von $F_k = F + k \cdot F_s$	14
5	Berechnung und Darstellung von Theorem 2.1	19
6	Berechnung und Darstellung eines komplexen exponentiellen Signals	22
7	Zerlegung eines Signals in seinen geraden und ungeraden Anteil.	24
8	Spät divergierende Orbits der Mandelbrot-Iteration	26
9	Akkumulator für zwei verschiedene Eingangssignale.	27
10	Gleitendes Mittel mit Länge $\ell = 4$. Wir vergleichen die direkte Berechnung mit der Berechnung über die Faltung	32
11	Verkettung von mehreren Moving Averages der Länge $\ell = 3$	33
12	Die beiden möglichen Implementierungen eines kumulativen Mittelwertes	36
13	Iteratives Verfahren zur Berechnung von \sqrt{A}	36
14	Approximation eines exponentiellen Mittelwertes, also einem IIR-System, durch ein FIR-System	38
15	Berechnung und Darstellung von (4.1.1)	51
16	Modifiziertes Codeschnipsel 15 und Plot von P im Frequenzbereich.	52
17	Berechnung und Darstellung von (4.1.4)	55

18	Berechnung und Darstellung von (4.2.1)	58
19	Berechnung und Darstellung von (4.2.5)	60
20	Berechnung und Darstellung von (4.3.1)	63
21	Darstellung der DC-Sequenz und der Nyquist-Sequenz für $N = 8$	66
22	Zyklische/periodische Faltung im Zeitbereich und im Frequenzbereich.	67
23	Zero-padding im Zeitbereich einer rect-Funktion liefert eine Überabtastung der sinc-Funktion im Frequenzbereich	70
24	Einfluss der Fensterbreite bei der Analyse von harmonischen Schwingungen	71
25	Geschickte Fensterung kann den Dynamikbereich erhöhen	72
26	STFT für verschiedene Fensterlängen	73
27	Analyse einer „Bassline“ mittels STFT, die eine G-Dur-Tonleiter spielt	74
28	Vergleich verschiedener Sendesignale $x[\cdot]$ bezüglich der resultierenden Korrelation $r_{x,y}[\cdot]$. .	83
29	Simulation eines LFSR mittels binären Operationen	85
30	Simulation eines RADAR-Systems basierend auf MLBS, die von LFSR erzeugt wurden. Oben verrauschttes Empfangssignal $y[\cdot]$, mitte DFT $Y[\cdot]$, unten $r_{x,y}[\cdot]$	87
31	Implementierung von (6.2.1)	90
32	Berechnung der B-Spline Koeffizienten	92

Linkverzeichnis

1	https://erleuchtet.org/2010/07/ridiculously-large-buddhabrot.html	25
2	https://en.wikipedia.org/wiki/Hartman-Grobman_theorem	29
3	https://encyclopediaofmath.org/index.php?title=Newton_method	35
4	https://mathworld.wolfram.com/LaurentSeries.html	39
5	https://de.wikipedia.org/wiki/Cauchyscher_Integralsatz	42
6	https://de.wikipedia.org/wiki/Fundamentalsatz_der_Algebra	43
7	https://de.wikipedia.org/wiki/Satz_von_Parseval	50
8	https://en.wikipedia.org/wiki/Plancherel_theorem	54
9	https://en.wikipedia.org/wiki/Gibbs_phenomenon	59
10	https://docs.scipy.org/doc/scipy/reference/signal.windows.html	72
11	https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.windows.hann.html	72
12	https://mixbutton.com/mixing-articles/music-note-to-frequency-chart/	74
13	https://en.wikipedia.org/wiki/Crest_factor	82
14	https://de.wikipedia.org/wiki/Irreduzibles_Polynom	85
15	https://developer.nvidia.com/gpugems/gpugems2/part-iii-high-quality-rendering/chapter-20-fast-third-order-texture-filtering	94

Allgemeine Hinweise

Die Idee hinter dem Skript zur Vorlesung ist, dass es die Zuhörer der Bürde des Mitschreibens entledigt und Zeit und Platz zum Folgen der Vorlesung frei macht. Das Skript sollte deshalb immer zur Vorlesung und Übung mitgebracht und im Idealfall mit Notizen versehen werden, bzw. zum Nachschlagen verwendet werden.

Eine stetig aktualisierte Version des Skriptes findet man unter <https://github.com/SebastianSemper/lecturenotes>.

Als Begleitmaterial ist auch eine Auswahl an Codeschnipseln bereitgestellt, die einerseits in Auszügen im Skript direkt eingebunden sind, aber auch unter dem obigen Link aufzufinden sind.

Zum Ausführen der Codeschnipsel empfehlen wir ein Python-Environment^a, in welchem folgende Pakete installiert sein sollten:

- numpy – <https://numpy.org>
- scipy – <https://scipy.org>
- matplotlib – <https://matplotlib.org>
- cupy – <https://cupy.dev> (optional für ZOOM GPU speed)
- sympy – <https://sympy.org> (optional für symbolisches Rechnen in Python)

Diese können ganz einfach via

```
conda create -n dsv python=3.10
conda activate dsv
python -m pip install numpy scipy matplotlib
```

installiert werden.

Alternativ steht unter <https://jup.rz.tu-ilmenau.de/hub/login> ein Jupyter-Hub zur Verfügung, der eine Python IDE im Browser bereitstellt.

^a<https://github.com/conda-forge/miniforge>

1 Theoretische Grundlagen

Digitale Signalverarbeitung ist ein Feld, das sich vieler verschiedener mathematischer Grundlagen bedient, um die gefundenen Zusammenhänge rigoros, knapp und gleichzeitig elegant zu formulieren. Deshalb kommen wir nicht umhin, uns einiger dieser Grundlagen zu erinnern. Alles hier knapp aufgelistete sollte schon bekannt sein und dient nur als bequemes Nachschlagewerk für das kommende Semester.

1.1 Komplexe Zahlen

Die *komplexen Zahlen* \mathbb{C} sind die Menge aller $z = x + jy$, wobei $x, y \in \mathbb{R}$ und für die imaginäre Einheit j gilt, dass $j^2 = -1$. Wir nutzen hier speziell j in Abgrenzung zu i oder j , da diese oft als Indices oder Laufvariablen auftreten. Bei $z = x + jy$ nennen wir $x = \Re(z)$ den Realteil und respektive $y = \Im(z)$ den Imaginärteil. Komplexe Zahlen lassen sich auch in der Polarform $z = r \exp(j\phi)$ darstellen, wobei $r = |z| = \sqrt{\Re(z)^2 + \Im(z)^2} \geq 0$ den Betrag und $\phi = \angle(z) = \arctan(y/x)$ das Argument von z darstellen. Die zu $z = r \exp(j\phi) = x + jy$ komplex konjugierte Zahl ist $z^* = r \exp(-j\phi) = x - jy$.

Komplexe Zahlen haben viele interessante Eigenschaften und Anwendungen, vor allem in der digitalen Signalverarbeitung. Beispielsweise für die Darstellung von einem modulierten reellen Passband Signal $s : \mathbb{R} \rightarrow \mathbb{R}$ dargestellt durch

$$s(t) = x(t) \cos(\omega t) + y(t) \sin(\omega t) \in \mathbb{R},$$

die äquivalente Darstellung im komplexen Basisband

$$s_B(t) = x(t) + jy(t) \quad \text{mit} \quad \Re(s_B(t) \exp(j\omega t)) = s(t). \quad (1.1.1)$$

existiert. Man sagt auch, dass $s_B \exp(j\omega \cdot)$ das analytische Signal zu s darstellt. Dass komplexe Zahlen viele Überraschungen bereithalten sieht man wenn man simuliert für welche $c \in \mathbb{C}$ die Folge

$$z_{n+1} = z_n^2 + c$$

konvergiert oder divergiert, wenn man $z_0 = 0$ setzt (Übung).

1.2 Signale

1.2.1 Definition und Typen

Wir haben gerade schon von Signalen gesprochen, ohne sie etwas genauer einzuführen. Ganz allgemein kann man sich Signale als Objekte vorstellen, die abhängig von Raum, Zeit, oder beidem, physikalische Messgrößen, wie Spannungen, Feldstärken, oder Temperaturen modellieren/abbilden.

Die theoretische Darstellung von Signalen erfolgt durch *Funktionen*. Eine Funktion $s : D \rightarrow B$ besitzt einen Namen (s), einen Definitionsbereich D und einen Bildbereich B . Hierbei sind D und B zunächst irgendwelche Mengen. Die Funktion s bildet nun Paare (d, b) zwischen Mengenelementen von D und B , indem man schreibt $(d, s(d))$, oder $d \mapsto s(d) = b$. Der Witz ist nun, dass man ein Signal mit physikalischer Bedeutung erhält, indem man lediglich D und B geschickt wählt.

Ist $D = B = \mathbb{R}$ so sprechen wir von einem reellen Signal s und meist denken wir dabei bei D an die Zeitachse, weshalb wir auch $s \mapsto s(t)$ schreiben. Ist $D = \mathbb{R}^3$, $B = \mathbb{R}$, so denken wir meist an den dreidimensionalen Raum für den Definitionsbereich und haben als ein Signal im Raum gegeben. Ist nun

jedoch $D = \mathbb{Z}$, $B = \mathbb{R}$, so ist das Signal nur für die ganzen Zahlen \mathbb{Z} definiert, weshalb wir dann von einem Zeitdiskreten Signal sprechen. Meist schreiben wir hierfür kurz $s[k] \in \mathbb{R}, k \in \mathbb{Z}$. Man soll sich hier nicht vorstellen, dass die Werte „zwischen“ den ganzen Zahlen nur fehlen würden. So ist dies *nicht* zu verstehen. Zwischen den gegebenen Werten ist keine Information vorhanden! In manchen Situationen werden wir die diskreten Signale explizit aufschreiben wollen. In diesen Fällen markieren wir die Stelle $k = 0$ via

$$\dots, 0, 1, 2, 3, 2, 1, 0, \dots,$$

um eine bequeme Schreibweise für solche Folgen zu erhalten.

Versuchen sie für möglichst viele verschiedene Kombinationen von D und B Beispiele zu finden (Übung).

1.2.2 Signale als Vektoren

Um mit Signalen gut umgehen zu können, ist es wichtig ihre Eigenschaften als mathematische Objekte zu kennen. Intuitiv stellt man sich vor, dass man Signale in ihrer Intensität verändern können sollte, und für beliebige Änderung der Intensität wieder ein Signal erhält. Wir gehen hier zunächst der Einfachheit halber von $D = B = \mathbb{R}$ aus.

Definiert man für $a \in \mathbb{R}$ das Objekt $a \cdot s$ als $t \mapsto a \cdot s(t)$ so erhält man wieder ein Signal. Die Werte von s werden also einfach skaliert. Betrachtet man nun zwei Signale s_1, s_2 und definiert $s_1 + s_2$ als $t \mapsto s_1(t) + s_2(t)$, so erhalten wir die Summe oder die Superposition von s_1 und s_2 . Da Signale oft physikalische Messgrößen darstellen, macht dies auch oft Sinn, da in der Physik das Prinzip der Superposition oft eine Rolle spielt. Wenn wir die beiden Fakten nun kombinieren erhalten wir für $a_1, a_2 \in \mathbb{R}$ und zwei Signale s_1, s_2 , dass

$$(a_1 s_1 + a_2 s_2)(t) = a_1 s_1(t) + a_2 s_2(t)$$

wieder ein Signal repräsentiert. Objekte, die diese Eigenschaft haben, nennt man *Vektoren* und diese leben in einem *Vektorraum*.

Das mag erstmal nicht so schockieren, aber wir gewinnen dadurch *alle* Werkzeuge aus der linearen Algebra für unsere Zwecke. Beispielsweise können wir nun geschickt Bausteine für eine gewisse Untergruppe von Signalen finden, mit denen sich diese Signale gut und informativ beschreiben lassen. Beispielsweise könnten wir uns fragen, ob es für den Vektorraum der Bild-Signale eine Basis gibt, sodass für jedes Bild b eine Darstellung existiert, dass

$$b(x, y) = c_1 b_1(x, y) + c_2 b_2(x, y) + \dots,$$

gilt. Die Zahlen c_1, c_2, \dots können also das Signal b darstellen, indem man einfach die Elemente aus der Basis hennimmt, entsprechend skaliert und summiert. In gewisser Weise sind die Koeffizienten c_i das Signal b . Vielleicht gelingt es uns, die Menge $\{b_1, b_2, \dots\}$ so zu konstruieren, dass wir immer nur wenige von diesen b_i brauchen, sodass wir jedes beliebige Bild aus einer Fotokamera durch geschickte Kombination von diesen darstellen können (*sadMP3noises*).

1.2.3 Transformation von Signalen

Noch interessanter ist aber die Manipulation von Signalen durch *Transformationen*. Der Sinn von Transformationen ist es, neue oder einfache bestimmte Einsichten in ein Signal zu gewinnen. Es kann aber auch sein, dass man Operationen, die auf Signalen ausgeführt werden sollen, „einfach“ mittransformieren kann. Vielleicht

ist die gewünschte Operation nach Transformation deutlich einfacher anzuwenden? Jede Transformation liefert hierbei andere Informationen oder ist für andere Signale definiert.

Mathematisch ist eine Transformation nichts anderes als eine Abbildung zwischen Signalen. D.h. auch eine Transformation T bildet Paare zwischen Objekten aus Mengen – in diesem Fall Signalen – also $s \mapsto Ts = S$. Nach Anwendung der Transformation T auf s erhalten wir also ein anderes Signal $Ts = S$. Gerade haben wir schon festgestellt, dass man Signale beliebig skalieren und addieren kann und es als eine Art grundlegende Eigenschaft von Signalen festgehalten. Nehmen wir nun ein Signal mit Werten

$$s(t) = a_1 s_1(t) + a_2 s_2(t)$$

und wir wenden die Transformation T auf beiden Seiten der Gleichung an

$$\{Ts\}(t) = \{T(a_1 s_1 + a_2 s_2)\}(t).$$

Ist nun die Transformation so, dass wir schreiben können

$$\{Ts\}(t) = \{T(a_1 s_1 + a_2 s_2)\}(t) = a_1 \{Ts_1\}(t) + a_2 \{Ts_2\}(t),$$

so nennen wir T eine *lineare* Transformation. Zusammen mit der Superpositionseigenschaft von Signalen sieht man nun, warum Linearität so wichtig für Transformationen ist, weil es einfach zur Vektorraumstruktur von Signalen passt. Die Linearität erlaubt es uns beispielsweise auch das obige Signal b ganz einfach zu transformieren. Nehmen wir es in seiner Darstellung als

$$b(x, y) = c_1 b_1(x, y) + c_2 b_2(x, y) + \dots,$$

und wir haben eine beliebige lineare Transformation T , deren Effekt wir auf b angewendet sehen wollen. Wir suchen also $\{Tb\}(x, y)$. Aber das ist mit der Linearität ganz einfach. Wir müssen nur Tb_i kennen, also die Wirkung von T auf die Basisvektoren b_i , denn

$$\{Tb\}(x, y) = c_1 \{Tb_1\}(x, y) + c_2 \{Tb_2\}(x, y) + \dots,$$

ist eine valide Darstellung von Tb . Cool!

Beispiele für solche linearen Transformationen sind Differentiation (falls möglich), bilden der Stammfunktion (falls möglich), Verzögerung eines Zeitsignals um Zeit $a \in \mathbb{R}$, Stauchung und Streckung eines Zeitsignals, Rotation eines Bildes, die Fourier-Transformation, die diskrete Fourier-Transformation, zyklische Faltung, oder Korrelation mit einem anderen Signal p . Gegenbeispiele sind $p(t) = \sin(s(t))$, oder $p(t) = (s(t))^\alpha$ für $\alpha \neq 1$.

Man sieht, dass viele wichtige Operationen lineare Transformationen darstellen und wir haben mit linearer Algebra ein mächtiges Tool an unserer Seite, um mit ihnen umzugehen.

1.2.4 Zufällige Signale

Man kann auch noch eine weitere Sichtweise auf Signale haben. In manchen Fällen ist es nicht zweckmäßig, dass man ein Signal s als vollständig bekannte und fixe Funktion modelliert. Stattdessen modelliert man die Werte $s(t)$ des Signals s an den Stellen t als *Zufallsgröße*. Das heißt, dass die Werte $s(t)$ einer Verteilung $X(t)$ folgen. An jedem Zeitpunkt t „hängt“ eine solche Verteilung, die bestimmt mit welcher Wahrscheinlichkeit die Werte $s(t)$ in einem gewissen Intervall liegen. Man spricht in diesem Fall auch von *stochastischen* Signalen, im Gegensatz zu den obigen *deterministischen* Signalen.

Es kann verschiedene Gründe haben, dass man ein Signal nicht mehr deterministisch beschreiben kann/will/sollte:

- Sobald die Werte von s durch eine Messung entstanden sind, enthalten diese normalerweise Messrauschen. Dann modelliert man s meistens als Summe

$$s(t) = x(t) + n(t),$$

wobei $n(t) \sim \mathcal{N}(0, \sigma^2(t))$ meist als eine Realisierung einer mittelwertfreien Normalverteilung mit Varianz $\sigma^2(t)$ angenommen wird, und x als ein deterministisches Signal.

- Wenn man generell nicht genug Information über das Signal hat, beispielsweise, kennt man nur dessen Verteilung im Frequenzbereich, also Wahrscheinlichkeiten, dass gewisse Frequenzen vorhanden sind, oder nicht. Dennoch ist man natürlich an dem Verhalten des Signals im Zeitbereich interessiert.
- Wenn es für die Anwendung nicht notwendig ist. Dies kann der Fall sein, wenn man einen Filter entwickelt, der eine gewisse Klasse von Signalen als Eingang bekommt, kann es reichen die Verteilung der Signale zu kennen und dann den Ausgang des Filters nur stochastisch zu beschreiben.

„In 99.99 % der Fälle ist der nachgeschaltete Verstärker nicht übersteuert.“

Für solche Aussagen ist es sogar *notwendig* die Verteilung der Eingangssignale zu kennen, ansonsten ist so eine Aussage gar nicht möglich, da man eben keine Verteilung für ein deterministisches Signal angeben kann.

Um stochastiche Signale korrekt handhaben zu können, ist einige Mathematik notwendig, die wir einfach übergehen und stattdessen versuchen ein *intuitives* Verständnis zu entwickeln.

1.2.5 Spezielle Signale

Uns werden immer wieder einige spezielle Signale begegnen, die wir hier kurz auflisten wollen.

- Die *Delta-Funktion (Dirac- δ)* als Funktional δ , das angewendet auf ein Signal s , liefert, dass $\delta(s) = s(t=0)$. Visualisiert wird dieses nicht-Signal, durch einen Impuls der Höhe 1 bei $t=0$. Es ist nicht ohne Ironie, dass eines der wichtigsten Objekte der Signalverarbeitung selbst kein Signal ist, wie eines behandelt wird, aber immer mit Vorsicht.
- Die *Heavyside-Funktion $u : \mathbb{R} \rightarrow \mathbb{R}$* mit

$$u(t) = \begin{cases} 1 & \text{für } t > 0 \\ \frac{1}{2} & \text{für } t = 0 \\ 0 & \text{für } t < 0. \end{cases}$$

Man kann δ als distributionelle Ableitung von u auffassen.

- Die *komplexe Schwingung $s : \mathbb{R} \rightarrow \mathbb{C}$* bei Frequenz $f > 0$ ist definiert als $s(t) = \exp(jft)$ und wir uns im Verlauf des Semesters noch einige Male begegnen. Beispielsweise gilt $s^*(t) = s(-t)$.
- Der *diskrete δ -Stoß $\delta[k]$* ist definiert als

$$\dots, 0, \underset{\uparrow}{1}, 0, \dots$$

- Endliche, diskrete Signale können wir entweder durch

$$s = [0, 1, 2, \underset{\uparrow}{3}, 2, 1, 0]$$

darstellen, oder als endliche Summe von einigen diskreten δ -Stößen:

$$s[n] = \sum_{k=-2}^{k=+2} s[k] \delta[n - k]$$

1.2.6 Beispiel: LTI-Systeme

Wir werden uns zwar noch später ausführlich mit **Linear Time-Invariant (LTI)** Systemen beschäftigen, doch sie sollen hier schon als nicht-triviales Beispiel dienen. Wir sind also mit einem System \mathcal{H} konfrontiert, das einerseits die Eigenschaft hat, dass für Anregungen $x : \mathbb{R} \rightarrow \mathbb{R}$ eine Verschiebungsinvarianz mit $y(t - \tau) = (\mathcal{H}x(\cdot - \tau))(t)$ gilt. Außerdem ist \mathcal{H} linear.

Dann kann man die Wirkung von \mathcal{H} auch durch Faltung mit der sog. Impulsantwort h des Systems darstellen, also

$$y(t) = (\mathcal{H}x)(t) = \int_{-\infty}^{+\infty} x(t - \tau)h(\tau) d\tau = (x * h)(t), \quad (1.2.1)$$

wobei $h = \mathcal{H}\delta$, also die Reaktion des Systems auf einen Dirac-Stoß darstellt. An dieser Darstellung sieht man sehr gut, dass \mathcal{H} linear ist, weil die Integration linear in x ist.

Natürlich ist der Zeitbereich für diese Art von System nicht der richtige Anschauungsort. Nach Laplace-Transformation von y zu $Y = \mathcal{L}y$ sehen wir, dass wir stattdessen

$$Y(s) = X(s) \cdot H(s),$$

schreiben können. Hierbei sind $X = \mathcal{L}x$ und $H = \mathcal{L}h$ die Laplace-Transformationen des Eingangs und der Impulsantwort h . Nicht nur hat sich die „Berechnung“ von Y vereinfacht, sondern wir haben auch ein besseres Gefühl für das Verhalten des Systems in Abhängigkeit von h , bzw. H , weil der Einfluss einfach multiplikativ ist.

Wir können die lineare Algebra noch ein wenig weiter treiben. Betrachten wir als Eingang die Funktion $x_s(t) = \exp(st)$ für ein beliebiges $s \in \mathbb{C}$. Dann rechnen wir einfach mit (1.2.1) nach, dass

$$(H \exp(s \cdot))(t) = \int_{-\infty}^{+\infty} \exp(s(t - \tau))h(\tau) d\tau = \exp(st) \int_{-\infty}^{+\infty} \exp(-s\tau)h(\tau) d\tau = \exp(st)H(s),$$

gilt. Das heißt, dass die Funktionen $\exp(s \cdot)$ die *Eigenvektoren* des Operators \mathcal{H} , weil gilt

$$(\mathcal{H}x_s)(t) = x_s(t) \cdot H(s),$$

wobei H die Laplace-Transformation von h ist. Das heißt auch, dass $H(s)$ die zugehörigen *Eigenwerte* sind. Wir sehen hier also, dass Signale *wirklich* wie Vektoren funktionieren können und es sich im Fall von linearen System förmlich aufzwingt, da die Linearität des Systems zur linearen Vektorraumstruktur „passt“.

2 Abtastung von Signalen

Als ersten Schritt der digitalen Signalverarbeitung wollen wir uns den Übergang von einem analogen Signal zu einem digitalen näher ansehen. Intuitiv können wir diesen Vorgang in vielen Anwendungen beobachten. Wir nehmen im Tonstudio mit einem Mikrofon Ton auf und eine Soundkarte wandelt das analoge Signal in einen WAV-Datenstrom um. In einer Fotokamera, trifft ein Feld von Lichtstrahlen ein und wird von einem **Complementary Metal Oxide Semiconductor (CMOS)**-Sensor „direkt“ abgetastet und in Helligkeitswerte pro Farbkanal umgewandelt. Eine Antenne wandelt ein anliegendes elektro-magnetisches Feld in eine Spannung um, welche nachträglich von einem **Analog-to-Digital Converter (ADC)** abgetastet und quantisiert wird.

Mathematisch modellieren wir analoge Signale $s_a : D \rightarrow B$ meist mit D und B , die auf die reellen Zahlen \mathbb{R} zurückgreifen. Die Wandlung von analog zu digital transformiert dieses Signal in eine Funktion $s : \mathbb{Z} \rightarrow Q$ um, wobei auch $|Q| < \infty$ gilt. Das heißt, dass das Signal nach AD-Wandlung nur noch endliche Werte annehmen kann und, dass es nur noch aus einer *Folge* von Werten aus der Menge Q besteht. Es wurde also zeit- und wertdiskretisiert. Wir werden uns zunächst nur mit der Diskretisierung in Zeit befassen, weil es einfacher ist. Das heißt, dass wir uns vorstellen, dass das diskretisierte Signal nur an einer diskreten Menge an Punkten noch Informationen über das abgetastete Signal beinhaltet. Weiterhin sind wir nicht an der physikalischen Umsetzung von **ADCs** interessiert, sondern höchstens an deren systemtheoretischer Modellierung.

Die zentralen Fragen sind nun:

- Wie muss der Vorgang der Abtastung gestaltet sein, dass keine Information verloren geht?
- Wie können wir die Eigenschaften des analogen Signals in dessen abgetasteter Version wiederfinden?
- Wie können wir eine Intuition für den Vorgang der Abtastung entwickeln?

2.1 Frequenz von Signalen

2.1.1 Zeit-Kontinuierliche Sinussignale

Meistens werden wir uns in der Vorlesung mit reell- oder komplexwertigen Zeitsignalen befassen, d.h. wir modellieren unsere Signale als $x_a : \mathbb{R} \rightarrow \mathbb{R}$ oder $x_a : \mathbb{R} \rightarrow \mathbb{C}$. Wobei physikalische Signale natürlich nur reellwertig sind, doch manchmal ist die Darstellung als komplexwertige Funktion besser handhabbar, siehe (1.1.1). Das heißt, dass die Abtastung im Zeitbereich vonstatten geht, was sofort den Begriff der *Frequenz* auf den Plan ruft, da Frequenz mit Einheit $1/\text{s}$ eng mit Zeit s verknüpft ist.

Betrachten wir also erst einmal, welchen Einfluss Abtastung von Signalen mit einzelnen Frequenzen hat, am Beispiel von

$$x_a(t) = A \cos(\Omega t + \theta), \quad (2.1.1)$$

wobei wir hier $A \in \mathbb{R}$ als Amplitude, $\Omega \in \mathbb{R}_0^+$ als Kreisfrequenz 1 rad s^{-1} , $t \in \mathbb{R}$ als Zeit 1 s und die Phase $\theta \in \mathbb{R}$ mit Einheit 1 rad nutzen. Alternativ können wir auch zur Frequenz $F \in \mathbb{R} 1 \text{ s}^{-1} = 1 \text{ Hz}$ übergehen. Dann erhalten wir

$$x_a(t) = A \cos(2\pi F t + \theta).$$

Diese Funktion ist in Abb. 1 dargestellt. Wir sehen, dass die Funktion periodisch ist mit Periode $T_p = 1/F$.

Das heißt, dass $x_a(t + k \cdot T_p)$ für $k \in \mathbb{Z}$ nicht vom Signal $x_a(t)$ zu unterscheiden ist!

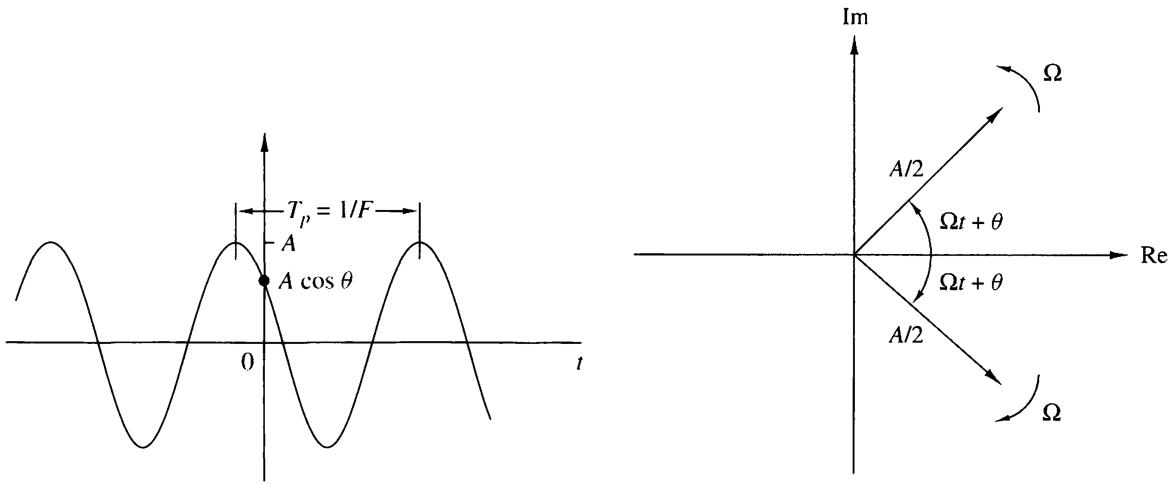


Abbildung 1: $x_a(t) = A \cos(2\pi F t + \theta)$, Quelle: [1]

Es gibt noch eine alternative Darstellung von der obigen Funktion durch die Addition von zwei *Phasoren* als

$$x_a(t) = A \cos(2\pi F t + \theta) = \frac{A}{2} \exp(j(\Omega t + \theta)) + \frac{A}{2} \exp(-j(\Omega t + \theta)).$$

Da die beiden überlagerten Phasoren so interpretiert werden können als rotierten diese in gegensätzliche Richtungen, ist es gerechtfertigt der physikalischen Intuition entgegen auch von „negativen“ Frequenzen zu sprechen. Wir erlauben also $F \in \mathbb{R}$, womit auch der Spezialfall $T_p = \infty$, also $x_a(t) = A$ abgedeckt ist. Ein kleines Beispiel findet man in Codeschnipsel 1.

```

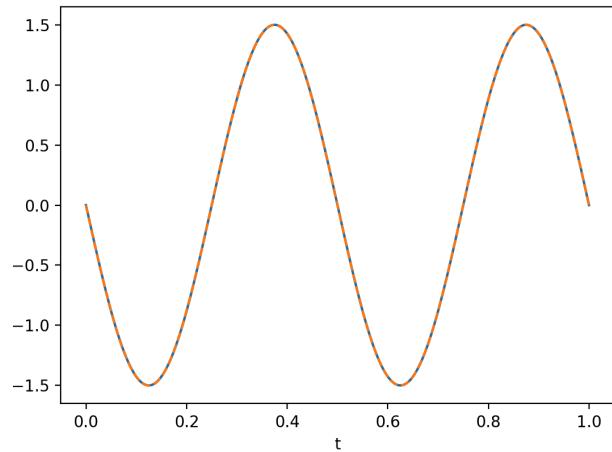
A = 1.5
F = 2
theta = np.pi / 2

def harm(t: float) -> float:
    return A * np.cos(
        2 * np.pi * (F * t + theta))

def phasor(t: float) -> complex:
    return 0.5 * A * np.exp(
        -1j * (2 * np.pi * (F * t + theta)))

T = np.linspace(0, 1, 255)
plt.plot(T, harm(T), label="harm(T)")
plt.plot(T, phasor(T) + phasor(T).conj(),
         linestyle="--", label="harm(T)")
plt.xlabel("t")
plt.show()

```



Codeschnipsel 1: Berechnung und Darstellung von (2.1.1), siehe [dsv/code/cont_harms.py](#)

2.1.2 Zeit-Diskrete Sinussignale

Als nächstes gehen wir zu dem eigentlich interessanten Fall über, bei welchem wir von zeitdiskreten harmonischen Signalen sprechen. Dabei gehen wir vorerst *nicht* davon aus, dass das Signal durch Abtastung eines analogen Signals entstanden ist, sondern betrachten es ganz losgelöst für sich. In Analogie zu (2.1.1) definieren wir

$$x[n] = A \cos(\omega n + \theta) = A \cos(2\pi f n + \theta). \quad (2.1.2)$$

Wichtig bei diskreten Signalen ist, dass ihre physikalische Interpretierbarkeit nicht direkt gegeben ist, da $n \in \mathbb{Z}$ nur die diskreten Werte „nummeriert“, also *einheitenlos* ist. Deshalb hat $f \in \mathbb{R}$ lediglich als Einheit „Zyklen pro Sample“, was man auch daran sieht, dass für $f = 1$ gilt $x[n] = A \cos(2\pi n + \theta) = A \cos(\theta)$. Es existiert nun ein wichtiger Unterschied zwischen $x[\cdot]$ von (2.1.2) und $x_a(\cdot)$ von (2.1.1). Das Signal $x[\cdot]$ ist nur periodisch, falls f eine rationale Zahl ist, also $f = p/q$ für $p, q \in \mathbb{Z}$ und $q \neq 0$.

Wieso?

Ein zeitdiskrtes Signal ist periodisch, falls $x[n+N] = x[n]$ für alle $n \in \mathbb{Z}$. Für unser Signal in (2.1.2) heißt das also, dass

$$\cos(2\pi f n + \theta) = \cos(2\pi f(n+N) + \theta) = \cos(2\pi f n + 2\pi f N + \theta)$$

Da \cos Periode $2\pi k$ für $k \in \mathbb{Z}$ besitzt, muss $2\pi f N = 2\pi k$ gelten, also

$$f = \frac{k}{N}.$$

Andersherum kann man die kleinste Periode N ermitteln, indem man $f = k/N$ vollständig kürzt, sodass also Zähler und Nenner keine gemeinsamen Teiler mehr haben, und man dann den Nenner des resultierenden Bruches als N setzt. Ein Beispiel wird in Codeschnipsel 2 gezeigt. Man kann interessante Ergebnisse erzielen, wenn man diesen Plot für $\omega = 0, \pi/8, \pi/4, \pi/2$ und $\omega = \pi$ erzeugt (Übung).

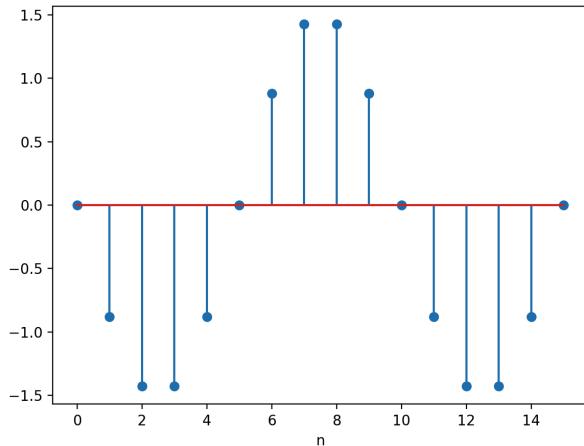
```

A = 1.5
f = 0.1
theta = 0.25

def harm(n: int) -> float:
    return A * np.cos(
        2 * np.pi * (f * n + theta))

n = np.arange(16)
plt.stem(n, harm(n), label="harm[n]")
plt.xlabel("n")
plt.show()

```



Codeschnipsel 2: Berechnung und Darstellung von (2.1.2), siehe [dsv/code/disc_harms.py](#)

Die Signale der Form (2.1.2) haben noch eine andere interessante Eigenschaft, die sich wieder aus der 2π -Periodizität von \cos ergibt. Betrachten wir noch einmal (2.1.2) und wir finden, dass

$$\cos(\omega n + \theta) = \cos(\omega n + 2\pi n + \theta) = \cos((\omega + 2\pi)n + \theta).$$

Das heißt, dass

$$x[n] = \cos(\omega n + \theta) = \cos((\omega + 2\pi k)n + \theta) = x_k[n]$$

für alle $k \in \mathbb{Z}$ gilt. Das heißt, dass sich $x_k[\cdot]$ nicht von $x[\cdot]$ unterscheiden lässt. Man nennt dann jedes der $x_k[\cdot]$ einen *Alias* von $x[\cdot]$. Man kann deshalb auch sagen, dass für jedes ω mit $|\omega| > \pi$ ein zugehöriges ω_a mit $|\omega_a| < \pi$ existiert, sodass

$$\cos(\omega n + \theta) = \cos(\omega_a n + \theta)$$

gilt. Vergewissern sie sich von dieser Tatsache, indem sie verschiedene Aliase basierend auf Codeschnipsel 2 visualisieren (Übung).

Stellen wir uns für einen kurzen Moment vor, dass wir wissen, dass wir $x[n]$ durch Abtastung einer Funktion wie in (2.1.1) erhalten haben. Selbst wenn wir wissen, dass nur *eine* Frequenz in diesem Signal vor Abtastung vorhanden war, können wir *nicht* entscheiden, welche das war.

2.2 Zeit-Kontinuierliche Komplexe Sinussignale

Wir wollen eine bestimmte Menge an Funktionen betrachten. Wir wollen kontinuierliche komplexe Schwingungen betrachten, welche mit einer Frequenz F_k schwingen, welche ein ganzzahliges Vielfaches einer Frequenz F_0 ist. Das heißt, wir betrachten dann

$$F_k = k \cdot F_0 \quad \text{für } k \in \mathbb{Z} \quad , \text{was} \quad x_k(t) = \exp(j2\pi F_k t) = \exp(j2\pi k F_0 t)$$

ergibt. Jedes der x_k hat Periode $1/F_k = T_k = T_0/k$. Das heißt für wachsendes $|k|$ werden die Perioden immer um ein Vielfaches kürzer. Umgekehrt haben dann alle x_k gemeinsame Periode, T_0 , da für jedes k gilt, dass $T_k \cdot k = T_0$. Wir haben auch kein Problem mit Aliasing zwischen den x_k , da bei kontinuierlichen Signalen gilt, dass $x_{k_1} \neq x_{k_2}$, falls $k_1 \neq k_2$.

Wie wir in Abschnitt 1.2.2 gesehen haben, können wir beliebige Linearkombination aus Signalen bilden und erhalten wieder ein Signal. Wir können also für eine Folge von $c_k \in \mathbb{C}$ die Linearkombination der x_k bilden und erhalten

$$x_a = \sum_{k \in \mathbb{Z}} c_k x_k : \mathbb{R} \rightarrow \mathbb{C} \quad \text{mit} \quad t \mapsto x_a(t) = \sum_{k \in \mathbb{Z}} c_k x_k(t) = \sum_{k \in \mathbb{Z}} c_k \exp(j2\pi k F_0 t). \quad (2.2.1)$$

Die erste Schreibweise ist absichtlich ohne das Argument t , um zu verdeutlichen, dass Signale *wirklich* als eigenständige Signale behandelt werden können und dass $x_a(t) \in \mathbb{C}$ „nur“ die Auswertung von x_a an der Stelle t ist, welche *strikt* von dem Vektor x_a zu unterscheiden ist.

Natürlich ist (2.2.1) als Fourier-Reihe von x_a bekannt, und die c_k sind die Fourierkoeffizienten von x_a . Wie in Abschnitt 1.2.2 können wir also die c_k durch (2.2.1) mit x_a *identifizieren*, da uns die c_k eine alternative Darstellung von x_a liefern.

2.3 Zeit-Diskrete Komplexe Sinussignale

Analog zu Abschnitt 2.2, wollen wir uns zeit-diskrete komplexe Schwingungen herleiten, die von einer bestimmten Grundfrequenz definiert werden. Da wir, im Unterschied zum kontinuierlichen Fall, nicht für alle $f \in \mathbb{R}$ eine periodische Funktion erhalten, wählen wir $f_0 = 1/N$ für ein $N \in \mathbb{N}$. Die Intension ist, dass wir so $1/N$ Perioden pro Abtastwert erhalten werden. Demzufolge wird die Schwingung mit Frequenz f_0 genau Periodenlänge N haben. Dann definieren wir die Signale $x_k[\cdot] : \mathbb{N} \rightarrow \mathbb{C}$ als

$$x_k[n] = \exp(j2\pi k f_0 n) \quad \text{mit} \quad k \in \mathbb{Z}. \quad (2.3.1)$$

Man sieht nun leicht, dass $f_k = k/N$ immer eine rationale Zahl ist und $x_k[\cdot]$ Periodenlänge k/N hat, falls $k/N \in \mathbb{Z}$. Dies ist in Codeschnipsel 3 dargestellt.

Außerdem findet man wieder, dass $x_k[\cdot]$ ein Alias von $x_{k+N}[\cdot]$ sein muss, denn mit $f_0 = 1/N$ ergibt sich

$$x_{k+N}[n] = \exp(j2\pi(k+N)f_0 n) = \exp\left(j2\pi\frac{k+N}{N}n\right) = \exp\left(j2\pi\frac{k}{N}n\right) \exp\left(j2\pi\frac{N}{N}n\right) = x_k[n].$$

Das heißt, dass nur N verschiedene $x_k[\cdot]$ existieren. Normalerweise nimmt man jene $x_k[\cdot]$ für $k = 0, 1, \dots, N-1$.

Nun können wir auch wieder, wie in (2.2.1) eine Linearkombination der $x_k[\cdot]$ bilden. Man beachte, dass in diesem Fall die Summation natürlich *endlich* sein wird, da wir nur N verschiedene $x_k[\cdot]$ zur Verfügung haben. Wir bilden also

$$x[\cdot] = \sum_{k=0}^{N-1} c_k x_k[\cdot]$$

und erhalten so ein Signal mit Werten

$$x[n] = \sum_{k=0}^{N-1} c_k x_k[n] = \sum_{k=0}^{N-1} c_k \exp\left(j2\pi\frac{k}{N}n\right), \quad (2.3.2)$$

was die Fourier-Reihe eines diskreten und periodischen Signals darstellt, wobei in diesem Fall der Vektor $\mathbf{c} = [c_k]_{k=0}^{N-1} \in \mathbb{C}^N$ eine alternative Repräsentation des Signals ist.

Das Signal $x[\cdot]$ selbst ist periodisch mit Periodenlänge N , d.h. das Signal ist durch die Werte $\mathbf{x} = [x[n]]_{n=0}^{N-1} \in \mathbb{C}^N$ *vollständig* definiert. Das heißt wir können das Signal $x[\cdot]$ mit dem *endlich-dimensionalen*

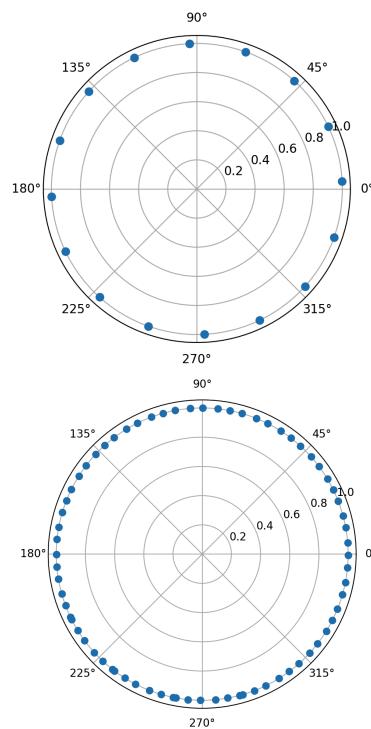
```

A = 1
f = 1 / 16
# f = 1 / (5 * np.exp(1))
theta = np.pi / 2
# N = 32
N = 720

def harm(
    n: np.ndarray[int],
) -> np.ndarray[complex]:
    return A * np.exp(
        1j * 2 * np.pi * (f * n + theta)
    )

n = np.arange(N)
x_n = harm(n)
fig, ax = plt.subplots(
    subplot_kw={"projection": "polar"}
)
ax.plot(
    np.angle(x_n),
    np.abs(x_n),
    marker="o",
    linewidth=0,
)
plt.show()

```



Codeschnipsel 3: Berechnung und Darstellung von (2.3.1) für $f_0 = 1/16$ und $f_0 = 1/(5e)$, siehe [dsv/code/disc_harms_comp.py](#)

Vektor $\mathbf{x} \in \mathbb{C}^N$ identifizieren. Da genauso jedes der $x_k[\cdot]$ auch Periodenlänge N hat, erhalten wir auf die gleiche Weise Vektoren $\mathbf{x}_k \in \mathbb{C}^N$. Mit diesen endlich-dimensionalen Vektoren sind wir nun in der Lage die Fourier-Reihe in (2.3.2) mit „normalen“ Vektoren durch

$$\mathbf{x} = c_1 \mathbf{x}_1 + c_2 \mathbf{x}_2 + \dots + c_{N-1} \mathbf{x}_{N-1} \quad (2.3.3)$$

auszudrücken. Wir sehen also *direkt*, dass Signale wirklich wie Vektoren behandelt werden können.

Gleichung (2.3.2) kann auch als Abbildung $M : \mathbb{C}^N \rightarrow \mathbb{C}^N$ interpretiert werden, da wir in die rechte Seite von (2.3.2) einfach ein $\mathbf{c} \in \mathbb{C}^N$ stecken können und wir erhalten das entsprechende $\mathbf{x} = M(\mathbf{c})$. Noch weiter ist die Abbildung M sogar linear, da

$$M(\mathbf{c}^1 + \mathbf{c}^2) = \sum_{k=0}^{N-1} (c_k^1 + c_k^2) \exp\left(j2\pi \frac{k}{N} n\right) = \sum_{k=0}^{N-1} c_k^1 \mathbf{x}_k + \sum_{k=0}^{N-1} c_k^2 \mathbf{x}_k = M(\mathbf{c}^1) + M(\mathbf{c}^2).$$

Das heißt, dass es auch eine *Matrix* $M \in \mathbb{C}^{N \times N}$ geben muss, welche uns einfach \mathbf{c} in \mathbf{x} umtransformiert, indem wir $\mathbf{x} = M \cdot \mathbf{c}$ als Matrix-Vektor-Produkt berechnen. Wenn wir (2.3.3) genau betrachten sehen wir, dass wir die Matrix M bilden können, indem wir deren k -te Spalte $M_{:,k}$ gleich \mathbf{x}_k setzen. Wenn wir noch sicher sein könnten, dass M invertierbar ist, könnten wir sogar aus \mathbf{x} via $\mathbf{c} = M^{-1} \cdot \mathbf{x}$ die Fourierkoeffizienten \mathbf{c} direkt aus einem gegebenen N -periodischen Signal $x[\cdot]$ bestimmen.

2.4 Abtastung von Sinussignalen

Es gibt viele Möglichkeiten ein analoges Signal zu digitalisieren. Wir beschränken uns auf Abtastung, welche ein analoges Signal auf eine regelmäßige Art und Weise *direkt* auswertet. Diese Art wird manchmal auch „Nyquist-Sampling“ genannt, weil die theoretische Grundlage für „erfolgreiches“ Sampling durch das Nyquist-Theorem gelegt ist. Wir stellen uns Abtastung so vor, dass wir das Signal $x_a : \mathbb{R} \rightarrow \mathbb{R}$ direkt an gewissen Stellen beobachten können. Wir können uns eine Art Sampling-Operator \mathcal{S} vorstellen, der ein analoges Signal x_a in eine abgetastete Version $x_a \mapsto \mathcal{S}(x_a)[\cdot] = x[\cdot]$ transformiert. Durch die regelmäßige/uniforme Abtastung in Zeitabständen $T > 0$ von x_a erhalten wir also

$$x[n] = \mathcal{S}(x_a)[n] = x_a(nT) \quad \text{mit } n \in \mathbb{Z}.$$

Wir nennen $F_s = T^{-1}$ die Sampling-Frequenz, oder die Abtastrate. Der Vorgang ist schematisch in Abbildung 2 dargestellt.

Da wir uns $x[\cdot]$ so vorstellen, dass es „einfach“ eine Folge von reellen Zahlen ist, müssen wir bei der Interpretation von $x[\cdot]$ auch immer gleichzeitig im Hinterkopf behalten, dass der Wert $x[n]$ dem Wert $x_a(nT) = x_a(n/F_s)$ entspricht. Das bedeutet, dass t (als Argument von x_a) und n (als Argument von $x[\cdot]$) durch

$$t = nT = n/F_s$$

miteinander in Verbindung stehen. Man sieht auch nun eindrucksvoll, dass dadurch $n = t \cdot F_s$ einheitenlos geworden ist, bzw. sein muss.

Weiterhin folgt aus $t = n/F_s$, dass es auch einen Zusammenhang zwischen der Frequenz F einer kontinuierlichen Signals und der Frequenz f im Zeit-diskreten geben muss. Um diesen herzuleiten, betrachten wir eine einfache analoge Schwingung, wie in (2.1.1), also

$$x_a(t) = A \cos(2\pi F t + \theta)$$

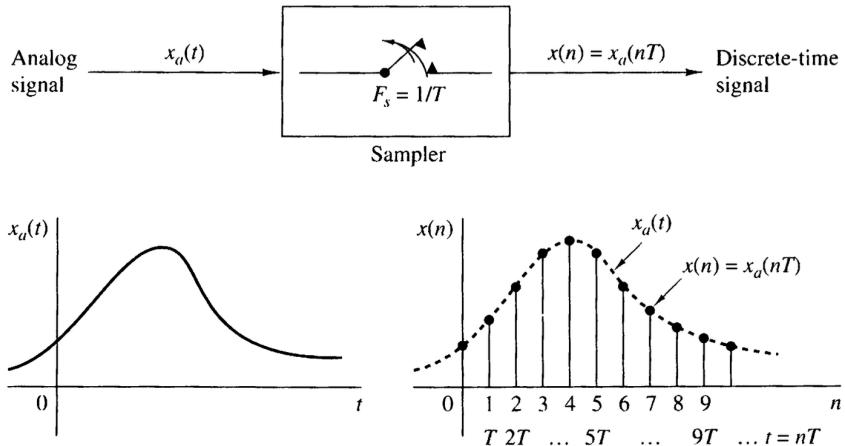


Abbildung 2: Uniforme Abtastung einer Signals. Quelle: [1]

und wir stellen uns vor, dass wir dieses Signal mit Samplerate $F_s = 1/T$ abtasten. Dann erhalten wir zunächst

$$x[n] = x_a(nT) = A \cos(2\pi F n T + \theta) = A \cos\left(\frac{2\pi F n}{F_s} + \theta\right).$$

Wenn wir nun $x[n]$ in die Form von (2.1.2) bringen, sehen wir, dass $f = F/F_s$ gelten muss.

Wie ist dies zu interpretieren? Wir sind mit dem analogen Signal x_a gestartet, welches die Frequenz F enthält. Durch das Sampling mit Rate F_s entsteht eine abgetastete Schwingung mit Frequenz $f = F/F_s$. Doch wir haben bereits gesehen, dass sich die Frequenz f von keiner Frequenzen $f + k$ unterscheiden lässt. Das heißt im Umkehrschluss, dass sich *nach* der Abtastung die ursprüngliche Frequenz nicht eindeutig bestimmen lässt, da alle

$$F_k = F + k \cdot F_s$$

bei Abtastung von $A \cos(2\pi F_k t + \theta)$ dieselben Abtastwerte $x[n]$ ergeben würden. Wenn wir nun also behaupten wollen, dass wir im digitalen irgendetwas sinnvolles zu tun gedenken, dann können wir dies gerade nicht so ohne weiteres. Denn bis jetzt haben wir keine Möglichkeit das wahre analoge Signal zu rekonstruieren. Diese missliche Lage wird in Codeschnipsel 4 dargestellt, wo ein mögliches x_a , dessen Abtastwerte $x[n]$ und zwei mögliche Aliase dargestellt sind. Man sieht, wie die Aliase so geschaffen sind, dass auch sie Ursprung für die Abtastwerte $x[n]$ sein könnten.

Die Frage ist nun, wie wir das Sampling gestalten müssen, dass wir aus $x[n]$ eindeutig das Signal x_a rekonstruieren können. In diesem Fall ist mit „rekonstruieren“ gemeint, dass wir aus den Werten $x[n]$ den Wert $x_a(t)$ für beliebige $t \in \mathbb{R}$ korrekt bestimmen können. Wie wir oben gesehen haben, ist im Digitalen nur sinnvoll von Frequenzen $f \in [-1/2, +1/2]$ zu sprechen, da wir für alle anderen $f' \notin [-1/2, +1/2]$ ein $f \in [-1/2, +1/2]$ finden, das dieselben Werte in (2.1.2) produziert. Wegen des Zusammenhangs $f = F/F_s$ macht es also Sinn sich auch im *Analogen* auf den entsprechenden Bereich zu beschränken. Das heißt, wenn wir nur analoge Signale betrachten, bei welchen

$$-\frac{F_s}{2} \leq F \leq +\frac{F_s}{2}$$

```

theta = -np.pi / 2 # rad
F = 1.5 # Hz
F_s = 1.8 # Hz
T_s = 1.0 / F_s # s

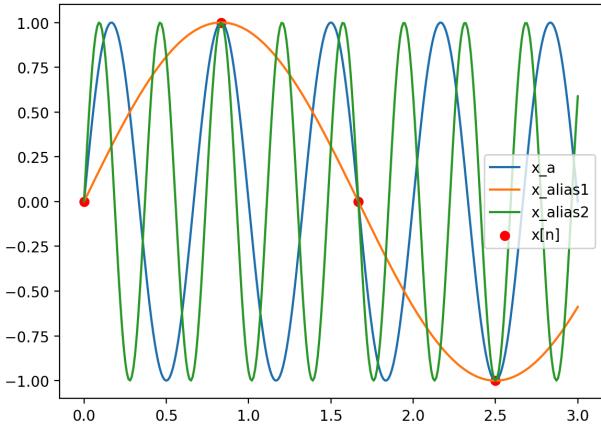
F_alias1 = F - 1 * F_s # Hz
F_alias2 = F + 1 * F_s # Hz

def x_a(t, F, theta):
    return np.cos(2 * np.pi * F * t + theta)

T = np.linspace(0, 3, 301, endpoint=True) # s
n = np.arange(4) * T_s # none

plt.plot(T, x_a(T, F, theta), label="x_a")
plt.plot(
    T, x_a(T, F_alias1, theta), label="x_alias1"
)
plt.plot(
    T, x_a(T, F_alias2, theta), label="x_alias2"
)
plt.scatter(
    n, x_a(n, F, theta), color="r", label="x[n]"
)
plt.legend()
plt.show()

```



Codeschnipsel 4: Visualisierung von $F_k = F + k \cdot F_s$, siehe [dsv/code/aliasing.py](#)

gilt, dann sind wir in der Lage aus $x[\cdot]$ das originale x_a zu rekonstruieren, weil wir wissen, auf welche der Aliase wir uns beschränken müssen.

Eine Möglichkeit der perfekten Rekonstruktion von analogen mit einzelnen Frequenzen Signalen aus uniformen digitalen Abtastwerten ist die Einschränkung auf einen bestimmten Frequenzbereich.

Umgekehrt können wir auch bei Vorwissen über die maximale Frequenz F_{\max} , also $F \in [-F_{\max}, +F_{\max}]$ in einem Signal x_a die Samplingrate ermitteln, sodass im Digitalen die Aliase $f + k$ für $k \neq 0$ nicht im Bereich $[-F_{\max}, +F_{\max}]$ liegen. Damit

$$-\frac{1}{2} \leq f \leq +\frac{1}{2}$$

gilt, muss also auch

$$-\frac{1}{2} \leq \frac{F}{F_s} \leq +\frac{1}{2} \quad \text{für alle } F \in [-F_{\max}, +F_{\max}]$$

gelten. Deshalb muss schlussendlich $F_s > 2F_{\max}$ gelten. Es ist zu beachten, dass wir diese Überlegungen erst einmal „nur“ für Signale der Form (2.1.1) und (2.1.2) durchgeführt haben. Im nächsten Abschnitt werden wir das Abtasttheorem für beliebige aperiodische Signale herleiten.

2.5 Das Samplingtheorem

Wir beschäftigen uns nun also allgemein mit dem Problem der Abtastung einer beliebigen analogen aperiodischen Signals x_a zur Folge

$$x[n] = x_a(nT).$$

Für die bequeme Analyse von kontinuierlichen Signalen und ihrem Anteil an harmonischen Komponenten, benutzen wir die **Fourier Transform (FT)** \mathcal{F} , welche ein Signal $x : \mathbb{R} \rightarrow \mathbb{C}$ transformiert in $x \mapsto \mathcal{F}x = X : \mathbb{R} \rightarrow \mathbb{C}$, wobei X durch

$$X(F) = \int_{-\infty}^{+\infty} x(t) \exp(-j2\pi Ft) dt \tag{2.5.1}$$

definiert ist. Wir werden versuchen, die Kombination von Klein- und Großbuchstaben $x \leftrightarrow X$ als **FT-Paar** beizubehalten. Die zugehörige inverse **FT**, also die Abbildung von X auf x ist gegeben durch

$$x(t) = \int_{-\infty}^{+\infty} X(F) \exp(j2\pi Ft) dF. \tag{2.5.2}$$

Wir wollen nun ausarbeiten, wie man diese Art Integral-Transformation rein mathematisch interpretieren kann, um eine alternative Sicht auf solche Art Transformation zu erhalten. Betrachten wir dazu das Signal x wieder als Vektor in einem Vektorraum, in dem ein *Skalarprodukt* definiert ist. Um zu erläutern, was das bringt, starten wir mit etwas Bekanntem. Im endlich-dimensionalen Fall, also wenn $\mathbf{x}, \mathbf{y} \in \mathbb{C}^N$, dann ist ein mögliches Skalarprodukt $\langle \cdot, \cdot \rangle : \mathbb{C}^N \times \mathbb{C}^N \rightarrow \mathbb{R}$ durch

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^N x_i \cdot y_i^* = \mathbf{y}^H \cdot \mathbf{x}$$

gegeben, wobei der letzte Ausdruck das Matrix-Vektor-Produkt einer $\mathbb{C}^{N \times 1}$ Matrix und einem Vektor in \mathbb{C}^N darstellt. Man kann nun $\langle \mathbf{x}, \mathbf{y} \rangle$ betrachten, um zu ermitteln, wie „ähnlich“ die Vektoren \mathbf{x} und \mathbf{y} sind. Beispielsweise nennen wir die beiden Vektoren orthogonal/senkrecht, falls $\langle \mathbf{x}, \mathbf{y} \rangle = 0$. In diesem Fall

interpretieren wir dies intuitiv so, dass x und y keine „gemeinsamen“ Anteile haben. Beispielsweise, wenn wir einen dritten Vektor $z = \alpha x + \beta y$ betrachten und es gilt, dass $\langle x, y \rangle = 0$, dann schlägt sich dies auch auf das Skalarprodukt von $\langle x, z \rangle$ nieder, da

$$\langle x, z \rangle = \langle x, \alpha x + \beta y \rangle = \alpha \langle x, x \rangle.$$

Beim Bilden von $\langle x, z \rangle$ können wir also einen etwas asymmetrischen Standpunkt einnehmen und uns vorstellen, dass wir ermitteln, wie viele „Anteile“ von x in z zu finden sind.

Man sieht, dass dies in gewisser Weise erlaubt mit hochdimensionalen Objekten eine Art Geometrie zu betreiben. Man kann noch weiter gehen und mit einem Skalarprodukt Korrelationen, Winkel und Längen definieren.

Wir wollen nun dieses Konzept auf kontinuierliche Signale erweitern und dann die **FT** in diesem Licht interpretieren. Man kann zeigen, dass für zwei Funktionen $x, y : \mathbb{R} \rightarrow \mathbb{C}$ die Abbildung $(x, y) \mapsto \langle x, y \rangle$ mit

$$\langle x, y \rangle = \int_{-\infty}^{+\infty} x(t) \cdot y^*(t) dt \quad (2.5.3)$$

ein Skalarprodukt definiert. Auch hier halten wir an der Interpretation fest, dass wir damit berechnen, wie ähnlich sich x und y sind. Definieren wir nun die Signale $k_F : \mathbb{R} \rightarrow \mathbb{C}$ durch $k_F(t) = \exp(j2\pi F t)$, dann können wir (2.5.1) umschreiben zu

$$\langle x, k_F \rangle = \int_{-\infty}^{+\infty} x(t) \cdot \exp(j2\pi F t)^* dt = \int_{-\infty}^{+\infty} x(t) \cdot \exp(-j2\pi F t) dt = X(f).$$

Dies suggeriert uns also, dass wir die **FT** als Skalarprodukt von dem zu transformierenden Signal und einem geeignet gewählten Transformationskern interpretieren können. Die Zuordnung von $F \mapsto \langle x, k_F \rangle$ definiert dann im Grunde die Funktion X für alle F .

Genauso erhalten wir für die Folge der Abtastwerte $x[\cdot]$ die **Discrete Time Fourier Transform (DTFT)** als Skalarprodukt von $x[\cdot]$ und der Folge $k_f[n] = \exp(j2\pi f n)$, also

$$X(f) = \sum_{n \in \mathbb{Z}} x[n] k_f[n]^* = \sum_{n \in \mathbb{Z}} x[n] \exp(-j2\pi f n). \quad (2.5.4)$$

Die inverse **DTFT** ergibt sich durch

$$x[n] = \int_{-1/2}^{+1/2} X(f) \exp(j2\pi f n) df.$$

Wir haben nun alle Werkzeuge zusammen, um genauer zu Analysieren, was bei Abtastung von x_a zu $x[\cdot]$ geschieht. Wir erinnern uns, dass gilt für die Abtastpunkte gilt, dass $t = nT = n/F_s$ gilt. Wir erhalten dann mit $x[n] = x_a(nT)$ und der inversen **FT** angewandt auf X_a , dass

$$x[n] = x_a(nT) = \int_{-\infty}^{+\infty} X_a(F) \exp(j2\pi F/F_s n) dF$$

gelten muss. Setzen wir nun auch für $x[\cdot]$ die inverse DTFT ein, erhalten wir mit $f = F/F_s$ (siehe Abschnitt 2.4) als Variablentransformation, dass

$$\frac{1}{F_s} \int_{-F_s/2}^{+F_s/2} X(f) \exp(j2\pi F/F_s n) dF = \int_{-\infty}^{+\infty} X_a(F) \exp(j2\pi F/F_s) dF$$

gilt. Wichtig ist an diesem Ausdruck, dass wir nun einen Zusammenhang zwischen der DTFT X und der FT X_a gefunden haben. Die Integration auf beiden Seiten ist auch bezüglich desselben Integrationskernes $\exp(j2\pi \cdot / F_s n)$. Das heißt, dass es möglich sein sollte, einen Ausdruck herzuleiten, der X in Abhängigkeit von X_a ausdrückt. Mit ein wenig algebraischem Grindwork findet man

$$X(f) = F_s \sum_{k \in \mathbb{Z}} X_a((f - k)F_s). \quad (2.5.5)$$

Wie ist dies nun zu interpretieren? Im Grunde bestätigt (2.5.5) für beliebige Signale, was wir bereits für Signale mit einzelnen Frequenzen (siehe (2.1.1)) gefunden hatten. Das Spektrum wird durch die Abtastung periodifiziert und zwar entsteht das periodische Spektrum X durch Wiederholung von um $1/F_s$ gestauchte Kopien von X_a mit Abstand $1/F_s$.

Andererseits kann man es auch so sehen, dass der Wert $X(f)$ sich ergibt aus Summation der Werte $X_a(f \cdot F_s - kF_s)$ für alle $k \in \mathbb{Z}$. Also an der Stelle f erscheinen die Werte von X_a an den Stellen $fF_s - kF_s$. Das heißt, wie die Periodifizierung vonstatten geht hängt maßgeblich von der Samplingfrequenz F_s ab!

Man sieht nun auch wieder schön, dass unsere obige Argumentation zum Verhältnis von Samplingfrequenz und maximaler Frequenz im Signal immernoch gilt, da nun gelten muss, dass alle belegten Frequenzen von x_a sich im Intervall $[-F_s/2, +F_s/2]$ befinden müssen, da sich sonst die Kopien von X_a in (2.5.5) überlappen. Dies ist äquivalent zur Forderung, dass

$$|X_a(F)| = 0 \quad \text{für } |F| > F_s/2$$

gelten muss. In diesem Fall spricht man davon, dass X_a bandbegrenzt ist. Das kleinste $F \in \mathbb{R}$, was obige Ungleichung erfüllt, nennen wir F_{\max} .

Wir sind nun beinahe am Ziel angelangt, dass wir das Samplingtheorem formulieren können. Wir benötigen lediglich noch eine Möglichkeit aus den Abtastwerten $x[\cdot]$ das Signal x_a zu rekonstruieren. Wenn kein Aliasing vorliegt, also $F_{\max} < F_s/2$, dann können wir X_a aus X berechnen, indem wir

$$X_a(F) = \begin{cases} \frac{X(F/F_s)}{F_s}, & |F| \leq F_s/2, \\ 0 & \text{sonst} \end{cases}$$

setzen. Wir wissen außerdem, dass sich X durch die DTFT durch

$$X(f) = \sum_{n \in \mathbb{Z}} x[n] \exp(-j2\pi n F/F_s)$$

berechnen lässt. Weiter können wir nun auch x_a durch Integration über einen endlichen Bereich durch

$$x_a(t) = \int_{-F_s/2}^{+F_s/2} X_a(F) \exp(j2\pi F t) dF.$$

erhalten. Wir setzen nun einfach alles ein und erhalten

$$x_a(t) = \frac{1}{F_s} \int_{-F_s/2}^{+F_s/2} \sum_{n \in \mathbb{Z}} x[n] \exp(-j2\pi n F/F_s) \exp(j2\pi F t) dF.$$

Toll, aber wir müssen dies nun noch ein wenig umformen. Zuerst vertauschen wir Integration und Summation, weil wir es dürfen und nutzen $x[n] = x_a(n/F_s)$. Dies ergibt

$$x_a(t) = \frac{1}{F_s} \sum_{n \in \mathbb{Z}} x_a(n/F_s) \int_{-F_s/2}^{+F_s/2} \exp(-j2\pi n F/F_s) \exp(j2\pi F t) dF.$$

Wir sind nicht wirklich schlauer, aber wir wissen aus unserer Erfahrung von Integration von komplexen Funktionen, dass

$$\frac{1}{F_s} \int_{-F_s/2}^{+F_s/2} \exp(-j2\pi n F/F_s) \exp(j2\pi F t) dF = \frac{\sin(\pi F_s(t - n/F_s))}{\pi F_s(t - n/F_s)} = \text{sinc}(F_s(t - n/F_s)).$$

Wir nutzen nun wiederum die Interpretation von Integration eines Produktes von Funktionen als deren Skalarprodukt. Wir bilden also das Skalarprodukt von zwei Funktionen mit Periode F_s , d.h. es genügt das Integral im Intervall $[-F_s/2, +F_s/2]$ zu bilden. Sehen wir genauer hin, bilden wir also das Skalarprodukt $\langle k_t, k_{n/F_s} \rangle$, wobei ähnlich wie oben

$$k_t(F) = \exp(j2\pi F t)$$

definiert wurde. Das heißt nun in der Sprache von Skalarprodukten, dass wir die sinc-Funktion erhalten, als

$$\text{sinc}(F_s(t - n/F_s)) = \langle k_t, k_{n/F_s} \rangle.$$

Wenn man es ein wenig allgemeiner formuliert, kann man sagen, dass die sinc-Funktion nur ein spezieller *Interpolationskern* ist, der sich aus der Definition von k_t ergibt. Wir werden später noch ein weiteres Beispiel für so einen Kern betrachten. Zusammenfassend für diesen Abschnitt 2 steht nun folgendes

Theorem 2.1 (Samplingtheorem). *Gegeben sei ein analoges Signal x_a mit Frequenzen in $[-F_{\max}, +F_{\max}]$ und dessen Abtastwerte $x[\cdot]$ mit $x[n] = x_a(nT)$, wobei $T = 1/F_s$.*

Falls $F_s > 2F_{\max}$, dann gilt

$$x_a(t) = \sum_{n \in \mathbb{Z}} x[n] \cdot g_{F_s}(t - nT), \quad (2.5.6)$$

wobei der Interpolationskern $g : \mathbb{R} \rightarrow \mathbb{R}$ gegeben ist durch

$$g_{F_s}(t) = \frac{\sin(\pi F_s t)}{\pi F_s t}.$$

Hinweis: Das Samplingtheorem, wie es in [1] formuliert ist, beinhaltet einen kleinen, aber entscheidenden Fehler. Die Definition des Interpolationskernes g_F ist dort mit

$$g(t) = \frac{\sin(2\pi B t)}{2\pi B t}.$$

gegeben. Doch in diesem Falle würde der Interpolationskern von der Bandbreite des Signals x_a , aber nicht von der Abtastrate F_s abhängen. Die angegebene Definition ist nur korrekt, falls $F_s = 2B$, wir also mit der *minimalen* Samplerate abgetastet haben. In diesem Fall spricht man auch von *kritischer Abtastung*.

```

F_max = 1.0
F = np.random.choice(
    np.linspace(
        -F_max, +F_max, 11, endpoint=True
    ),
    5,
    replace=False,
) # Hz
theta = np.random.uniform(0, 2 * np.pi, 5) # rad
A = np.random.randn(5) # amplitude

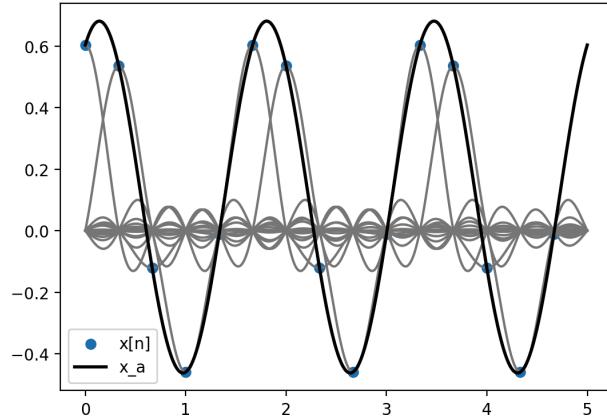
def x_a(t: np.ndarray) -> np.ndarray:
    return np.sum(
        np.cos(
            2 * np.pi * np.outer(t, F) + theta
        )
        * A,
        axis=1,
    )

t = np.linspace(0, 5, 1024)
F_s = 3

def g(t: np.ndarray) -> np.ndarray:
    nenner = np.pi * F_s * t
    nenner[np.isclose(t, 0)] = 1
    result = np.sin(np.pi * F_s * t) / nenner
    result[np.isclose(t, 0)] = 1
    return result

n = (
    np.arange(int(t[-1] * F_s)).astype(float)
    / F_s
)
x_n = x_a(n)
interpol = np.zeros_like(t)
for nn in np.arange(len(n)):
    interpol += x_n[nn] * g(t - nn / F_s)
plt.plot(
    t,
    x_n[nn] * g(t - nn / F_s),
    color="grey",
)

```



Codeschnipsel 5: Berechnung und Darstellung von Theorem 2.1, siehe [dsv/code/sampling_theorem.py](#)

Es ist weiterhin noch zu erwähnen, dass wir genau genommen immer noch nicht wissen, wie wir überprüfen können, welche maximale Frequenz F_{\max} von einem analogen Signal x_a belegt wird. Dies liegt daran, dass wir im Vorhinein – also vor Abtastung – keinen Zugriff auf das abzutastende Signal haben. Das heißt, dass wir auch noch nicht überprüfen können, ob wir das Samplingtheorem eingehalten haben, oder einhalten werden, falls wir Abtasten sollten. Man muss sich also für ein gegebenes System, das einem Signale produziert, die abgetastet werden sollen, auf anderem Weg überlegen, wie man die Bedingung $|X_a(F)| = 0$ für $|F| > F_s/2$ überprüfen kann.

3 Diskrete Signale und Systeme

3.1 Diskrete Signale

Wir sind nun endlich im Digitalen angekommen. Wir wollen uns als erstes verschiedene Möglichkeiten der Klassifikation von diskreten Signalen $x[\cdot] : \mathbb{Z} \rightarrow \mathbb{C}$ ansehen. Dabei folgend wir weitestgehend [1, Kap. 2.1]. In Abschnitt 1.2.5 haben wir bereits den Einheitsstoß $\delta[\cdot]$ und die Heavy-Side-Funktion $u[\cdot]$ kennengelernt.

- Eine linear ansteigende Version $u_r[\cdot]$ von $u[\cdot]$ ist gegeben durch

$$u_r[n] = n \cdot u[n] = \begin{cases} n, & \text{für } n \geq 0 \\ 0, & \text{sonst} \end{cases}.$$

In Python ist die Funktion auch sehr einfach zu implementieren:

```
def u_r(n: int) -> int:
    return n if n>0 else 0
```

Will man die Funktion effizienter mittels Numpy [2] implementieren, dann liest sie sich

```
import numpy as np
def u_r_np(n: np.ndarray[int]) -> np.ndarray[int]:
    u_r = n.copy()
    u_r[n <= 0] = 0
    return u_r
```

- Für eine komplexe Zahl $a = r \exp(j\theta) \in \mathbb{C}$ erhält man das zugehörige exponentielle Signal als

$$x[n] = a^n = r^n \exp(j\theta n) = r^n \cos(\theta n) + j r^n \sin(\theta n).$$

Damit gilt $x[0] = 1$ unabhängig von a . Beispielsweise erhalten wir das Signal x_k aus (2.3.1) indem wir $r = 1$ und $\theta = 2\pi kf_0$ setzen. Eine Implementierung von $x[n]$ ist in Codeschnipsel 6 gegeben. Es ist sicher interessant für verschiedene Werte von a und n die Ausgabe zu betrachten.

Beispielsweise kann man sehen, dass für $a \in \mathbb{R}$ gelten muss, dass $\lim_{n \rightarrow \infty} x[n] = 0$, falls $a < 0$, aber $\lim_{n \rightarrow -\infty} |x[n]| = \infty$ andernfalls. Weiterhin gilt für $a \in \mathbb{C}$ und $|a| = 1$, dass dann auch $|x[n]| = 1$ für alle n .

3.1.1 Energie Diskreter Signale

Oft ist es interessant zu bemessen, wie viel Energie in einem Signal vorhanden ist. Für eine physikalisch korrekte Bemessung dieser Energie, müsste man das Signal zwar mit Einheiten versehen, aber diese ergeben nur einen entsprechenden Proportionalitätsfaktor. Hierzu betrachten wir

$$E(x[\cdot]) = \sum_{n \in \mathbb{Z}} |x[n]|^2 = \sum_{n \in \mathbb{Z}} x[n]^* \cdot x[n]. \quad (3.1.1)$$

Wenn gilt $E(x[\cdot]) < \infty$, dann sprechen wir erstaunlicherweise von einem Signal endlicher Energie.

Es ist nun interessant sich eine Menge \mathcal{E} zu definieren, die alle Signale enthält, welche endliche Energie besitzen, also

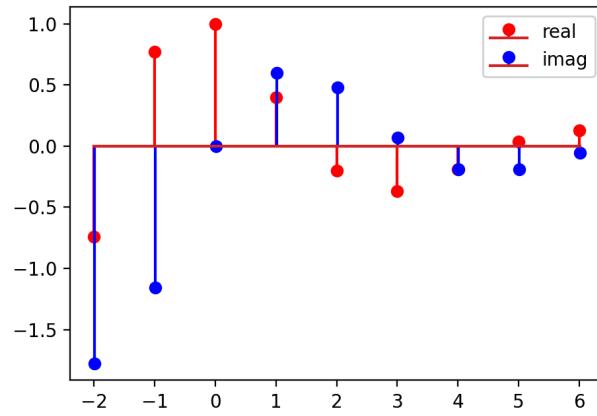
$$\mathcal{E} = \{x : \mathbb{Z} \rightarrow \mathbb{C} \text{ mit } E(x[\cdot]) < \infty\}.$$

```

def complex_exp(
    n: np.ndarray[int], a: complex
) -> np.ndarray[complex]:
    return np.abs(a) ** n * np.exp(
        1j * np.angle(a) * n
    )

n = np.linspace(-2, +10, 13, dtype=int)
a = 0.4 + 1j * 0.6
plt.stem(
    n,
    complex_exp(n, a).real,
    linefmt="r",
    label="real",
)
plt.stem(
    n + 0.1,
    complex_exp(n, a).imag,
    linefmt="b",
    label="imag",
)
plt.legend()
plt.show()

```



Codeschnipsel 6: Berechnung und Darstellung eines komplexen exponentiellen Signals, siehe [dsv/code/complex_exp.py](#)

Man kann sich nun überlegen, dass

$$E(\alpha x[\cdot] + \beta y[\cdot]) \leq E(\alpha x[\cdot]) + E(\beta y[\cdot]) = \alpha^2 E(x[\cdot]) + \beta^2 E(y[\cdot]) < \infty$$

gelten muss, falls $E(x[\cdot]), E(y[\cdot]) < \infty$. Das heißt, dass Linearkombinationen von Signalen mit endlicher Energie wieder ein Signal mit endlicher Energie ergeben. Das heißt, dass die Signale endlicher Energie bilden einen *Unterraum*. Wir können noch einen Schritt weiter gehen und wie in (2.5.4) die Summe in (3.1.1) als Skalarprodukt auffassen.

Definieren wir für zwei Signale endlicher Energie die Abbildung $\langle \cdot, \cdot \rangle : \mathcal{E} \times \mathcal{E} \rightarrow \mathbb{C}$ als

$$(x[\cdot], y[\cdot]) \mapsto \langle x[\cdot], y[\cdot] \rangle = \sum_{n \in \mathbb{Z}} x[n]^* \cdot y[n], \quad (3.1.2)$$

dann kann man sich überlegen, dass dies die Bedingungen an ein *Skalarprodukt* erfüllt. Beispielsweise kann man nachrechnen, dass die unendliche Summe in (3.1.2) immer endlich ist, falls $x[\cdot], y[\cdot] \in \mathcal{E}$, da

$$|\langle x[\cdot], y[\cdot] \rangle| \leq E(x[\cdot]) \cdot E(y[\cdot]) < \infty$$

Nun kann man aber auch E durch

$$E(x[\cdot]) = \langle x[\cdot], x[\cdot] \rangle$$

ausdrücken.

Beispiel 3.1. Betrachten wir $x[n] = a^n \cdot u[n]$ für $a = r \exp j\theta \in \mathbb{C}$. Dann berechnet sich $E(x[\cdot])$ durch

$$E(x[\cdot]) = \sum_{n \geq 0} |a^n|^2 = \sum_{n \geq 0} (r^2)^n.$$

Ist nun $r \geq 1$, dann $E(x[\cdot]) = \infty$, falls aber $r < 1$, dann ergibt sich aus der geometrischen Reihe, dass

$$E(x[\cdot]) = \frac{1}{1 - r^2}$$

gilt. Das heißt auch, dass das Heavy-Side-Signal $u[\cdot]$ keine endliche Energie besitzt.

3.1.2 Periodische Signale

Gilt für ein Signal $x[\cdot]$, dass $x[n + N] = x[n]$ für ein $N \in \mathbb{N}$ und alle $n \in \mathbb{Z}$, so nennt man $x[\cdot]$ periodisch mit Periodenlänge/Periode N , oder kurz N -periodisch, siehe beispielsweise Gleichung (2.3.1). Falls $x[\cdot]$ nun N -periodisch ist, dann ist $x[\cdot]$ auch kN -periodisch, falls $k \in \mathbb{N}$. Das heißt, dass es sinnvoller ist, das *kleinste* $N \in \mathbb{N}$ zu betrachten, sodass $x[\cdot]$ dann N -periodisch ist. Man nennt N dann Fundamentalperiode. Falls solch ein N nicht existiert, dann nennt man $x[\cdot]$ aperiodisch, oder nicht-periodisch. Falls $x[\cdot] \neq 0$, dann gilt für periodische Signale, dass $E(x[\cdot]) = \infty$. Beispielsweise haben wir bereits in Abschnitt 2.4 gesehen, dass

$$x[n] = \exp(j2\pi f)$$

periodisch mit Periode N ist, falls $f = k/N$, also eine rationale Zahl ist.

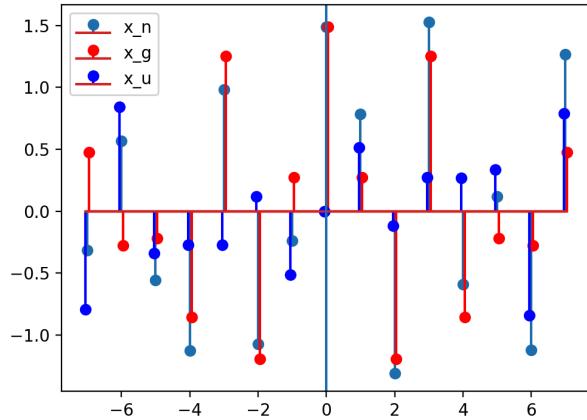
```

# assume n=0 in center, size(x_n) = 2K+1
def even(x_n: np.ndarray) -> np.ndarray:
    return 0.5 * (x_n + x_n[::-1])

# assume n=0 in center, size(x_n) = 2K+1
def odd(x_n: np.ndarray) -> np.ndarray:
    return 0.5 * (x_n - x_n[::-1])

K = 7
n = np.linspace(-K, +K, 2 * K + 1)
x_n = np.random.randn(n.size)

```



Codeschnipsel 7: Zerlegung eines Signals in seinen geraden und ungeraden Anteil., siehe [dsv/code/even_odd.py](#)

3.1.3 Symmetrie von Signalen

Gilt für ein Signal $x[n] = x[-n]$, dann nennt man es *symmetrisch* bzw. *gerade*. Gilt andererseits $x[n] = -x[-n]$, so nennt man es *anti-symmetrisch* bzw. *ungerade*.

Ist ein beliebiges Signal $x[\cdot]$ gegeben, so kann man

$$x_g[n] = \frac{1}{2} (x[n] + x[-n]) \quad \text{und} \quad x_u[n] = \frac{1}{2} (x[n] - x[-n])$$

definieren. Dann ist $x_g[\cdot]$ gerade und falls $x[\cdot]$ bereits gerade ist, so gilt $x[\cdot] = x_g[\cdot]$. Genauso ist $x_u[\cdot]$ ungerade und falls $x[\cdot]$ bereits ungerade ist, so gilt $x[\cdot] = x_u[\cdot]$. Außerdem gilt

$$x[n] = x_g[n] + x_u[n].$$

Wir haben das Signal $x[\cdot]$ also in einen geraden und einen ungeraden Teil zerlegt. Dies ist manchmal sinnvoll, wenn man solch ein Signal linear transformiert und weiß, dass die lineare Transformation für gerade oder ungerade Signale gewisse Eigenschaften hat. Wie man an Codeschnipsel 7 gut sehen kann, muss gelten $x_u[0] = 0$, da $x_u[0] = x[0] - x[0] = 0$ und $x_g[0] = x[0]$, da $2x_g[0] = x[0] + x[0]$.

3.2 Diskrete Systeme

Nachdem wir uns nun ein wenig mit diskreten Signalen vertraut gemacht haben, sind wir in der Lage und mit diskreten Systemen zu befassen. Ganz allgemein kann man fast jeden Prozess, an dessen Anfang ein diskretes Signal steht und dessen Ergebnis wiederum ein diskretes Signal ist, als ein diskretes System auffassen. Sobald man dieses System nun einmal vorliegen hat, will man Techniken und Werkzeuge entwickeln, wie man dieses System systematisch untersuchen kann – eine Systematik der Systeme.

Ein System \mathcal{T} wird mathematisch als Abbildung eines (Eingabe-)Signals $x[\cdot]$ auf ein anderes (Ausgabesignal) $y[\cdot]$ aufgefasst. Wir schreiben dafür dann

$$x[\cdot] \mapsto \mathcal{T}(x[\cdot])[\cdot] = y[\cdot]. \quad (3.2.1)$$

Das System \mathcal{T} bildet also die Paare $(x[\cdot], \mathcal{T}(x[\cdot])) = (x[\cdot], y[\cdot])$.

Betrachten wir folgendes

Beispiel 3.2. Gegeben sei das Eingabesignal

$$x[n] = \begin{cases} |n|, & \text{falls } -3 \leq n \leq +3, \\ 0 & \text{sonst.} \end{cases}$$

Wir sind nun an den Werten der Ausgabesignals $y[\cdot]$ interessiert, für

- a) das Einheitssystem $y[n] = x[n]$,
- b) das Einheitsdelay-System $y[n] = x[n - 1]$,
- c) das Einheitsadvance-System $y[n] = x[n + 1]$

interessiert.

Diese Systeme waren noch ein wenig simpel und man ist vielleicht noch nicht überzeugt, dass eine genauere Analyse von diskreten Systemen notwendig sein sollte. Dies liegt vor allem daran, dass die Systeme in Beispiel 3.2 nur „lokal“ gearbeitet haben, da Werte $y[n]$ nur von Werten $x[n - 1]$, $x[n]$ und $x[n + 1]$ abhängen.

Betrachten wir wiederum die Mandelbrot-Iteration¹, aber diesmal als System

$$y[n + 1] = y[n]^2 + x[n]$$

für verschiedene Eingangssignale $x[n] = c \in \mathbb{C}$ mit $y[0] = 0$. Wir sind nun an solchen c interessiert für welche das System divergiert, also $|y[n]| > 2$ ab einem gewissen n . Wir wollen aber solche c finden, für welche wir in einem gewissen Bereich $n \in [n_{\text{low}}, n_{\text{high}}]$ divergieren. Das in Codeschnipsel 8 gezeigte Beispiel ist hierbei am anderen Ende des Komplexitätsspektrums, da man bei diesem System eher von einem „chaotischen“ System sprechen sollte. Kleine Veränderungen an $x[n] = c$ haben großen Einfluss auf das Divergenzverhalten der Folge $y[n]$ ².

Um zu sehen, wie Systeme von ihrem Anfangszustand abhängen können, wollen hierfür ein etwas einfacheres Beispiel betrachten. Gegeben ist das System

$$y[n] = \sum_{k=-\infty}^n x[k].$$

Wir sehen hier, dass es für die Berechnung von $y[n]$ nicht ausreicht, den Zustand des Eingangs zum Zeitpunkt n , also $x[n]$ zu kennen. Schließlich müssen wir die gesamte Vergangenheit von $x[\cdot]$ bis zum Zeitpunkt n in die Berechnung einfließen lassen. Wir können das System aber umschreiben in

$$y[n] = \sum_{k=-\infty}^{n-1} x[k] + x[n] = y[n-1] + x[n],$$

wobei wir nun auch sehen, warum dieses System *Akkumulator* genannt wird, da $y[\cdot]$ im Prinzip die Werte von $x[\cdot]$ „aufsammelt“.

¹siehe [dsv/code/mandelbrot.py](#)

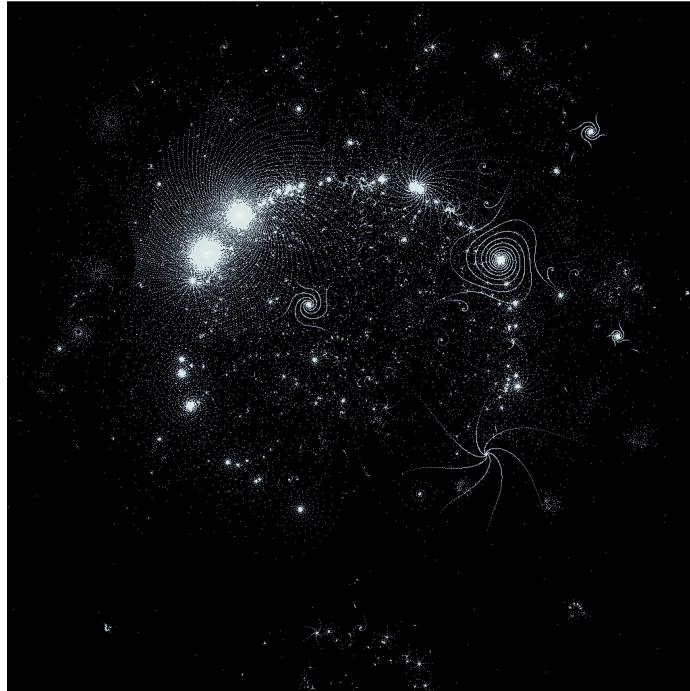
²<https://erleuchtet.org/2010/07/ridiculously-large-buddhabrot.html>

```

success = []
for tt in trial():
    valid, counter = sample(tt, 100, 50000)
    if valid:
        success += [(tt, counter)]
    if len(success) > 400:
        break

image = np.zeros((1024, 1024), dtype=int)
for ss in success:
    x_n = ss[0]
    for ii in range(ss[1]):
        image[
            int(
                image.shape[0]
                * (x_n.real / 2 + 0.5)
            )
            %
            image.shape[0],
            int(
                image.shape[1]
                * (x_n.imag / 2 + 0.5)
            )
            %
            image.shape[1],
        ] += 1
    x_n = x_n**2 + ss[0]

```



Codeschnipsel 8: Spät divergierende Orbits der Mandelbrot-Iteration, siehe [dsv/code/buddhabrot.py](#)

```

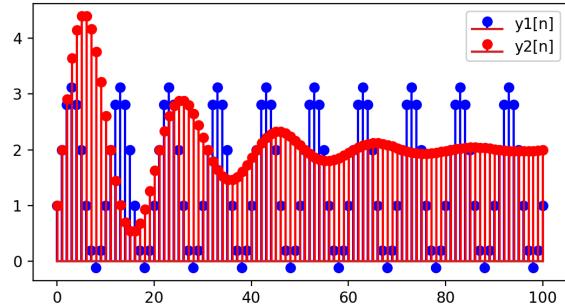
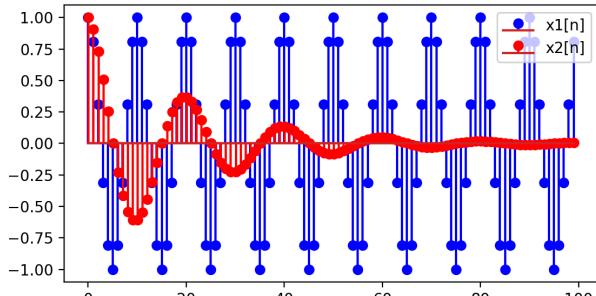
n = np.arange(100)
x_1_n = np.cos(2 * np.pi * 0.1 * n)
x_2_n = np.cos(2 * np.pi * 0.05 * n) * np.exp(
    -n / 20
)
y_0 = 1

```

```

y_1_n = np.concatenate(
    [[y_0], y_0 + np.cumsum(x_1_n)]
)
y_2_n = np.concatenate(
    [[y_0], y_0 + np.cumsum(x_2_n)]
)

```



Codeschnipsel 9: Akkumulator für zwei verschiedene Eingangssignale, siehe [dsv/code/accumulator.py](#)

Stellen wir uns nun vor, dass wir dieses System modellieren/simulieren wollen für $n \leq n_0$, so benötigen wir entweder die Werte $x[n]$ für $n < n_0$, oder die sogenannte *Anfangsbedingung* $y[n_0] = y_0$. Dies erinnert an das Lösen einer Differentialgleichung

$$\dot{y}(t) = x(t),$$

wonach dann gilt, dass

$$y(t) = \int_{-\infty}^t x(s) ds \quad \text{oder} \quad y(t) = y_0 + \int_{t_0}^t x(s) ds,$$

damit $y(t_0) = y_0$. Ein Beispiel für die Wirkung von einem Akkumulator ist in Codeschnipsel 9 gegeben. Man sieht sehr gut, welchen Einfluss die Anfangsbedingung auf $x_2[\cdot]$ hat, da dies den Grenzwert $\lim_{n \rightarrow \infty} y[n]$ maßgeblich beeinflusst. Falls gilt, dass $y_0 = 0$, so spricht man vom Ruhezustand, beziehungsweise dem Nullzustand in dem sich das System zum Zeitpunkt $n = n_0$ befindet.

Statisch vs. Dynamisch Man kann Systeme nun auf verschiedene Arten klassifizieren. Beispielsweise nennen wir Systeme *statisch*, wenn der Wert $y[n]$ nur von $x[n]$ abhängt, aber nicht von vergangenen oder gar zukünftigen Werten (entweder von $y[\cdot]$ oder $x[\cdot]$). Die Systeme

$$y[n] = ax[n], \quad \text{oder} \quad y[n] = \sqrt{x[n]} + x^4[n]$$

sind statisch. Hängt nun $y[n]$ von seiner eigenen Vergangenheit oder der von $x[n]$ ab, so nennen wir diese Systeme dynamisch, oder Systeme mit Gedächtnis. Die Systeme

$$y[n] = x[n] + ax[n-1], \quad y[n] = \sum_{k=0}^N x[n-k]$$

sind dynamisch und haben jeweils Gedächtnis der Länge $g = 1$, beziehungsweise $g = N - 1$. Man sieht bereits an Codeschnipsel 9, dass dynamische Systeme im Allgemeinen interessanter sein werden.

Kausal vs. Akausal Hängen die Werte $y[n]$ nur von $x[n], x[n-1], \dots$ ab, also nicht auch von $x[n+1], x[n+2], \dots$, so nennen wir das System kausal. Intuitiv bedeutet dies die intuitiv bekannte Kausalität in dem Sinne, dass nur die Zeitliche Vergangenheit notwendig ist, um den aktuellen Zustand des Systems zu bestimmen. Ist die nicht gegeben, so nennt man das System *akausal* oder *nicht-kausal*.

Was die Realisierung von aksualen Systemen angeht, wird es nicht möglich sein, diese in Echtzeit umzusetzen, da man diese nur mit einer Verzögerung, oder gar nicht für sequentiell verfügbares $x[\cdot]$ implementieren kann. Sind die Werte $x[n]$ jedoch beispielsweise durch Messung oder Simulation entstanden und „offline“ verfügbar, so können solche Systeme durchaus angewandt werden und nützlich sein.

Stabil vs. Instabil Eine der zentralen Eigenschaften, die auch bei analogen Systemen eine Rolle spielt ist Stabilität. Zwar hat man normalerweise bereits einen intuitiven Begriff für Stabilität im Sinn, doch formal gibt es hiervon verschiedene Ausprägungen. Wir beschränken uns auf die Version der **Bounded-Input-Bounded-Output (BIBO)**-Stabilität, welche fordert, dass für beschränktes Eingangssignal $x[\cdot]$ der Ausgang $y[\cdot]$ ebenfalls beschränkt bleibt. Formal fordern wir also, dass falls

$$|x[n]| \leq M_x \quad \text{für alle } n \in \mathbb{Z}$$

für eine Konstante $M_x \in \mathbb{R}$ gilt, dann auch

$$|y[n]| \leq M_y \quad \text{für alle } n \in \mathbb{Z}$$

gelten muss. Man sieht, dass M_x und M_y an $x[\cdot]$ und $y[\cdot]$ gebunden sind, also keine „universellen“ Konstanten sind. Andersherum reicht es also für Instabilität nur ein beschränktes Eingangssignal $x[\cdot]$ konstruiert werden muss, für welches $y[\cdot]$ nicht beschränkt bleibt. Betrachten wir folgendes

Beispiel 3.3. Gegeben sei

$$y[n] = C \cdot y[n-1]^2 + x[n]$$

und wir nutzen als Eingang den Einheitsstoß $x[\cdot] = C\delta[\cdot]$, welcher beschränkt ist, mit $M_x = |C|$ für $C \in \mathbb{R}$. Doch dieser produziert für $y[n] = 0$ für $n \leq -1$ die Folge

$$y[n] = \underbrace{\{0, C, C^2, C^4, \dots, C^{2^n}\}}_{\uparrow},$$

welche für $|C| > 1$ gegen ∞ divergiert.

Zeitvariant vs. Zeitinvariant Systeme deren Eingabe-Ausgabe-Verhalten nicht zeitlich konstant ist, nennt man *zeitvariant*. Systeme, die für zeit verzögerte Eingaben, die um den gleichen Zeitraum verzögerte Ausgaben produzieren, nennt man *zeitinvariant*, siehe Abschnitt 1.2.6. Formal fordern wir für Zeitinvarianz, dass wenn für beliebige Eingabe $x[\cdot]$

$$x[\cdot] \xrightarrow{T} y[\cdot]$$

gilt, dass dann für jedes $k \in \mathbb{Z}$ auch gilt, dass

$$x[\cdot - k] \xrightarrow{T} y[\cdot - k]$$

erfüllt ist. In der Schreibweise von (3.2.1) heißt dies, dass für

$$y[n] = \mathcal{T}(x[\cdot])[n]$$

auch gelten muss, dass

$$y[n-k] = \mathcal{T}(x[\cdot-k])[n]$$

und zwar für alle $x[\cdot]$ und k .

Was erst einmal relativ abstrakt daherkommt, ist eigentlich eine sehr intuitive Sache. Wenn wir beispielsweise an ein Audiointerface denken, so hätten wir schon gerne, dass es egal ist, zu welchem Zeitpunkt jemand ins Mikrofon singt – unabhängig vom Zeitpunkt sollte die Aufnahme „gleich klingen“. Das System, welches die Aufnahme und eventuelle Audioverarbeitung realisiert, sollte keine zeitlichen Veränderungen zeigen. Auch in der umgekehrten Richtung, beim Abspielen von Ton, sollte es irrelevant sein, zu welchem Zeitpunkt man ein gewisses Stück hören möchte – das Hörerlebnis sollte davon nicht beeinflusst sein.

Im Grunde ist Zeitinvarianz also etwas, das wir normalerweise von einem System „erwarten“ und nicht untersuchen wollen.

Linear vs. Nicht-Linear Kommen wir zum Schluss dieser Klassifikationen zu einer der wichtigsten Unterscheidungen. Ein System, das für Eingänge $x_1[\cdot]$ und $x_2[\cdot]$ die Antworten

$$y_1[n] = \mathcal{T}(x_1[\cdot])[n] \quad \text{und} \quad y_2[n] = \mathcal{T}(x_2[\cdot])[n]$$

produziert, nennen wir *linear*, falls die Antwort des Systems auf den Eingang

$$x[\cdot] = a_1x_1[\cdot] + a_2x_2[\cdot]$$

sich durch

$$y[n] = \mathcal{T}(x[\cdot])[n] = \mathcal{T}(a_1x_1[\cdot] + a_2x_2[\cdot])[n] = a_1\mathcal{T}(x_1[\cdot])[n] + a_2\mathcal{T}(x_2[\cdot])[n]$$

ausdrücken lässt.

Abbildung 3 zeigt die beiden möglichen Interpretationen dieser Eigenschaft. Man kann sich also vorstellen, dass die Eingänge *erst* skaliert und addiert werden und dann das System durchlaufen, oder man prozessiert beide Eingänge durch \mathcal{T} und skaliert und addiert die *Ausgänge nachdem* das System im Grund „zweifach“ angewandt wurde.

Anders gesagt „passen“ lineare Systeme genau zu der linearen Struktur von Signalen, wenn wir sie als Vektoren in einem Vektorraum auffassen. Als Konsequenz ergibt sich, dass sich lineare Systeme deutlich einfacher analysieren lassen, weil wir Werkzeuge der linearen Algebra benutzen können. Diese Eigenschaft ist so attraktiv, dass man oft versucht nichtlineare Systeme durch geeignete lineare Systeme zu approximieren (Bspw.: Pendel $\sin(x) \approx x$). Man nimmt also Fehler in der Analyse in Kauf³, ist damit aber immerhin in der Lage überhaupt Aussagen treffen zu können.

³https://en.wikipedia.org/wiki/Hartman-Grobman_theorem

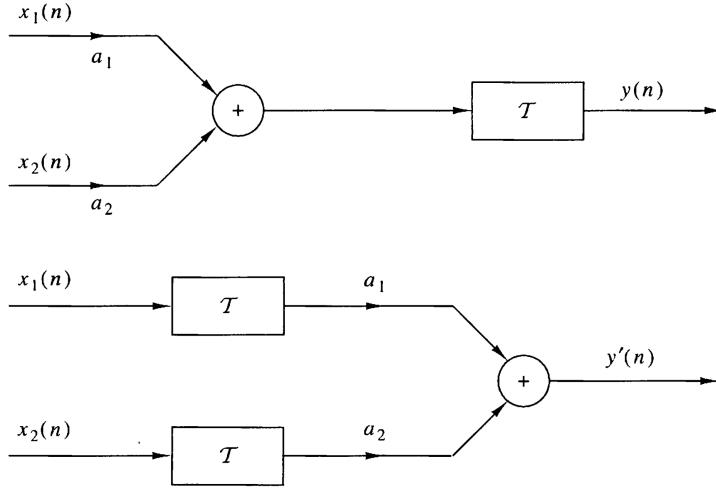


Abbildung 3: zeigt zwei verschiedene systemtheoretische Interpretationen von linearen Systemen. Quelle: [1]

3.3 Diskrete LTI-Systeme

Wir schränken nun die Menge der Systeme ein, die wir betrachten wollen, indem wir fordern, dass das System \mathcal{T} gleichzeitig linear und zeitinvariant ist. Das heißt, es gilt einerseits, dass ein Eingang $x[\cdot]$ zum System \mathcal{T} mit Ausgang $y[\cdot]$ bei Verzögerung zu $x[\cdot - k]$ den entsprechend verzögerten Ausgang $y[\cdot - k]$ zur Folge hat. Gleichzeitig kann der Ausgang des Systems \mathcal{T} für Eingänge, die lineare Superpositionen sind als lineare Superposition von den entsprechenden Ausgängen ausgedrückt werden, siehe Abbildung 3.

3.3.1 Faltungsformel

Wir wollen die Struktur von LTI-Systemen ausnutzen, um eine allgemeine und einfache Formel für das Eingangs-Ausgangsverhalten von jenen angeben zu können. Als Erstes verallgemeinern wir hierzu Abbildung 3 zu beliebigen, aber endlichen Summen, also gegeben ist ein Eingang $x[\cdot]$ der Form

$$x[n] = \sum_{k=1}^K a_k \cdot x_k[n], \quad (3.3.1)$$

wobei wir wissen, wie die Ausgänge von jedem $x_k[\cdot]$ zu berechnen sind, also

$$y_k[n] = (\mathcal{T}x_k[\cdot])[n].$$

Dann wissen wir wegen der Linearität und (3.3.1), dass

$$y[n] = \left[\mathcal{T} \left(\sum_{k=1}^K a_k \cdot x_k[\cdot] \right) \right] [n] = \sum_{k=1}^K a_k (\mathcal{T}x_k[\cdot])[n] = \sum_{k=1}^K a_k y_k[n] \quad (3.3.2)$$

gelten muss. Es lohnt sich einige Zeit über diese Sache zu meditieren und es gibt verschiedene Interpretationen.

- Wie bereits erwähnt passt dies zur linearen Struktur des Systems \mathcal{T} .

- Ist ein Eingangssignal aus anderen Signalen zusammengesetzt, dann setzt sich die Reaktion des Systems auf dieses zusammengesetzte Signal aus den Reaktionen auf die Signalbausteine zusammen. Wichtig ist hierbei, dass die Art der Zusammensetzung sich nicht ändert. Die a_k in (3.3.2) sind die gleichen, wie in (3.3.1).

Wir wollen nun einen Schritt weiter gehen und eine Menge von $x_k[\cdot]$ angeben, die es erlauben *alle* möglichen Signale darzustellen. Dazu betrachten wir, was geschieht, wenn wir den Einheitsstoß $\delta[\cdot]$ mit einem beliebigen Signal multiplizieren. Wir rechnen demzufolge für ein beliebiges Signal

$$x[n] \cdot \delta[n] = \begin{cases} x[0] & \text{für } n = 0 \\ 0 & \text{sonst.} \end{cases}$$

Wenn wir nun die Einheitsstöße verschieben um $k \in \mathbb{Z}$ erhalten wir

$$x[n] \cdot \delta[n - k] = \begin{cases} x[k] & \text{für } n = k \\ 0 & \text{sonst.} \end{cases}$$

Im Grunde „pickt“ $\delta[\cdot - k]$ bei Multiplikation den Wert von $x[\cdot]$ an der Stelle k heraus. Deshalb können wir nun schreiben

$$x[n] = \sum_{k \in \mathbb{Z}} x[k] \cdot \delta[n - k], \quad (3.3.3)$$

was nach Definition von $x_k[\cdot] = \delta[\cdot - k]$ genau die Form von (3.3.1) mit $a_k = x[k]$ annimmt. Die Kernbeobachtung ist nun, dass jedes $x_k[\cdot]$ eine verschobene Kopie von $\delta[\cdot]$ ist. Das heißt, dass wir nun die LTI-Eigenschaft ausnutzen können, weil (3.3.2) impliziert, dass wir nur $(\mathcal{T}\delta[\cdot])[\cdot]$ berechnen müssen und $(\mathcal{T}\delta[\cdot - k])[\cdot]$ sich als $(\mathcal{T}\delta[\cdot])(\cdot - k)$ ergibt.

Wir geben dem Kind nun einen Namen, also definieren wir $h : \mathbb{Z} \rightarrow \mathbb{C}$ als die Antwort des Systems \mathcal{T} auf den Eingang $\delta[\cdot]$, also

$$h[n] = (\mathcal{T}\delta[\cdot])[n]. \quad (3.3.4)$$

Man nennt $h[\cdot]$ die *Impulsantwort* des Systems. Dann können wir also mit (3.3.2) folgern, dass

$$y[n] = (\mathcal{T}x[\cdot])[n] = \sum_{k \in \mathbb{Z}} x[k]h[n - k] \quad (3.3.5)$$

geltet muss.

Das heißt, dass sich die Antwort $y[\cdot]$ eines diskreten LTI-Systems aus der *Faltung* des Einganges $x[\cdot]$ mit der Impulsantwort $h[\cdot]$ ergibt. Wir schreiben in Kurzform

$$y[n] = (x * h)[n] = (h * x)[n].$$

Es wird sich zeigen, dass sich viele Eigenschaften des Systems \mathcal{T} an oft einfacher zu prüfenden Eigenschaften der Impulsantwort $h[\cdot]$ ergeben. Das heißt, dass $h[\cdot]$ in gewisser Weise das System \mathcal{T} repräsentiert.

In Codeschnipsel 10 zeigen wir das Verhalten eines gleitenden Mittelwertes (*moving average*), welches sich durch

$$y[n] = \frac{1}{\ell} \sum_{k=0}^{k=\ell} x[n - k]$$

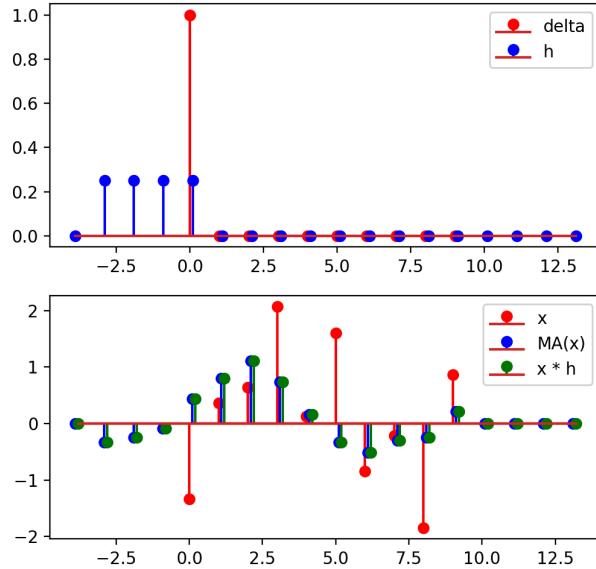
ergibt. Man sieht schön, dass durch das Mitteln die (in diesem Fall) zufällige Eingabe-Sequenz am Ausgang geglättet erscheint. Wichtig bei diesem Beispiel ist die Tatsache, dass wir direkt einsehen, dass Anwendung

```

def MA(x, l):
    xpad = np.pad(x, (l, l))
    y = np.zeros_like(xpad)
    for n in range(x.size + l):
        y[n] = np.mean(xpad[n : n + l])
    return y

delta = np.eye(N)[0, :]
h = MA(delta, l)
x = np.random.randn(N)
y1 = MA(x, l)
y2 = np.convolve(x, h[: 2 * l + 1])

```



Codeschnipsel 10: Gleitendes Mittel mit Länge $\ell = 4$. Wir vergleichen die direkte Berechnung mit der Berechnung über die Faltung, siehe [dsv/code/moving_average.py](#)

der direkten Formel für das gleitende Mittel aus einer Eingabe $x[\cdot]$ denselben Effekt hat, wie die Faltung mit $h[\cdot]$, das sich aus der Anwendung der Mittelung auf $\delta[\cdot]$ ergibt.

Es lohnt sich, sich einige Eigenschaften der Faltung zu merken. Diese sind:

- Bi-Linearität: Es gilt $(a_1x_1[\cdot] + a_2x_2[\cdot]) * h[\cdot] = a_1(x_1 * h)[\cdot] + a_2(x_2 * h)[\cdot]$. Dies ist die „normale“ Linearität in den Eingängen, die sich aus der Linearität des Systems \mathcal{T} ergibt. Es gilt aber auch $(x * (a_1h_1 + a_2h_2))[\cdot] = a_1(x * h_1)[\cdot] + a_2(x * h_2)[\cdot]$. Das heißt, wenn wir ein System $\mathcal{T} = a_1\mathcal{T}_1 + a_2\mathcal{T}_2$ gegeben haben, dann ist die Impulsantwort des Systems \mathcal{T} die gleiche Linearkombination der Impulsantworten $h_1[\cdot]$ und $h_2[\cdot]$ der beiden Systeme $\mathcal{T}_{1,2}$. Das heißt wiederum, dass LTI-Systeme selbst ein linearer Raum sind! Wir sprechen hier von *Bi*-Linearität, weil die Faltung eben linear in zwei Argumenten ist.
- Die Faltung ist assoziativ: Es gilt also, dass $(x * h_1) * h_2 = x * (h_1 * h_2)$. Dies impliziert, dass die Verkettung von zwei LTI-Systemen wieder ein LTI-system ergibt, wobei sich die Impulsantwort der Verkettung durch Faltung der beiden Impulsantworten der verketteten Systeme ergibt.
- Kommutativität: Es gilt $h_1 * h_2 = h_2 * h_1$, demnach auch, dass $x * (h_1 * h_2) = x * (h_2 * h_1)$. Das heißt erstaunlicherweise, dass man verkettete LTI-Systeme in ihrer Reihenfolge vertauschen kann, ohne das Eingangs-Ausgangsverhalten des Gesamtsystems zu beeinflussen.

In Codeschnipsel 11 untersuchen wir einige der oben genannten Eigenschaften. Einerseits sehen wir, dass die Impulsantwort von $MA(MA2(\cdot))$ mit der von $MA2(MA(\cdot))$ identisch ist. Wir haben also Kommutativität nachgeprüft. Wir sehen außerdem, dass sich die Impulsantwort der Verkettung aus Faltung der einzelnen Impulsantworten ergibt. Aus dem abgetasteten „Rechteck“ wird nach nochmaliger Anwendung ein abgetastetes, aber breiteres, „Dreieck“, was schlussendlich zu einer abgetasteten, stückweise quadratischen, Impulsantwort wird. Generell kann man das komplette System als eine dreifache Verkettung gleitender

```

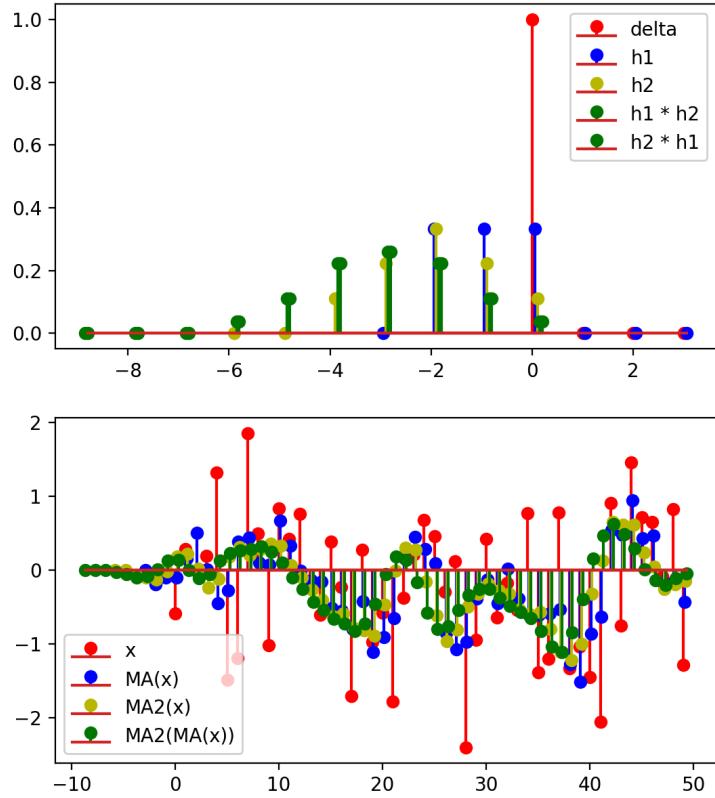
def MA(x, l):
    xpad = np.pad(x, (l, l))
    y = np.zeros_like(xpad)
    for n in range(x.size + l):
        y[n] = np.mean(xpad[n : n + l])
    return y[:-l]

def MA2(x, l):
    return MA(MA(x, l), l)

delta = np.eye(N)[0, :]
h1 = MA(delta, l)
h2 = MA2(delta, l)
h3 = MA2(MA(delta, l), l)
h4 = MA(MA2(delta, l), l)

x = np.random.randn(N)
y1 = MA(x, l)
y2 = MA2(x, l)
y3 = MA(MA2(x, l), l)

```



Codeschnipsel 11: Verkettung von mehreren Moving Averages der Länge $\ell = 3.$, siehe [dsv/code/ramp_ma.py](#)

Mittel der Länge $\ell = 3$ verstehen. Außerdem bestätigt Codeschnipsel 11 bei Vergleich der verschiedenen Ausgänge, dass wiederholtes Mitteln am Ausgang mit Anzahl der Mittelungen zunehmend „glattere“ Signale erzeugt.

3.3.2 Eigenschaften von LTI-Systemen

BIBO-Stabilität Wir hatten vorher schon diesen Begriff der Stabilität eingeführt und formulieren nun eine Bedingung an $h[\cdot]$, die es uns erlaubt auf Stabilität zu prüfen. Wir nennen die Eingabe beschränkt, falls ein $M_x < \infty$ existiert, sodass

$$|x[n]| \leq M_x \quad \text{für alle } n \in \mathbb{Z}$$

erfüllt ist. Dann können wir mit der Faltungsformel (3.3.5) berechnen, dass

$$|y[n]| = \left| \sum_{k \in \mathbb{Z}} x[k]h[n-k] \right| \leq \sum_{k \in \mathbb{Z}} |x[k]h[n-k]| \leq M_x \sum_{k \in \mathbb{Z}} |h[n-k]|$$

gilt. Damit nun $|y[n]| < \infty$, muss also gelten, dass

$$\sum_{k \in \mathbb{Z}} |h[n-k]| < \infty.$$

Das heißt, wenn $h[\cdot]$ absolut summierbar ist, dann ist das System mit $h[\cdot]$ als Impulsantwort **BIBO-stabil**.

Wie ist es um die umgekehrte Schlussfolgerung bestellt? Impliziert **BIBO-Stabilität**, dass $h[\cdot]$ absolut summierbar sein muss? Erinnern wir uns daran, dass man **BIBO-Stabilität** widerlegen kann, indem man *einen* beschränkten Eingang $x[\cdot]$ findet, für welchen der Ausgang $y[\cdot]$ nicht beschränkt bleibt. Nehmen wir an, dass

$$\sum_{k \in \mathbb{Z}} |h[n-k]| = \infty$$

und betrachten wir den Eingang

$$x[n] = \begin{cases} \frac{h[-n]^*}{|h[-n]|}, & \text{für } h[n] \neq 0 \\ 0 & \text{sonst.} \end{cases}$$

Man sieht, dass $|x[n]| \leq 1$, es ist also beschränkt. Dann berechnen wir einfach den ersten Wert am Ausgang mit (3.3.5) und der Definition des Einganges $x[\cdot]$, also

$$y[0] = \sum_{k \in \mathbb{Z}} x[-k]h[k] = \sum_{k \in \mathbb{Z}} \frac{|h[k]|^2}{|h[-k]|} = \sum_{k \in \mathbb{Z}} |h[k]| = \infty,$$

wobei das letzte Gleichheitszeichen gilt, weil wir angenommen hatten, dass $h[\cdot]$ nicht absolut summierbar ist. Man kann also schlussfolgern, dass sich Stabilität vollständig durch die Impulsantwort $h[\cdot]$ bestimmen lässt. Eine tolle Sache!

FIR vs. IIR Gilt für die Impulsantwort $h[\cdot]$, dass

$$h[n] = 0, \quad \text{für } n < m, M \leq n$$

für zwei Zahlen $m \leq M$, dann bezeichnet man das zugehörige System \mathcal{T} als **Finite Impulse Response (FIR)**-System. Ist obige Bedingung nicht erfüllt, so nennt man \mathcal{T} ein **Infinite Impulse Response (IIR)**-System.

Bei **FIR**-Systemen sind für die Berechnung von $y[n]$ nur endlich viele Werte, maximal $M - m$ viele, notwendig. Das System hat also ein endliches Gedächtnis der Länge $M - m$, wobei **IIR**-Systeme ein unendlich langes Gedächtnis haben. Sobald bei einem **FIR**-System $|h[n]| < \infty$ für alle $n \in \mathbb{Z}$ gilt, ist es auch stabil – also sind in der Praxis *alle* **FIR**-Systeme stabil.

Bei einem **IIR**-System ist die Frage nach Stabilität nicht so einfach zu beantworten, doch wir werden im Folgenden noch ein Werkzeug kennenlernen, mit welchem dies einfach nachzuprüfen sein wird. Außerdem haben **IIR**-Systeme noch das Problem, dass man die Faltungsformel (3.3.5) nicht nutzen kann, um ein **IIR**-System zu implementieren.

3.3.3 Rekursive IIR-Systeme

Um die Notwendigkeit von (3.3.5) zu umgehen, betrachten wir eine gewisse Klasse von Systemen, deren Ausgang $y[\cdot]$ auch über einen alternativen Weg bestimmt werden kann. Hierzu betrachten wir

$$y[n] = \frac{1}{n+1} \sum_{k=0}^n x[k],$$

was ein kumulatives Mittel von 0 bis n darstellt. So wie es hier scheint, muss man für die Berechnung von $y[n]$ alle vergangenen Werte von $x[\cdot]$ bereitliegen haben. Doch mit einer einfachen Umformung findet man, dass

$$(n+1)y[n] = ny[n-1] + x[n] \Leftrightarrow y[n] = \frac{n}{n+1}y[n-1] + \frac{1}{n+1}x[n].$$

Wir können also alternativ $y[n]$ aus $x[n]$ und $y[n-1]$ berechnen. Da also $y[n]$ von $y[n-1]$ abhängt, nennen wir das System *rekursiv*. Man sieht deutlich, dass diese Umformulierung deutlich macht, dass wir nur $y[n-1]$ im Speicher halten müssen und dieses mit einer Art *zeitverzögertem Feedback* ausstatten müssen. Hierbei ist natürlich die Zeitverzögerung essenziell, denn müssen wir $y[n]$ in Abhängigkeit von $y[n]$ berechnen, würde uns das vor ein halbwegs unlösbares Problem stellen. In Codeschnipsel 12 sehen wir die beiden unterschiedlichen Implementierungen. Es ist hier zu beachten, dass $y[nn] = np.sum(x[:nn]) / (nn + 1)$ immer auf die komplette Vergangenheit von $x[\cdot]$ zugreift und eben *nicht* auf Werte von $y[\cdot]$.

Für rekursive Systeme entsteht noch das Problem, dass wir beispielsweise für den Beginn der rekursiven Berechnung von $y[n]$ ab einem gewissen $n_0 \in \mathbb{Z}$ den Wert $y[n_0 - 1]$ kennen müssen. Das heißt wir benötigen einen *Startwert* für die Rekursion. Dieser beeinflusst auch maßgeblich das Verhalten des Systems.

Eine Möglichkeit, wie man das produktiv ausnutzen kann, ist für die Berechnung der Quadratwurzel einer positiven reellen Zahl A . Für die Herleitung definieren wir erst die Funktion $f : \mathbb{R} \rightarrow \mathbb{R}$ via

$$f(x) = x^2 - A.$$

Wie man leicht sieht, hat diese Funktion Nullstellen $x_{1,2} = \pm\sqrt{A}$. Die sogenannte Newton-Iteration ⁴ berechnet iterativ via

$$y[n] = y[n-1] - \frac{f(y[n-1])}{f'(y[n-1])}$$

eine Nullstelle der Funktion f . Setzen wir unsere Funktion ein und definieren $x[\cdot] = Au[\cdot]$, dann erhalten wir

$$y[n] = \frac{1}{2} \left(y[n-1] + \frac{x[n]}{y[n-1]} \right)$$

```

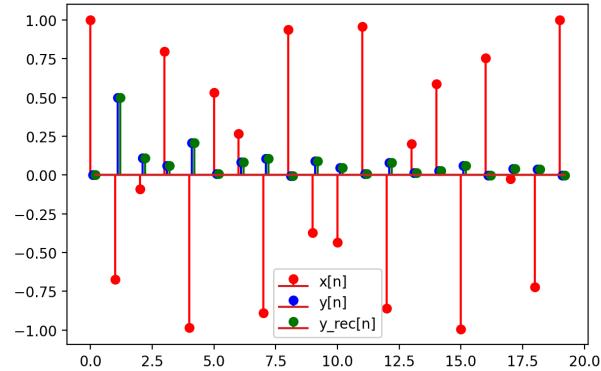
def cumulative_sum(
    x: np.ndarray, n: np.ndarray
) -> np.ndarray:
    y = np.zeros(n.size)
    for nn in n:
        y[nn] = np.sum(x[:nn]) / (nn + 1)

    return y

def cumulative_sum_rec(
    y: float, x: float, n: int
):
    return (n / (n + 1.0)) * y + 1.0 / (
        n + 1
    ) * x

```

Codeschnipsel 12: Die beiden möglichen Implementierungen eines kumulativen Mittelwertes, siehe [dsv/code/cumulative_sum.py](#)



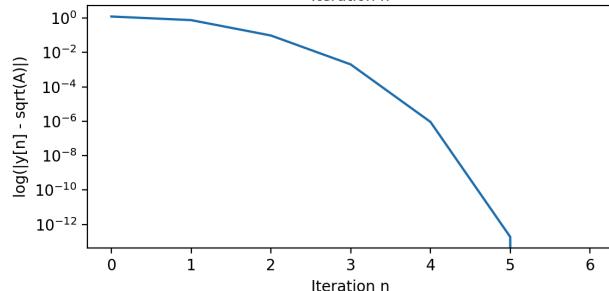
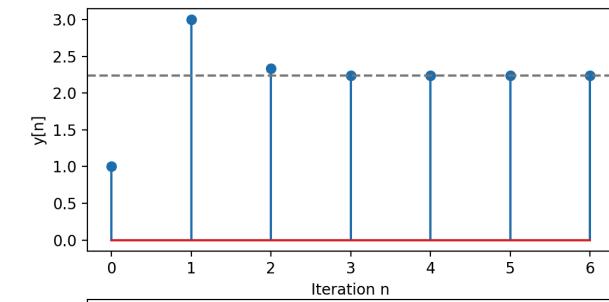
```

def root(A: float, y0: float, n):
    y = [y0]
    for ii in range(n):
        y.append((y[-1] + A / y[-1]) / 2)

    return y

A = 5.0
rootA = np.sqrt(A)
y = root(A, 1.0, 6)

```



Codeschnipsel 13: Iteratives Verfahren zur Berechnung von \sqrt{A} , siehe [dsv/code/square_root.py](#)

Nutzen wir einen geeigneten Startwert, beispielsweise $y[-1] = 1$, so erhalten wir eine *sehr schnell* konvergierende Folge von $y[n]$.

Die Folge konvergiert so schnell, dass bereits nach 5 Iterationen der Fehler an der endlichen Genauigkeit von Gleitkommaarithmetik kratzt. Es ist auch darauf hinzuweisen, dass das System nur sehr einfache arithmetische Operationen benutzt, um die numerisch nicht ganz triviale Berechnung von \sqrt{A} beliebig genau und sehr schnell zu approximieren.

Abschließend ist noch anzumerken, dass man rekursive Systeme auch „lösen“ kann, siehe [1, Kap. 2.4.3], indem man aus der rekursiven Vorschrift eine explizite entwickelt. Die dort entwickelte Theorie erinnert der Lösung von Differenzialgleichungen nach Funktionen, nur werden stattdessen Differenzengleichungen nach diskreten Folgen gelöst.

3.3.4 Approximation von IIR-Systemen durch FIR-Systeme

Um die Faltungsformel doch anwenden zu können, kann man für ein *stabiles* IIR-System mit Impulsantwort $h[\cdot]$ eine FIR-Approximation finden. Ein weiterer Vorteil ergibt sich dann, dass auch dieses resultierende System stabil sein muss. Für $n_{\text{low}} < n_{\text{high}}$ definieren wir

$$h_{\text{FIR}}[n] = \begin{cases} h[n], & \text{falls } n_{\text{low}} \leq n \leq n_{\text{high}}, \\ 0 & \text{sonst.} \end{cases}$$

Da aus der Stabilität von dem System mit Impulsantwort $h[\cdot]$ folgt, dass

$$\lim_{k \rightarrow \infty} \sum_{n=k}^{\infty} (|h[n]| + |h[-n]|) = 0,$$

da sonst

$$\sum_{n \in \mathbb{Z}} |h[n]|$$

nicht endlich sein könnte. Das heißt, dass die Werte der Impulsantwort $h[\cdot]$ gegen 0 konvergieren müssen und deshalb kann man $n_{\text{low}} < n_{\text{high}}$ so wählen, dass der Unterschied zwischen $h[\cdot]$ und $h_{\text{FIR}}[\cdot]$ beliebig klein wird.

Je nachdem wie schnell $h[\cdot]$ gegen 0 konvergiert, benötigen wir mehr oder von 0 verschiedene Einträge in h_{FIR} , was sich auf den Implementierungsaufwand auswirkt.

Ein Beispiel für einen exponentiellen Mittelwert-Filter findet man in Codeschnipsel 14. Man sieht, dass für $\alpha = 0.5$ der Abschneidefehler deutlicher zutage tritt, da die Faltung mit $h_{\text{FIR}}[\cdot]$ in diesem Fall ein deutlich schlechteres Ergebnis liefert als im Falle von $\alpha = 0.1$. Dies liegt eben daran, dass bei $\alpha = 0.1$ der IIR-Filter effektiv ein FIR-Filter wird. Wir wollen aber darauf hinweisen, dass es eine ganz eigene Theorie für die Approximation von Filtern durch andere Filter gibt, die wir hier nicht einmal anreißen wollen.

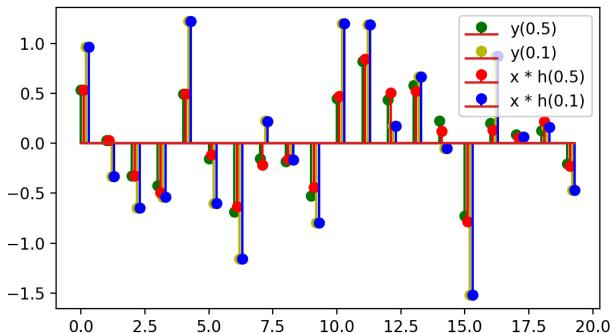
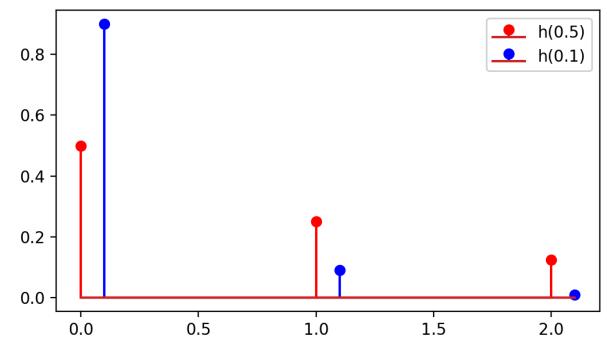
⁴https://encyclopediaofmath.org/index.php?title=Newton_method

```

def exp_mean(
    x: np.ndarray, alpha: float
) -> np.ndarray:
    y = np.zeros_like(x)
    y[0] = 0
    for ii in range(1, y.size):
        y[ii] = (
            alpha * y[ii - 1]
            + (1 - alpha) * x[ii]
        )
    return y

alpha1 = 0.5
alpha2 = 0.1
N = 20
L = 3
n = np.arange(N)
l = np.arange(L)
delta = np.eye(L)[0]
h1 = exp_mean(
    np.pad(delta, (L, L), mode="constant"),
    alpha1,
)[L:-L]
h2 = exp_mean(
    np.pad(delta, (L, L), mode="constant"),
    alpha2,
)[L:-L]

```



Codeschnipsel 14: Approximation eines exponentiellen Mittelwertes, also einem IIR-System, durch ein FIR-System, siehe [dsv/code/exp_mean.py](#)

3.4 z-Transformation

Wir wollen nun eines der wichtigsten Werkzeuge für die Analyse von diskreten LTI-Systemen einführen und untersuchen. Die z-Transformation nimmt für diskrete LTI-Systeme dieselbe Rolle ein, wie die Laplace-Transformation für zeitstetige LTI-Systeme. Wir werden sehen, dass die Faltungs-Formel (3.3.5) im z-Bereich eine einfachere Form annimmt. Außerdem erlaubt uns die Transformation eine neue Sicht auf diskrete Systeme zu gewinnen, mit welcher es eventuell einfacher sein sollte gewisse Eigenschaften von Systemen nachzuprüfen. Wir werden auch sehen, dass sich manche Signale oder Systeme im z-Bereich kompakter aufschreiben lassen.

Die z-Transformation $X_z : \mathbb{C} \rightarrow \mathbb{C}$ eines Signals $x[\cdot] : \mathbb{N} \rightarrow \mathbb{C}$ ist die komplexe Potenzreihe⁵

$$z \mapsto X_z(z) = \sum_{n \in \mathbb{Z}} x[n]z^{-n}. \quad (3.4.1)$$

Wegen der Summation über ganz \mathbb{Z} ist $X_z(z)$ für jedes z ein Grenzwert, der nicht existieren muss. Das heißt, es ist notwendig auch *immer* den Konvergenzbereich, oder Englisch die **Region of Convergence (ROC)**, von X_z anzugeben.

Betrachten wir beispielsweise

$$x[\cdot] = \begin{cases} 5, 6, 3, 4, -\pi \\ \uparrow \end{cases},$$

dann ergibt sich

$$X_z(z) = 5z^2 + 6z + 3 + 4z^{-1} - \pi z^{-2} \quad \text{mit ROC } \mathbb{C} \setminus \{0\}.$$

Im Fall von $x[\cdot] = \delta[\cdot - k]$, ergibt sich $X_z(z) = z^{-k}$ mit ROC $\mathbb{C} \setminus \{0\}$, wobei sich bei $x[\cdot] = \delta[\cdot + k]$, sich $X_z(z) = z^k$ mit ROC \mathbb{C} ergibt. Man sieht nun auch, weshalb das System, das ein Signal um einen Abtastwert verzögert, oft mit z^{-1} bezeichnet wird, weil dies die z-Transformation dessen Impulsantwort ist. Außerdem kann man bei endlichen Signalen im z-Bereich deren Werte im Zeitbereich einfach ablesen. Will man den Wert $x[n]$, also an Sample n , ermittelt man diesen, indem man den Koeffizienten vom passenden z^{-n} betrachtet.

Aber auch für unendlich lange Signale, wie beispielsweise $x[n] = a^n u[n]$, ergibt sich

$$X_z(z) = 1 + az^{-1} + a^2z^{-2} + \dots = \sum_{n=0}^{\infty} a^n z^{-n} = \sum_{n=0}^{\infty} (az^{-1})^n,$$

was eine geometrische Reihe über az^{-1} darstellt. Dann folgt, dass

$$X_z(z) = \frac{1}{1 - az^{-1}},$$

wobei diese Potenzreihe nur konvergiert, wenn $z > a$. Das heißt, dass die ROC hier alle $z \in \mathbb{C}$ umfasst mit $|z| > |a|$. Wir sehen, dass sich eigentlich unendlich lange Signale im z-Bereich durch Berechnen des Grenzwertes knapper darstellen lassen. Gleichzeitig können wir so auch die ROC bestimmen, indem wir ermitteln, wann der ermittelte Grenzwert existiert.

Wenn wir $z = r \exp(j\phi)$ in die Definition (3.4.1) einsetzen und $|X_z(z)|$ betrachten, finden wir

$$|X_z(z)| = \left| \sum_{n \in \mathbb{Z}} x[n]z^{-n} \right| \leqslant \sum_{n \in \mathbb{Z}} |x[n]z^{-n}| = \sum_{n \in \mathbb{Z}} |x[n]r^{-n}| |\exp(-jn\phi)| = \sum_{n \in \mathbb{Z}} |x[n]r^{-n}|.$$

⁵<https://mathworld.wolfram.com/LaurentSeries.html>

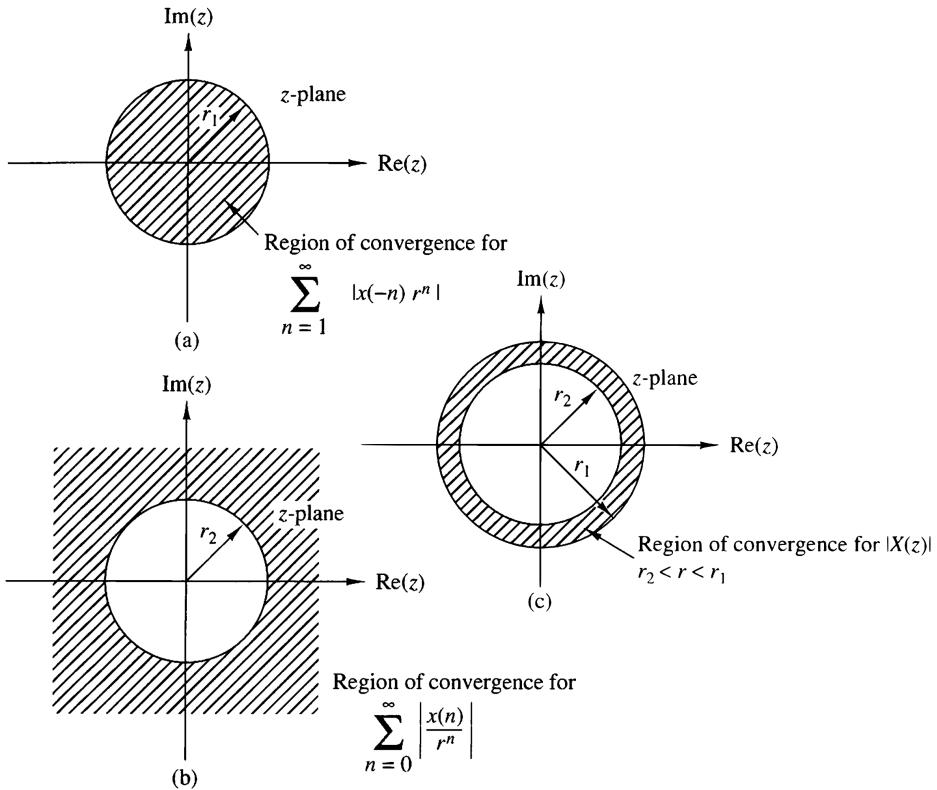


Abbildung 4: Veranschaulichung der Entstehung der ROC aus den Konvergenzkriterien. Quelle: [1]

Betrachten wir den letzten Ausdruck genauer indem wir ihn in zwei Summationen aufteilen, also

$$\sum_{n \in \mathbb{Z}} |x[n]r^{-n}| = \sum_{n=-1}^{-\infty} |x[n]r^{-n}| + \sum_{n=0}^{+\infty} |x[n]r^{-n}| = \sum_{n=1}^{+\infty} |x[-n]r^n| + \sum_{n=0}^{+\infty} \left| \frac{x[n]}{r^n} \right|.$$

Das ergibt dann also, dass

$$|X_z(z)| \leq \sum_{n=1}^{+\infty} |x[-n]r^n| + \sum_{n=0}^{+\infty} \left| \frac{x[n]}{r^n} \right|$$

Damit X_z existiert, müssen beide Grenzwerte existieren. Der erste Grenzwert existiert, wenn es ein r_1 gibt, sodass r_1^n die Werte $x[-n]$ so gewichtet, dass sich Konvergenz einstellt. Dieses r_1 ist ausschließlich durch die Werte in der „Vergangenheit“ von $x[\cdot]$ bestimmt! Wenn sich Konvergenz für ein r_1 einstellt, dann auch für alle $r < r_1$. Der Konvergenzbereich ergibt sich also als ein Kreis mit Radius r_1 . Der zweite Grenzwert existiert, falls für ein $r_2 > 0$ die Werte $1/r_2^n$ so die „Zukunft“ von $x[n]$ so wichten, dass die Summe konvergiert. Das heißt, falls so ein r_2 existiert, dann konvergiert der zweite Summand für alle $r > r_2$.

Die **ROC** ist also die Schnittmenge von der Menge $\{|z| \leq r_1\}$ mit der Menge $\{|z| \geq r_2\}$. Dies ist in Abbildung 4 zusammengefasst. Wir sehen auch, dass im Falle von $r_1 > r_2$ die **ROC** die leere Menge ist und deshalb die z -Transformation nicht existiert.

3.4.1 Eigenschaften

Wir wollen nun einige Eigenschaften der z -Transformation auflisten, die wir immer wieder verwenden werden. Besonderes Augenmerk legen wir hierbei auf die Behandlung der **ROC**.

Linearität Gegeben zwei diskrete Signale $x_{1,2}[\cdot]$ und komplexe Zahlen $a_{1,2} \in \mathbb{C}$, dann gilt für $x[\cdot] = a_1 x_1[\cdot] + a_2 x_2[\cdot]$, dass

$$X_z(z) = a_1 X_{z,1}(z) + a_2 X_{z,2}(z),$$

wobei die **ROC** von X_z der Schnittmenge der **ROCs** von $X_{z,1}$ und $X_{z,2}$ entspricht.

Zeitversatz Ist X_z die z -Transformation von $x[\cdot]$, so ist $z^{-k} X(z)$ die z -Transformation von $x[\cdot - k]$. Hierbei ist die **ROC** von X_z , falls $k \leq 0$. Im Falle $k > 0$ müssen wir aus der **ROC** von X_z den Wert $z = 0$ entfernen. Intuitiv sollte man diese Eigenschaft leicht verifizieren können, da man durch das Verschieben nur die Potenz des z ändert und man diese Änderung einfach ausklammern kann.

Um beispielsweise die z -Transformation von

$$x[n] = \begin{cases} 1, & \text{für } 0 \leq n < N \\ 0 & \text{sonst} \end{cases}$$

zu bestimmen, könnten wir einerseits die Definition bemühen, oder die beiden schon bekannten Eigenschaften ausnutzen, um zu folgern, dass

$$X_z(z) = \begin{cases} N & \text{für } z = 1, \\ \frac{1-z^{-N}}{1-z^{-1}} & \text{sonst} \end{cases}$$

gilt. Dazu nutzen wir, dass $x[n] = u[n] - u[n - N]$ gilt. Da außerdem $x[\cdot]$ nur endlich viele Werte verschieden von 0 hat, ist die **ROC** von X_z die ganze z -Ebene, außer $z = 0$.

Skalierung im z -Bereich Ist X_z die z -Transformation von $x[\cdot]$ mit **ROC** $r_1 < |z| < r_2$, dann ist $a^n x[n]$ die inverse z -Transformation von $X(a^{-1}z)$ mit **ROC** $|a|r_1 < |z| < |a|r_2$. Die Intuition hierbei ist, dass die Multiplikation mit der Folge a^n mit $a = r \exp(j\omega)$ eine Skalierung um r und eine Rotation um ω im z -Bereich bewirkt. Das heißt gewisse „geometrische“ Eigenschaften von X_z transformieren sich analog mit.

Zeitumkehrung Ist X_z die z -Transformation von $x[\cdot]$ mit **ROC** $r_1 < |z| < r_2$, dann ist $X_z(z^{-1})$ die z -Transformation von $x[-\cdot]$ mit **ROC** $1/r_1 < |z| < 1/r_2$. Mit anderen Worten entspricht Umkehrung im Zeitbereich Reflexion am Einheitskreis im z -Bereich.

Faltung im Zeitbereich Kommen wir nun zu einer der wichtigsten Eigenschaften und einem der Gründe, weshalb die z -Transformation als Analogie zur Laplace-Transformation verstanden wird. Gegeben zwei diskrete Signale $x_{1,2}[\cdot]$ und $x[\cdot] = x_1 * x_2$, dann gilt

$$X_z(z) = X_{z,1}(z) \cdot X_{z,2}(z), \quad (3.4.2)$$

wobei die **ROC** mindestens die Schnittmenge der **ROCs** von $X_{z,1}(z)$ und $X_{z,2}(z)$ ist. Der Grund für das „mindestens“ ist die Tatsache, dass für gewisse z die Divergenz von $X_{z,1}$ durch rasches Tendieren von $X_{z,2}$ gegen 0 kompensiert werden kann.

Wegen ihrer Bedeutung, wollen wir uns kurz überlegen, warum die obige Eigenschaft gilt. Wir nutzen im Grunde nur die Definition (3.4.1) und die Definition der Faltung, um zu sehen, dass

$$X_Z(z) = \sum_{n \in \mathbb{Z}} x[n]z^{-n} = \sum_{n \in \mathbb{Z}} \sum_{k \in \mathbb{Z}} x_1[k]x_2[n-k]z^{-n} = \sum_{k \in \mathbb{Z}} x_1[k] \sum_{n \in \mathbb{Z}} x_2[n-k]z^{-n}.$$

Jetzt nutzen wir die Eigenschaft, die sich aus dem Zeitversatz gibt zusammen mit (3.4.1), und erhalten schlussendlich

$$X_Z(z) = \sum_{k \in \mathbb{Z}} x_1[k] \sum_{n \in \mathbb{Z}} x_2[n-k]z^{-n} = \sum_{k \in \mathbb{Z}} x_1[k]X_{Z,2}(z)z^{-k} = X_{Z,2}(z) \sum_{k \in \mathbb{Z}} x_1[k]z^{-k} = X_{Z,1}(z)X_{Z,2}(z).$$

Wir können also zusammenfassend folgende Herangehensweise bei der Berechnung der Faltung angeben:

1. Transformation von $x_{1,2}[\cdot]$ zu $X_{Z,1,2}$
2. Punkweise Multiplikation von $X_Z(z) = X_{Z,1}(z)X_{Z,2}(z)$
3. Rücktransformation von $X_Z(z)$ zu $x[\cdot] = x_1 * x_2$

Inverse z-Transformation Obige Berechnung der Faltung verlangt im letzten Schritt die Invertierung der z-Transformation von X_Z zu $x[\cdot]$. Im allgemeinen Fall ist man darauf angewiesen den Cauchyschen Integralsatz⁶ zu bemühen. Am Ende erlaubt dieser die Herleitung von

$$x[n] = \frac{1}{j2\pi} \oint_{C \subset \text{ROC}} X_Z(z)z^{n-1} dz \quad (3.4.3)$$

als Möglichkeit der Bestimmung von $x[n]$ aus X_Z . Oft sind diese Integrale jedoch nicht leicht zu berechnen und man sollte sich auf geeignete Tabellen (siehe [1, Tabelle 3.2, Tabelle 3.3]) und die restlichen obigen Eigenschaften beziehen, wenn man z-Transformationspaare bestimmen möchte.

3.4.2 Anwendung bei diskreten LTI-Systemen

Wir kombinieren nun (3.3.5) mit (3.4.2), um eine alternative Beschreibung von LTI-Systemen zu erhalten. Wir wissen aus (3.3.5), dass sich das Eingabe-Ausgabe-Verhalten von einem LTI-System \mathcal{T} durch

$$y[\cdot] = h * x$$

ausdrücken lässt, wobei $h[\cdot] = y[\cdot]$ für $x[\cdot] = \delta$. Im z-Bereich transformiert sich dies wegen (3.4.2) zu

$$Y_Z(z) = H_Z(z)X_Z(z). \quad (3.4.4)$$

Man nennt dann H_Z als z-Transformation der Impulsantwort die *Transferfunktion* des Systems \mathcal{T} . Wir haben so eine alternative Sichtweise auf LTI-Systeme gewonnen, weil man eben durch die z-Transformation die Impulsantwort $h[\cdot]$ mit der Transferfunktion H identifiziert. Das heißt, dass beide Beschreibungen äquivalent sind. Wir werden uns wie in Abschnitt 3.3.2 überlegen, wie man gewissen Eigenschaften von LTI-Systemen an H „ablesen“ kann.

Aus diesem multiplikativen Zusammenhang kann man bei Kenntnis von zwei Größen in der Regel die fehlende Dritte bestimmen. Kennt man beispielsweise H_Z und X_Z , dann kann man sehr einfach Y_Z und damit

⁶https://de.wikipedia.org/wiki/Cauchyscher_Integralsatz

auch $y[\cdot]$ bestimmen. In bestimmten Anwendungen der Messtechnik ist man eher daran interessiert durch geschickte Messungen $h[\cdot]$ zu bestimmen. Dies kann unter anderem interessant sein, wenn in $h[\cdot]$ gewisse Parameter eines Systems „versteckt“ sind, die einen aber interessieren. Dann ist es oft geschickt das System durch $x[\cdot]$ anzuregen, $y[\cdot]$ zu beobachten und dann die Transferfunktion durch

$$H_z(z) = \frac{Y_z(z)}{X_z(z)}$$

zu bestimmen. Man muss sich im Klaren hierbei sein, dass es hierzu notwendig ist das Signal $x[\cdot]$ so zu wählen, dass obige Division ein hinreichend genaues Bild von $h[\cdot]$ erlaubt.

Rekursive Systeme Ist allgemein die Vorschrift von einem System gegeben durch

$$y[n] = -\sum_{k=1}^N a_k y[n-k] + \sum_{k=0}^M b_k x[n-k],$$

dann ergibt sich durch die Linearität, Zeitversatz und geschicktes Umstellen, dass

$$H(z) = \frac{\sum_{k=0}^M b_k z^{-k}}{1 + \sum_{k=1}^N a_k z^{-k}}.$$

Das heißt, dass Systeme, die durch lineare Differenzengleichungen charakterisiert sind, besitzen eine Transferfunktion, die die Struktur einer rationalen Funktion

$$H_z(z) = \frac{B(z)}{A(z)}$$

hat, wenn wir fordern, dass A und B Polynome in z sind. Allgemein schreiben wir

$$H(z) = \frac{b_0 + b_1 z^{-1} + \dots + b_M z^{-M}}{a_0 + a_1 z^{-1} + \dots + a_N z^{-N}} = \frac{\sum_{k=1}^M b_k z^{-k}}{\sum_{k=1}^N a_k z^{-k}} = G z^{N-M} \frac{(z - z_1) \dots (z - z_M)}{(z - p_1) \dots (z - p_N)}.$$

Dann nennen wir die z_1, \dots, z_M die „Nullstellen“, oder *zeros*, des Systems \mathcal{T} und die Menge p_1, \dots, p_N die *Pole*, oder *Polstellen*, von \mathcal{T} . Tritt ein p_i oder z_i mehrfach auf, so sprechen wir von einer mehrfachen Polstelle, bzw. Nullstelle. Die Anzahl an gleichen Null-/Polstellen nennt man ihre Vielfachheit.

Es gibt nun zwei Spezialfälle, die wir betrachten können. Erstens, es gilt $a_1 = \dots = a_N = 0$. In diesem Fall gilt

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

und H besitzt als Polynom vom Grad M dann M Nullstellen⁷ und eine M -fache Polstelle bei $z = 0$. Tritt eine Polstelle bei $z = 0$ auf, so nennen wir diese *trivial*, da sie keinen Einfluss auf das System hat. Schließlich korrespondiert z^{-M} nur zu einer Verschiebung im Zeitbereich, was bei LTI-Systemen per definitionem nur eine globale Verschiebung der ganzen Systems bedeutet. Da keine nicht-trivialen Polstellen auftreten, hat das

⁷https://de.wikipedia.org/wiki/Fundamentalsatz_der_Algebra

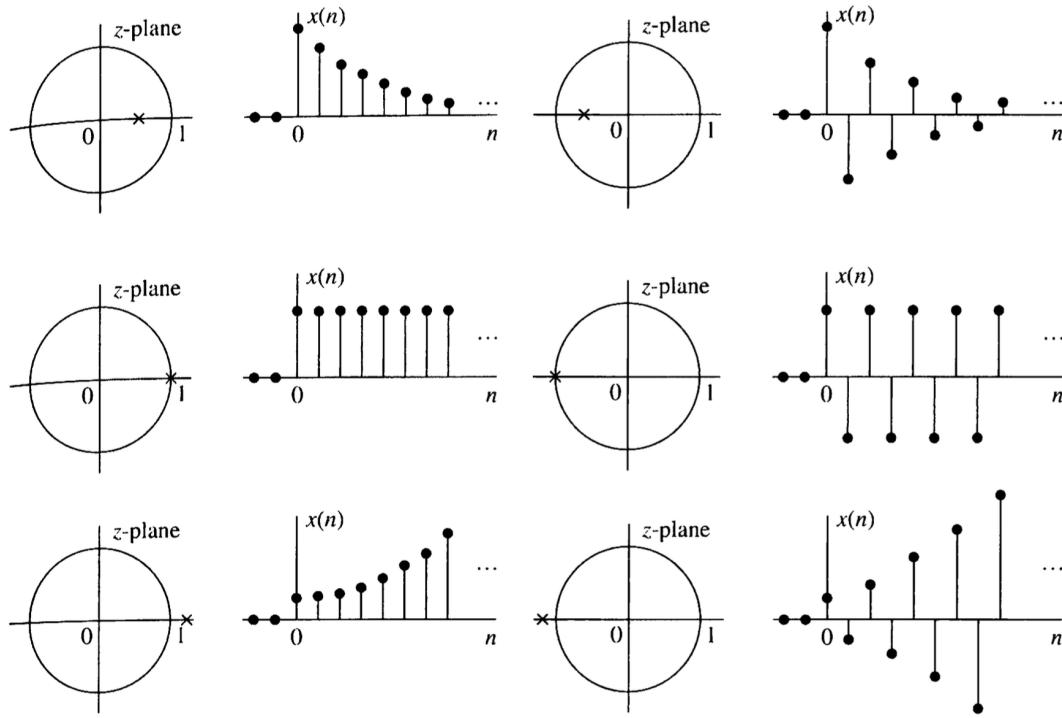


Abbildung 5: Impulsantwort $h[n] = a^n u[n]$ für verschiedene Werte von a , Quelle: [1]

System eine *endliche* Impulsantwort. Das können wir daran sehen, dass wir die Werte von $h[\cdot]$ direkt an den Koeffizienten von H ablesbar sind.

Zweitens, kann es vorkommen, dass $b_1 = \dots = b_N = 0$, was wiederum H zu einer rationalen Funktion macht, die keine Nullstellen und dementsprechend ausschließlich Polstellen besitzt. Um zu bestimmen, welche Implikation dies hat, betrachten wir ein einfaches System mit einem reellen Pol bei $a \in \mathbb{R}$, also

$$H(z) = \frac{1}{1 - az^{-1}}, \text{ ROC : } |z| > |a|.$$

Wie wir weiter oben schon gesehen haben, ist die zugehörige Impulsantwort $h[\cdot]$ gegeben durch

$$h[n] = a^n u[n].$$

Das Verhalten von diesem System ist in Abb. 5 dargestellt. Man sieht, dass der Einheitskreis in der komplexen Ebene eine entscheidende Rolle spielt. Liegt a innerhalb des Einheitskreises, so ergibt sich eine exponentiell abfallende Impulsantwort. Falls $a < 0$, stellt sich auch eine Oszillation ein. Für $|a| = 1$ entspricht der Betrag der Impulsantwort der Heavyside-Funktion $u[\cdot]$. Man sieht, dass im Fall von $|a| < 1$ die Impulsantwort absolut summierbar ist, also

$$\sum_{n \in \mathbb{Z}} |h[n]| < \infty,$$

was impliziert, dass das System stabil ist. Entsprechend, ist das System instabil, falls $|a| > 1$. Man sieht also, dass die Position der Pole im z -Bereich direkt interpretierbar ist. Betrachtet man nochmals die **ROC** $|z| > |a|$,

zusammen mit der Information, dass sich Stabilität einstellt, falls $|a| < 1$. Geometrisch bedeutet dies, dass das System stabil ist, falls der Einheitskreis in der **ROC** enthalten ist.

Diese Resultate lassen sich auf mehrere Polstellen verallgemeinern, da H nur mehr Pole enthält und die **ROC** dann sich nach dem Betrag der „größten“ Polstelle richtet. Das heißt ultimativ, dass sich für ein System H Stabilität einstellt, falls *alle* Polstellen sich im Einheitskreis befinden, *oder* äquivalent, dass sich der Einheitskreis in der **ROC** befinden muss. Später werden wir noch eine Bedingung für Stabilität in Bezug auf die Fourier-Transformation finden.

4 Fourier-Transformation

Wir wollen unsere Werkzeuge zur Analyse von Signalen und Systemen nun um das wahrscheinlich wichtigste erweitern. Hierbei zerlegen wir Signale, beziehungsweise Systemantworten/Impulsantworten, in ihre „harmonischen“ Anteile – wir transformieren in den *Frequenzbereich*. Man diese Art der *Analyse* auch oft *Fourier-Analyse*. Da wir verschiedene Arten von Signalen bereits kennengelernt haben, muss diese Zerlegung auch auf verschiedene Weisen durchgeführt werden. Für diskrete Signale, ergibt es beispielsweise wenig Sinn, eine Integraltransformation zu definieren, Aperiodische Signale wiederum können nicht in einer Fourier-Reihe entwickelt werden, da die inhärent der Periodizität widerspricht. Das heißt, dass für jede „Art“ von Signal und dessen Eigenschaften, die passende Transformation existiert. Weiterhin ändert sich auch immer die *Interpretation* dieser Zerlegung in harmonische Komponenten.

Darüber hinaus werden wir auch den umgekehrten Weg gehen. Es ist auch möglich, Signale aus dem Frequenzbereich in den jeweils richtigen Definitionsbereich zu transformieren. In diesem Fall spricht von von *Fouriersynthese*, da wir ein Signal aus dessen Information über harmonische Anteile erzeugen/synthetisieren. Wir liefern nun also die Definitionen und Zusammenhänge der Fourier-Transformation, die wir in Abschnitt 2 ohne Erläuterungen ausgenutzt haben.

4.1 Fourier-Transformation kontinuierlicher Signale

4.1.1 Fourier-Transformation kontinuierlicher periodischer Signale

Wir haben bereits in Abschnitt 2.2 mit (2.2.1) gesehen, dass man durch Linearkombination der komplexen Sinus-Funktionen

$$\{\exp(j2\pi kF_0 t), \quad \text{für } k \in \mathbb{Z}\}$$

eine $1/F_0 = T_0$ -periodische Funktion $x : \mathbb{R} \rightarrow \mathbb{C}$ durch

$$x(t) = \sum_{k \in \mathbb{Z}} c_k \exp(j2\pi kF_0 t)$$

erhält. Dies ist demnach ein Fall von *Fourier-Synthese*, da wir aus den Gewichten c_k in der Linearkombination eine Funktion x erhalten. Wir wollen nun den umgekehrten Weg gehen, auf welchem wir für eine gegebene Funktion x die Koeffizienten c_k bestimmen können. Wir starten dazu mit

$$x(t) = \sum_{k \in \mathbb{Z}} c_k \exp(j2\pi kF_0 t) \tag{4.1.1}$$

und multiplizieren beide Seiten mit $\exp(-j2\pi \ell F_0 t)$ und integrieren über eine Periode $[0, T_0]$. Dann erhalten wir

$$\int_0^{T_0} x(t) \exp(-j2\pi \ell F_0 t) dt = \int_0^{T_0} \left(\sum_{k \in \mathbb{Z}} c_k \exp(j2\pi kF_0 t) \right) \exp(-j2\pi \ell F_0 t) dt$$

und nach Vertauschung von Summation und Integration, dass

$$\sum_{k \in \mathbb{Z}} c_k \int_0^{T_0} \exp(j2\pi(k - \ell)F_0 t) dt = \sum_{k \in \mathbb{Z}} c_k \left[\frac{\exp(-j2\pi(k - \ell)F_0 t)}{j2\pi F_0(k - \ell)} \right]_0^{T_0}.$$

Da die Funktion $\exp(-j2\pi(k - \ell)F_0t)$ im Fall $k \neq \ell$ Periode T_0 besitzt, sind alle Summanden in der rechten Summation identisch 0, außer wenn $k = \ell$. Dann ergibt sich für $\exp(j2\pi(k - \ell)F_0t) = 1$, also

$$\int_0^{T_0} \exp(j2\pi(k - \ell)F_0t) dt = \int_0^{T_0} 1 dt = T_0.$$

Final erhalten wir für die Berechnung der Fourier-Koeffizienten $c : \mathbb{N} \rightarrow \mathbb{C}$ als Berechnungsvorschrift

$$c[\ell] = \frac{1}{T_0} \int_0^{T_0} x(t) \exp(-j2\pi\ell F_0 t) dt.$$

Wir haben in diesem Fall also *Fourier-Analyse* betrieben. Außerdem haben wir absichtlich die Schreibweise von c_ℓ auf $c[\cdot]$ angepasst, um zu verdeutlichen, dass man nun die $c[\cdot]$ als *komplexes diskretes Signal* auffassen können. Dieses diskrete Signal können wir nun durch die Fourier-Reihe mit dem Signal $x : \mathbb{R} \rightarrow \mathbb{C}$ *identifizieren*. Sowohl x als auch $c[\cdot]$ sind also Darstellungen desselben Sachverhaltes – im „Zeitbereich“ und im zugehörigen „Frequenzbereich“. Wir sehen auch, dass sich für kontinuierliche und periodische Signale ein *diskreter Frequenzbereich* ergibt.

Obige Herleitung verschleiert aber einen viel tiefer liegenden Zusammenhang. Definieren wir wie in Abschnitt 2.2 die Funktionen $x_k : \mathbb{R} \rightarrow \mathbb{C}$ als

$$x_k(t) = \exp(j2\pi k F_0 t),$$

dann können wir obige Rechnung auch so auffassen. Wir betrachten Multiplikation von x mit x_ℓ^* und anschließende Integration als Skalarprodukt $\langle x, x_\ell \rangle$. Außerdem haben wir aus der Fourier-Reihe gegeben, dass

$$x = \sum_{k \in \mathbb{Z}} c_k x_k$$

Wir haben also in dieser Denkweise lediglich auf beiden Seiten der Fourierreihe in (4.1.1) das Skalarprodukt mit x_ℓ gebildet. Also

$$\langle x, x_\ell \rangle = \left\langle \sum_{k \in \mathbb{Z}} c_k x_k, x_\ell \right\rangle.$$

Das Skalarprodukt ist linear, also können wir auch

$$\langle x, x_\ell \rangle = \sum_{k \in \mathbb{Z}} c_k \langle x_k, x_\ell \rangle$$

schreiben. Wir müssen also nur $\langle x_k, x_\ell \rangle$ bestimmen. Dies haben wir aber oben schon berechnet, denn das waren die Ausdrücke

$$\langle x_k, x_\ell \rangle = \int_0^{T_0} x_k(t) x_\ell(t)^* dt = \int_0^{T_0} \exp(j2\pi k F_0 t) \exp(-j2\pi \ell F_0 t) dt = \int_0^{T_0} \exp(j2\pi(k - \ell) F_0 t) dt$$

Von oben wissen wir, dass

$$\langle x_k, x_\ell \rangle = \left[\frac{\exp(-j2\pi(k - \ell) F_0 t)}{j2\pi F_0(k - \ell)} \right]_0^{T_0} = \begin{cases} T_0 & \text{für } k = \ell \\ 0, & \text{sonst.} \end{cases}$$

Die x_k stehen also paarweise *senkrecht/orthogonal* zu einander und es gilt $\langle x, x_\ell \rangle = T_0$. Sie bilden ein sogenannten *Orthogonalsystem*. Hätten wir $\hat{x}_k = x_k / \sqrt{T_0}$ definiert, würde sogar gelten $\langle \hat{x}_k, \hat{x}_\ell \rangle = 1$ und die \hat{x}_k bilden eine *Orthonormalsystem*.

Wenn wir nun noch einmal obige Gleichung für $\langle x, x_\ell \rangle$ betrachten, finden wir

$$\langle x, x_\ell \rangle = \sum_{k \in \mathbb{Z}} c_k \langle x_k, x_\ell \rangle = T_0 c_\ell,$$

weil alle Summanden durch die Orthogonalität verschwinden und nur im Falle von $k = \ell$ eben T_0 übrig bleibt. Der Fourier-Koeffizient c_ℓ ergibt sich also als Skalarprodukt von x mit dem zugehörigen x_ℓ . Es ist wichtig zu sehen, dass wir nur für die Berechnung von $\langle x_k, x_\ell \rangle$ die spezielle Form der x_k eingesetzt haben und sonst nur allgemein mit den Eigenschaften von Skalarprodukten gearbeitet haben. Das heißt, dass man ganz allgemein Signale „transformieren“ beziehungsweise *analysieren* kann, indem man sie als Summe von paarweise orthogonalen Signalen ausdrückt. Die Fourier-Reihe ist nur ein Spezialfall von diesem allgemeinen Konzept!

In Codeschnipsel 15 wird eine Rechteckfunktion $\text{rect}_{[0,T]}$ in ihre Fourier-Reihe entwickelt. In diesem Fall müssen wir natürlich die Reihenentwicklung abbrechen, da kein K_{\max} existiert, sodass $c[k] = 0$ für $k > K_{\max}$. Wir könnten die Reihe zwar analytisch ausrechnen, da wir jedes x_k nur auf $[0, T]$ integrieren, also dem Bereich, auf dem die von uns definierte $\text{rect}_{[0,T]}$ -Funktion Werte ungleich 0 annimmt. Der Einfachheit halber nutzen wir `scipy.integrate.quad`, was in der Lage ist numerische Integration relativ präzise durchzuführen.

Es lohnt sich mit dem Wert von K_{\max} zu experimentieren. Man sieht hierbei, dass größere Werte von K_{\max} an den Unstetigkeitsstellen 0 und T und in deren Nähe nicht zu einer besseren Übereinstimmung der Fourierreihe mit x führen. Die Fourier-Reihe muss also nicht immer gegen x konvergieren. Im Falle von $\text{rect}_{[0,T]}$ ergibt sich das Problem genau aus dem Verhalten an 0 und T – also Unstetigkeit, was ein generelles Problem bei der Entwicklung von Signalen in Fourier-Reihen darstellt.

Im oberen Plot von Codeschnipsel 15 sieht man auch, dass der Realteil der Koeffizienten $\Re c[\cdot]$ ein gerades diskretes Signal ist, also $c[k] = c[-k]$. Der Imaginärteil $\Im c[\cdot]$ hingegen ist ein ungerade Signal, also $c[k] = -c[-k]$ ⁸. Dies liegt daran, dass das Signal lediglich reelle Werte annimmt, weshalb die Symmetrieeigenschaften der $c[\cdot]$ allgemein für reelle Signale gelten.

In Abbildung 6 sind noch mehr Eigenschaften der Fourier-Reihe deutlich gemacht. Generell stellen wir fest, dass die beiden Signale jeweils gut durch eine endliche Fourier-Reihe approximiert werden können, da sie keine Unstetigkeiten aufweisen. Im oberen Plot sieht man außerdem, dass Achsen-Symmetrie des Signals x dazu führt, dass die Imaginärteile von $c[\cdot]$ verschwinden. Wie man in Abbildung 6 unten sieht, ist es bei Anti-Symmetrie des Signals x der Realteil von $c[\cdot]$, der verschwindet. Dies liegt daran, dass der Realteil der x_k eine gerade Funktion ist und der Imaginärteil respektive eine ungerade Funktion. Da wir in Codeschnipsel 7 schon gesehen haben, dass ungerade und gerade Anteile eines Signals im Sinne von Skalarprodukten orthogonal sind, gilt dies auch für die resultierenden Fourier-Koeffizienten. In Abbildung 6 unten sieht man auch, dass man für gewisse Signale die Fourier-Reihe direkt angeben kann. Im Falle des Beispiels gilt nämlich

$$x(t) = \sin(10 \cdot 2\pi t) = \frac{1}{2j} (\exp(10 \cdot 2\pi t) - \exp(-10 \cdot 2\pi t)) = \frac{1}{2j} x_{10} - \frac{1}{2j} x_{-10}.$$

⁸siehe Codeschnipsel 7

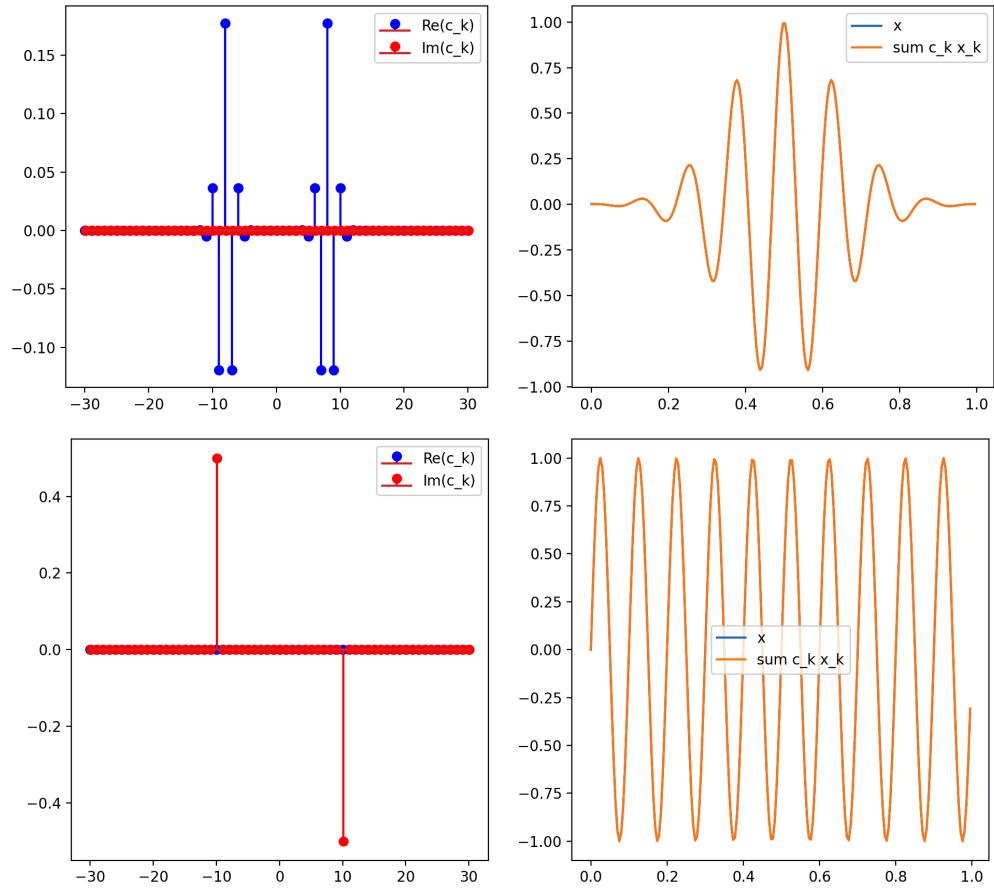


Abbildung 6: Mehr Versionen von Codeschnipsel 15; Oben: $x(t) = \exp(-25(t - 0.5)^2) \cos(16\pi t)$; Unten: $x(t) = \sin(20\pi t)$;

Das heißt, dass wir x direkt in seine Fourier-Reihe entwickelt haben, da wir es als Linearkombination der x_k dargestellt haben. Das heißt es sind nur x_{10} und x_{-10} notwendig, um x darzustellen und beide Koeffizienten haben ausschließlich imaginäre Anteile.

Ähnlich wie bei der z -Transformation kann also an der Fourier-Reihe Eigenschaften des Signals x direkt ablesen, oder umgekehrt von Eigenschaften des Signals x auf Eigenschaften der Fourier-Koeffizienten $c[\cdot]$ schließen. Außerdem werden wir für die noch folgenden Versionen der **FT** sehr analoge Zusammenhänge finden.

4.1.2 Leistungsdichte-Spektrum periodischer Signale

Das Leistungsdichtespektrum eines T_0 -periodischen Signals $x : \mathbb{R} \rightarrow \mathbb{C}$ ist gegeben durch

$$P(x) = \frac{1}{T_0} \int_0^{T_0} |x(t)|^2 dt = \frac{1}{T_0} \int_0^{T_0} x(t)x(t)^* dt = \frac{1}{T_0} \langle x, x \rangle. \quad (4.1.2)$$

Wir wollen nun $P(x)$ in Abhängigkeit der Fourier-Koeffizienten $c[\cdot]$ berechnen. Wir entwickeln also

$$x(t) = \sum_{k \in \mathbb{Z}} c_k \exp(j2\pi k F_0 t)$$

und setzen dies in P ein, um

$$\frac{1}{T_0} \int_0^{T_0} x(t)x(t)^* dt = \frac{1}{T_0} \int_0^{T_0} x(t) \sum_{k \in \mathbb{Z}} c_k^* \exp(-j2\pi k F_0 t) dt = \sum_{k \in \mathbb{Z}} c_k^* \frac{1}{T_0} \int_0^{T_0} x(t) \exp(-j2\pi k F_0 t) dt = \sum_{k \in \mathbb{Z}} c_k^* c_k \quad (4.1.3)$$

als den *Satz von Parseval*⁹ zu erhalten. Die physikalische Interpretation ist, dass $|c[k]|$ der Leistung des Signals bei der Frequenz kF_0 entspricht. Jeder Index k hat also eine physikalische Größe, die mit ihm assoziiert ist.

Da nur die Frequenzen kF_0 für $k \in \mathbb{Z}$ auftreten, also Frequenzen wie $0.1F_0$ nicht vorhanden sind, sprechen wir von einem *diskreten Spektrum*. Es ergibt sich also direkt folgender wichtiger Zusammenhang: Periodische Signale im Zeitbereich besitzen ein *diskretes Spektrum*. Andersherum ergibt sich auch: Signale mit diskretem Spektrum sind periodisch. Beide Argumente ergeben sich aus der Fourier-Reihe. Damit haben wir das duale Ergebnis zu (2.5.5) gefunden. Dort führte Diskretisierung im Zeitbereich, also Sampling/Abtastung, zu einer Periodifizierung im Frequenzbereich.

In Codeschnipsel 16 zeigen wir eine Modifikation von Codeschnipsel 15, in welcher wir $T_0 = 2$ setzen, also $F_0 = 1/2$ erhalten. Auf der Frequenzachse sehen wir, dass demnach diese für $K_{\max} = 30$ also von -15 bis $+15$ reicht.

⁹https://de.wikipedia.org/wiki/Satz_von_Parseval

```

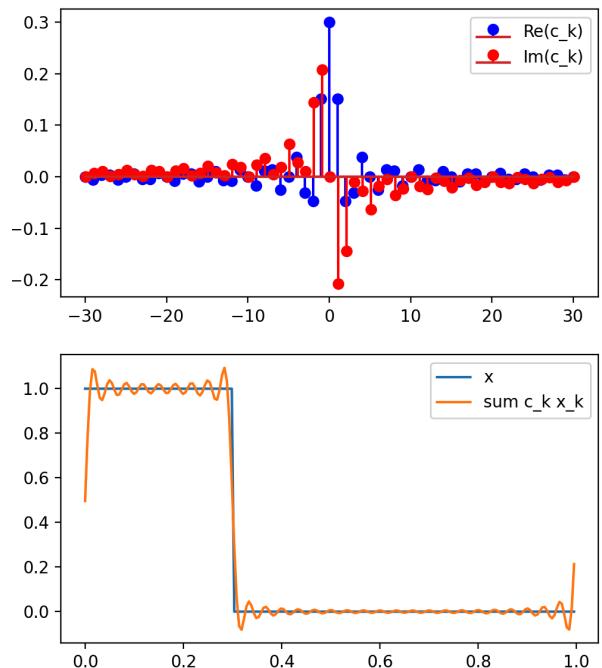
def rect(t: float) -> float:
    if (t < 0.0) or (t > 0.8):
        return 0
    else:
        return 1

def x_k(t: float, k: int) -> complex:
    return np.exp(1j * 2 * np.pi * F_0 * k * t)

def synth(
    t: float, c_k: list[complex], allK: list[int]
) -> complex:
    return np.sum(
        cckk * x_k(t, kk)
        for kk, cckk in zip(allK, c_k)
    )

allK = np.linspace(
    -Kmax, +Kmax, 2 * Kmax + 1, dtype=int
)
c_k = np.array(
    [
        quad(
            lambda t: x_k(t, kk).conj()
            * rect(t),
            0,
            T_0,
            complex_func=True,
        )[0]
        for kk in allK
    ]
)

```



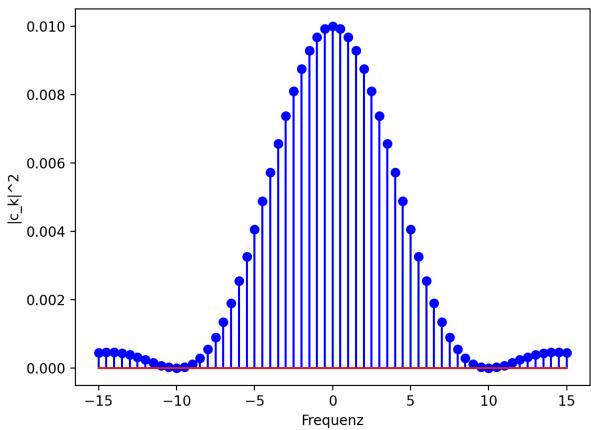
Codeschnipsel 15: Berechnung und Darstellung von (4.1.1), siehe [dsv/code/fourier_series.py](#)

```

T_0 = 2
F_0 = 1.0 / T_0
Kmax = 30

def x(t: float) -> float:
    if (t < 0.0) or (t > 0.1):
        return 0
    else:
        return 1

```



Codeschnipsel 16: Modifiziertes Codeschnipsel 15 und Plot von P im Frequenzbereich., siehe [dsv/code/period_psd.py](#)

4.1.3 Fourier-Transformation von kontinuierlichen aperiodischen Signalen

Um die Einschränkung auf periodische Signale zu vermeiden, nutzen wir die **Fourier Transform**, wie wir sie bereits in (2.5.1) für ein Signal $x : \mathbb{R} \rightarrow \mathbb{C}$ durch

$$X(F) = \int_{-\infty}^{+\infty} x(t) \exp(-j2\pi F t) dt \quad (4.1.4)$$

definiert haben. Im Unterschied zu (4.1.1) integrieren wir nun über ganz \mathbb{R} , da sich das Signal nicht mehr periodisch wiederholt. Aus diesem Grund ergibt sich auch ein *kontinuierliches* Spektrum, da wir uns nicht mehr auf eine abzählbare Menge von diskreten Frequenzen zurückziehen können. Deshalb ergibt sich auch für die Synthese des Signals, dass wir auch in diesem Fall integrieren anstatt summieren müssen, es gilt also

$$x(t) = \int_{-\infty}^{+\infty} X(F) \exp(j2\pi F t) dF. \quad (4.1.5)$$

Da sich für die meisten Signale, die wir betrachten werden, Integration und Summation „ähnlich“ verhalten, finden wir auch die obigen Eigenschaften bezüglich Symmetrien, etc., von (4.1.1) wieder.

Es gibt aber eine Verbindung zur Fourier-Reihe, die wir im Folgenden kurz erläutern wollen. Nehmen wir an, es existiert ein $T > 0$, sodass $|x(t)| = 0$ für alle t mit $|t| > T$. Dann können wir das Signal x periodifizieren mit Periode T , indem wir

$$x_p(t) = \sum_{k \in \mathbb{Z}} x(t - 2kT)$$

setzen. Dies haben wir bereits in ähnlicher Form in (2.5.5) gesehen. Dort hat es sich aber aus Berechnungen ergeben und hier setzen wir diesen Zusammenhang explizit. Dann können wir x_p in seine Fourier-Reihe

$$x_p(t) = \sum_{k \in \mathbb{Z}} c_k \exp(j2\pi kt/T) \quad \text{mit} \quad T c_k = \int_{-T/2}^{+T/2} x_p(t) \exp(-j2\pi kt/T) dt = \int_{-\infty}^{+\infty} x(t) \exp(-j2\pi kt/T) dt$$

entwickeln. Aus der Definition in (4.1.4) und der letzten Gleichung sehen wir, dass sich die Koeffizienten der Fourier-Reihe finden lassen durch

$$c_k = \frac{1}{T} X\left(\frac{k}{T}\right), \quad (4.1.6)$$

diese sich also auch aus der Fourier-Transformation ablesen lassen. Das heißt, dass wir auch

$$x_p(t) = \sum_{k \in \mathbb{Z}} \frac{1}{T} X\left(\frac{k}{T}\right) \exp(j2\pi kt/T) = \sum_{k \in \mathbb{Z}} X(k\Delta F) \exp(j2\pi kt\Delta F) \Delta F$$

schreiben können. Dabei haben wir im letzten Schritt $1/T = \Delta F$ gesetzt. Dies können wir intuitiv (aber nicht rigoros!) so interpretieren, dass im Falle von $T \rightarrow \infty$ gilt, dass $\Delta F \rightarrow 0$. Je länger der Bereich der Funktion x , auf welchem gilt $x \neq 0$, desto kleiner ΔF . Der nicht-periodische Fall $T = \infty$ ergibt sich also als Grenzfall, bei welchem obige Summation zu einer Integration wird und ΔF zu einem dF . Man kann die **Fourier Transform** also als Grenzfall der Fourier-Reihe für $T \rightarrow \infty$ betrachten.

4.1.4 Leistungsdichte-Spektrum aperiodischer Signale

Analog zum Satz von Parseval für periodische Signale in (4.1.3) können wir auch hier wieder definieren und folgern, dass

$$E(x) = \int_{-\infty}^{+\infty} |x(t)|^2 dt = \int_{-\infty}^{+\infty} |X(F)|^2 dF \quad (4.1.7)$$

auch wieder ein Parseval Theorem für die Fourier Transform ergibt, was aber in diesem Fall als Plancherel Theorem¹⁰ genannt wird.

Andererseits kann man X auch in Betrag und Phase zerlegen, da es im Allgemeinen eine komplexe Größe ist, also

$$X(F) = |X(F)| \exp(j\angle(X(F))).$$

Man nennt dann $|X(F)|^2$ das *Leistungsdichte-Spektrum* von x .

In Codeschnipsel 17 zeigen wir das Vorgehen zur Fourier-Analyse mittels numerischer Integration. Es ist hier anzumerken, dass wir „nur“ die Funktion `rect` definieren müssen und der Rest, also die Funktionen `kernel`, `analyse` und `synthese` unabhängig hiervon sind. Wir haben hier ein Codeschnipsel, welches sich Methoden der Funktionalen Programmierung bedient. Die Funktionen `texttanalyse` und `synthese` haben als Eingabewert die entweder die Funktion, oder die Fourier Transform einer Funktion. Außerdem liefern sie als Ausgabewert wieder *Funktionen*, die wir einfach „aufrufen“ können.

¹⁰https://en.wikipedia.org/wiki/Plancherel_theorem

```

def rect(t: float) -> float:
    if np.abs(t) < 0.75:
        return 1
    else:
        return 0

def kernel(t: float, F: float) -> complex:
    return np.exp(1j * 2 * np.pi * F * t)

def analyse(x, Trange: list[float]):
    def x_ft(F: float):
        return quad(
            lambda t: kernel(-t, F) * x(t),
            np.min(Trange),
            np.max(Trange),
            complex_func=True,
        )[0].real

    return x_ft

def synthese(X, Frange: list[float]):
    def x(t: float):
        return quad(
            lambda F: kernel(t, F) * X(F),
            np.min(Frange),
            np.max(Frange),
            complex_func=True,
        )[0].real

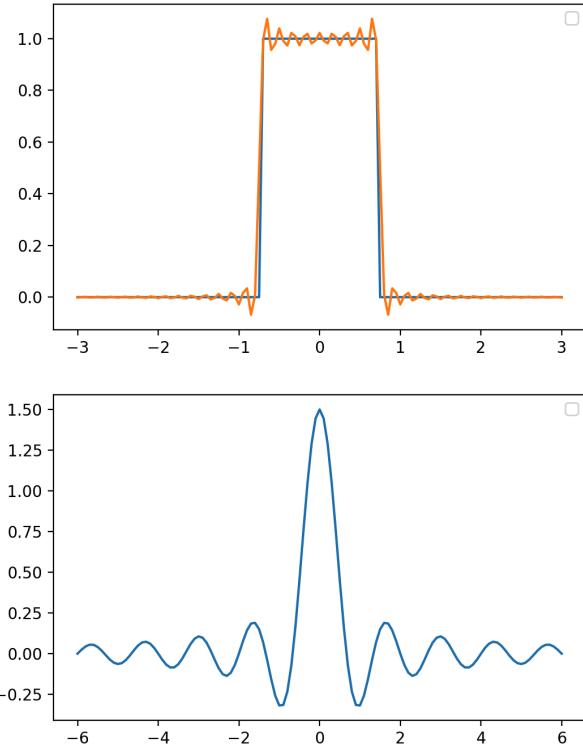
    return x

Trange = np.linspace(-3, +3, 121, endpoint=True)
Frange = np.linspace(-6, +6, 121, endpoint=True)

rect_ft = analyse(rect, Trange)
rect_synth = synthese(rect_ft, Frange)

```

Codeschnipsel 17: Berechnung und Darstellung von (4.1.4), siehe [dsv/code/fourier_trafo.py](#)



4.2 Fourier-Transformation diskreter Signale

Wir nähern uns langsam der Fourier-Analyse von diskreten Signalen. Schließlich wollen wir etwaige spektrale Analysen und dergleichen im Digitalen durchführen, um die Vorteile von digitalen Rechenwerken dabei nutzen zu können. Die vorher eingeführten Transformationen sind zwar hilfreich für theoretische Argumentation, wie beispielsweise beim Sampling-Theorem 2.1. Deshalb wenden wir uns nun der spektralen Analyse von diskreten Signalen zu. Wir werden aber im Verlauf auch wieder ähnliche Zusammenhänge wie in (4.1.6) finden.

4.2.1 Fourier-Transformation diskreter periodischer Signale

Wir beginnen mit diskreten Signalen, die gleichzeitig periodisch sind, also ein N existiert, sodass

$$x[n] = x[n + kN] \quad \text{für alle } n, k \in \mathbb{Z}$$

gilt. Aus vorherigen Diskussionen in Abschnitt 2 wissen wir einerseits, dass diskrete Signale ein periodisches Spektrum auf $(0, 1)$ besitzen. Andererseits wissen wir aus Abschnitt 4.1.1, dass periodische Signale ein *diskretes* Spektrum besitzen. Wir finden also nun intuitiv, dass das Spektrum von diskreten periodischen Signalen *ebenfalls* diskret und periodisch ist. Denken wir zurück an Abschnitt 2.3 so haben wir bereits alles Notwendige betrachtet. Ein N -periodisches diskretes Signal ergibt sich aus der Linearkombination der diskreten Signale $x_k[\cdot] : \mathbb{Z} \rightarrow \mathbb{C}$ definiert durch

$$x_k[n] = \exp\left(j2\pi \frac{k}{N}n\right) \quad \text{mit } k = 0, \dots, N-1.$$

Das heißt, für $x[\cdot]$ setzen wir mit

$$x[\cdot] = \sum_{k=0}^{N-1} c[k]x_k[\cdot]$$

an. Damit sind bereits beide Eigenschaften des Spektrums „eingepreist“. Denn man erkennt in den $x_k[\cdot]$ die vorher erwähnte *zweifache* Periodizität wieder, weil sowohl $x_{k+N}[\cdot] = x_k[\cdot]$ für alle k als auch $x_k[n+N] = x_k[n]$ für alle n gilt.

Es ist nun unser Ziel für gegebene Werte $x[n]$ von $x[\cdot]$ die Werte des *ebenfalls periodischen und diskreten* Signals $c[\cdot]$ zu bestimmen. Dies verläuft ganz analog zu Abschnitt 4.1.1. Wir definieren das Skalarprodukt $\langle \cdot, \cdot \rangle$ für N -periodische und diskrete Signale via

$$\langle x_1[\cdot], x_2[\cdot] \rangle = \sum_{n=0}^{N-1} x_1[n]x_2[n]^*.$$

Das heißt, dass wir periodisches Signal $x[\cdot]$ mit dem *endlich-dimensionalen* Vektor $\mathbf{x} \in \mathbb{C}^N$ identifizieren, wir setzen also die Einträge des Vektors als $x_i = x[i - 1]$. Dann können wir auch für das entsprechende Skalarprodukt der Vektoren $\mathbf{x}_{1,2} \in \mathbb{C}^N$

$$\langle \mathbf{x}_1, \mathbf{x}_2 \rangle = (\mathbf{x}_2^*)^T \cdot \mathbf{x}_1 = \mathbf{x}_2^H \cdot \mathbf{x}_1$$

schreiben. Analog identifizieren wir die periodische und diskrete Sequenz $c[\cdot]$ mit dem Vektor $\mathbf{c} \in \mathbb{C}^N$.

Wie in Abschnitt 4.1.1 müssen wir nur

$$\langle \mathbf{x}_k, \mathbf{x}_\ell \rangle = \sum_{i=0}^{N-1} x_k[i]x_\ell[i]^* = \sum_{i=0}^{N-1} \exp\left(j2\pi \frac{i(k-\ell)}{N}\right) = \begin{cases} N & \text{falls } k = \ell \\ \frac{1-\exp(j2\pi \frac{(k-\ell)}{N})}{1-\exp(j2\pi \frac{(k-\ell)}{N})} & \text{sonst} \end{cases} = \begin{cases} N & \text{falls } k = \ell \\ 0 & \text{sonst} \end{cases}$$

berechnen. Hierbei nutzen wir, dass $\exp(j2\pi k) = 1$ für alle $k \in \mathbb{Z}$. Wie in Abschnitt 4.1.1 finden wir, dass also gilt $\langle \mathbf{x}_k, \mathbf{x}_\ell \rangle = 0$, falls $k \neq \ell$ und $\langle \mathbf{x}_k, \mathbf{x}_k \rangle = N$. Damit ergibt sich bei Anwendung auf das eigentliche Signal \mathbf{x} für den Vektor \mathbf{c} , dass

$$\langle \mathbf{x}, \mathbf{x}_\ell \rangle = \left\langle \sum_{k=1}^N c_k \mathbf{x}_k, \mathbf{x}_\ell \right\rangle = \sum_{k=1}^N c_k \langle \mathbf{x}_k, \mathbf{x}_\ell \rangle = N c_\ell \Rightarrow \mathbf{c}_\ell = \frac{1}{N} \langle \mathbf{x}, \mathbf{x}_\ell \rangle.$$

Zusammenfassend, kann man also sagen, dass sich folgende Analyse- und Synthesegleichungen ergeben:

$$x[n] = \sum_{k=0}^{N-1} c[k] \exp(j2\pi kn/N), \quad \text{und} \quad c[k] = \frac{1}{N} \sum_{n=0}^{N-1} x[n] \exp(-j2\pi kn/N). \quad (4.2.1)$$

In Vektorschreibweise lässt sich der erste Teil durch

$$\mathbf{x} = \sum_{k=0}^{N-1} \langle \mathbf{x}, \mathbf{x}_k \rangle \mathbf{x}_k \quad (4.2.2)$$

ausdrücken.

Weiterhin, können wir eine Matrix $\mathbf{F}_N \in \mathbb{C}^{N \times N}$ definieren, deren k -te Spalte den Vektor \mathbf{x}_k beinhaltet. Wir definieren also

$$\mathbf{F}_N = [\mathbf{x}_1, \dots, \mathbf{x}_k, \dots, \mathbf{x}_N]$$

Dann können wir noch einen Schritt weitergehen und sehen, dass

$$\mathbf{c} = \frac{1}{N} \mathbf{F}_N^H \mathbf{x}, \quad \text{und} \quad \mathbf{x} = \mathbf{F}_N \mathbf{c}$$

gilt. Das heißt, dass sich Fourier-Analyse und Fourier-Synthetisierung von diskreten und periodischen Signalen durch eine Matrix-Vektor-Multiplikation durchführen lässt! Das sind erst einmal gute Nachrichten, denn damit wissen wir, dass digitale Rechenwerke sehr gut darin sind, diese Transformation durchzuführen. Die hier vorgestellte Transformation nennen wir **Discrete Fourier Transform (DFT)** und oft definiert man ein diskretes Signal $X[\cdot]$ durch

$$X[k] = \frac{1}{N} \sum_{n=0}^{N-1} x[n] \exp(-j2\pi kn/N) = c[k] = \mathbf{c}_k \quad (4.2.3)$$

und nennt dieses dann die **DFT** von $x[\cdot]$. Es ist anzumerken, dass der Vorfaktor N^{-1} in manchen Lehrbüchern oder Veröffentlichungen auch bei der Synthese anstatt der Analyse auftaucht, oder aber *beide* Gleichungen erhalten einen Faktor $N^{-1/2}$.

In Codeschnipsel 18 zeigen wir ein einfaches Beispiel für $x[\cdot] = u[\cdot] - u[\cdot - k]$. Außerdem zeigen wir auch die beiden Berechnungsmethoden der $c[\cdot]$ – einerseits mit der Definition und andererseits über ein Matrix-Vektor-Produkt.

4.2.2 Leistungsdichte-Spektrum diskreter periodischer Signale

Da periodische Signale keine endliche Energie besitzen, betrachten wir die durchschnittliche Leistung über eine Periode. Wir betrachten also für ein N -periodisches Signal $x[\cdot]$ das Funktional

$$P(x[\cdot]) = \frac{1}{N} \sum_{i=0}^{N-1} |x[n]|^2. \quad (4.2.4)$$

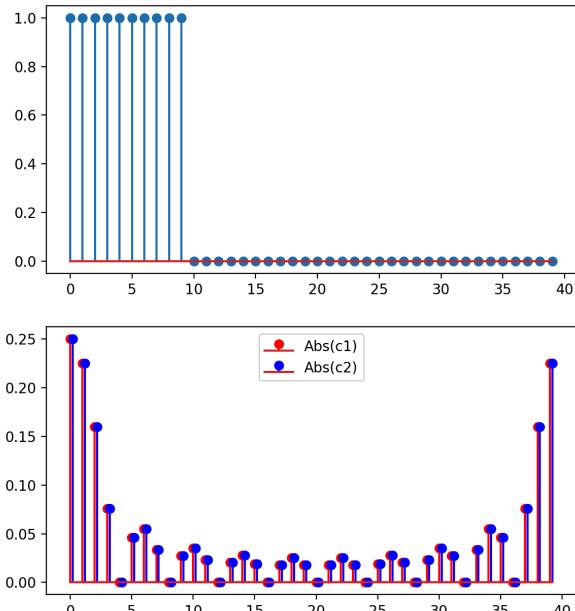
```

def x_k(k: int, N: int) -> np.ndarray[complex]:
    return np.exp(
        1j * 2 * np.pi * k * np.arange(N) / N
    )

N = 40
x = np.zeros(N)
k = N // 4
x[:k] += np.ones(k)

c1 = np.array([
    (x * x_k(kk, N).conj()).sum() / N
    for kk in range(N)
])
F = np.array([x_k(kk, N) for kk in range(N)]).T
c2 = F.conj().T.dot(x) / N

```



Codeschnipsel 18: Berechnung und Darstellung von (4.2.1), siehe [dsv/code/dft_1.py](#)

Dann können wir ganz analog zu vorher wieder herleiten, dass für

$$x[\cdot] = \sum_{i=0}^{N-1} c[i] x_i[\cdot]$$

und dessen Leistung gilt, dass

$$P(x[\cdot]) = \sum_{i=0}^{N-1} |c[i]|^2.$$

Also auch hier finden wir wieder, dass sich die durchschnittliche Leistung einer Periode im Zeitbereich auf die Leistung einer Periode im Frequenzbereich überträgt. Auch hier können wir die Folge $|c[\cdot]|^2$ als Leistungsdichte-Spektrum interpretieren.

4.2.3 Fourier-Transformation diskreter aperiodischer Signale

Wiederum analog zu Abschnitt 4.1 benötigen wir noch eine Transformation für diskrete Signale, die keine Periodizität aufweisen. Hierzu definieren wir die zugehörige Fourier-Transformation, die wir **DTFT** nennen, durch

$$X(f) = \sum_{n \in \mathbb{Z}} x[n] \exp(-j2\pi f n), \quad (4.2.5)$$

genau wie in Abschnitt 2 bei der Herleitung von (2.1). Im Unterschied zur „analogen“ Fourier-Transformation (4.1.4) sehen wir, dass X nur für Frequenzen $f \in (0, 1]$ definiert ist, da das diskrete Signal $x_f[n] = \exp(j2\pi f n)$ periodisch in f ist. Es gilt also $x_{f+k}[\cdot] = x_f[\cdot]$ für alle $k \in \mathbb{Z}$. Dies „passt“ auch zur Natur von diskreten Signalen, da deren Frequenzbereich *immer* periodisch sein muss.

Wir können nun für die inverse Transformation (4.2.5) mit (4.1.1) vergleichen. Bis auf das Vorzeichen in der Funktion $\exp()$ gleicht (4.2.5) einer Fourier-Reihe der periodischen Funktion X . In der Tat, können wir

die Folge $x[\cdot]$, also das ursprüngliche Signal, als Fourier-Koeffizienten der DTFT X durch

$$x[n] = \int_{-1/2}^{+1/2} X(f) \exp(j2\pi fn) df \quad (4.2.6)$$

wiederfinden. Diese Synthese-Operation nennen wir dann **Inverse Discrete Time Fourier Transform (iDTFT)**. Es ist wiederum anzumerken, dass die DTFT „nur“ ein theoretisches Werkzeug ist, da sich im Allgemeinen die unendliche Summe in (4.2.5) praktisch nicht realisieren lässt, genauso wenig wie deren Resultat, eine kontinuierliche Funktion.

In Codeschnipsel 19 zeigen wir dennoch, wie man die DTFT einer aperiodischen Folge approximieren kann. Wir studieren hierzu das Signal

$$x[n] = \begin{cases} \frac{\omega}{\pi} & \text{für } n = 0, \\ \frac{\omega}{\pi} \frac{\sin(\omega n)}{\omega n} & \text{sonst} \end{cases} .$$

Dessen analytisch bestimmte DTFT X ist

$$X(f) = \text{rect}(f/(2\pi\omega)),$$

welche wir durch eine endliche Summe über die von uns verfügbaren Werte von $x[\cdot]$ bestimmen.

Nun treten hierbei mehrere Effekte zutage.

- Die Approximation der DTFT mittels des „abgeschnittenen“ Signals x stimmt noch nicht sehr gut mit der analytischen Lösung überein. Verändert man den Wert der Variable N im Skript, tritt dieser Effekt stärker oder schwächer zutage.
- Auch bei Erhöhung von N stellt sich immer noch keine Konvergenz von X_{approx} gegen X_{true} ein. Dies liegt daran, dass die DTFT von $x[\cdot]$ nicht konvergiert, wie wir in Codeschnipsel 15 bereits gesehen hatten. Diesen Effekt bezeichnet man allgemein als Gibbsches Phänomen¹¹.
- Augenscheinlich ist es besser die iDTFT aus X_{approx} zu berechnen, als aus X_{true} . Doch dies liegt lediglich daran, dass die numerische Integration der rect-Funktion nicht genau genug ist. Erhöht man die Anzahl der Punkte im Array F, dann stimmen im dritten Plot die drei Graphen besser überein.

¹¹https://en.wikipedia.org/wiki/Gibbs_phenomenon

```

def dtft(
    x: np.ndarray[complex],
    n: np.ndarray[int],
    F: np.ndarray[float],
) -> np.ndarray[complex]:
    return np.array(
        [
            (
                x * np.exp(-2j * np.pi * n * f)
            ).sum()
            for f in F
        ]
    )

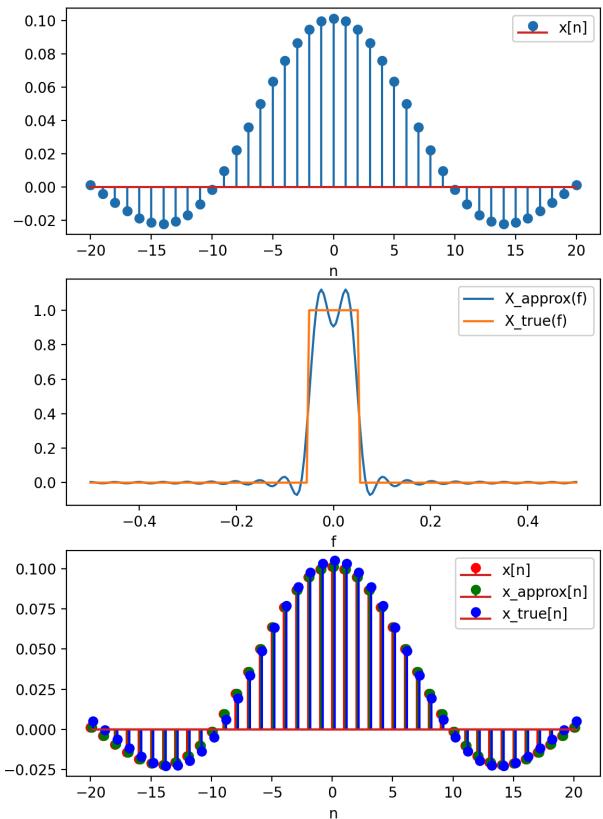
def idtft(
    X: np.ndarray[complex],
    F: np.ndarray[float],
    n: np.ndarray[int],
):
    return np.array(
        [
            (F[1] - F[0])
            * (
                X * np.exp(2j * np.pi * nn * F)
            ).sum()
            for nn in n
        ]
    )

N = 20
n = np.linspace(-N, +N, 2 * N + 1)
omega = 1 / np.pi

x = np.sin(omega * n) / (np.pi * n)
x[N] = omega / np.pi
F = np.linspace(-0.5, +0.5, 201)
X_approx = dtft(x, n, F)
X_true = np.zeros_like(F)
X_true[np.abs(F) <= omega / (2 * np.pi)] += 1

x_approx = idtft(X_approx, F, n)
x_true = idtft(X_true, F, n)

```



Codeschnipsel 19: Berechnung und Darstellung von (4.2.5), siehe [dsv/code/dtft.py](#)

4.2.4 Leistungsdichte-Spektrum diskreter aperiodischer Signale

Die Energie eines diskreten Signals haben wir in (3.1.1) durch

$$\mathcal{E}(x[\cdot]) = \sum_{n \in \mathbb{Z}} |x[n]|^2$$

definiert. Wiederum können wir uns überlegen, dass dann für die DTFT X gilt, dass

$$\mathcal{E}(x[\cdot]) = \int_{-1/2}^{+1/2} |X(f)|^2 df.$$

Schlussendlich bezeichnet man dann

$$S_{xx}(f) = |X(f)|^2$$

als das Leistungsdichte-Spektrum des Signals $x[\cdot]$. In manchen Anwendungen ist es auch wieder sinnvoll Betrag und Phase von $X(f)$ zu analysieren.

4.3 Eigenschaften der Fourier-Transformationen

Nachdem wir nun die notwendigen Definitionen gesammelt haben, wollen wir uns einige wichtige Eigenschaften der definierten Transformationen ansehen und eventuell auch für „praktische“ Dinge verwenden.

4.3.1 Beziehung der Fourier-Transformationen und der z -Transformation

Wie wir in Abschnitt 3.4 gesehen haben, ist die z -Transformation für ein diskretes Signal $x[\cdot]$ definiert durch

$$X_z(z) = \sum_{n \in \mathbb{Z}} x[n]z^{-n} \quad \text{mit ROC } r_2 < |z| < r_1.$$

Wenn wir nun z in Polarform $z = r \exp(j2\pi f)$ ausdrücken und annehmen, dass $r_2 < r < r_1$, dann sehen wir, dass die z -Transformation von $x[\cdot]$ nichts anderes ist als die DTFT von $x[n]r^{-n}$ ist. Mit anderen Worten ist die z -Transformation an einer Stelle $z = r \exp(j2\pi f)$ die DTFT X eines Signals gewichtet mit der Folge $w[n] = r^{-n}$ an der Stelle f . Ist nun $1 \in \text{ROC}$ (und damit der gesamte Einheitskreis der komplexen Ebene), dann gilt

$$X_z(\exp(j2\pi f)) = X(f) \tag{4.3.1}$$

Wir haben uns zwar bis jetzt wenig Gedanken über die Existenz der Fourier-Transformation gemacht, doch hier finden wir einen Hinweis. Die DTFT existiert, beziehungsweise konvergiert, falls $1 \in \text{ROC}$. Da die „Form“ der ROC immer kreisförmig ist, ist die gleichbedeutend mit der Tatsache, dass sich der Einheitskreis der komplexen Ebene in der ROC befindet. Außerdem kann man auch eine Fourier-Analyse von LTI-Systemen betreiben. Dann ist die BIBO-Stabilität von einem solchen System äquivalent zur Tatsache, dass der Einheitskreis in der ROC liegt. Wie wir gerade gesehen haben, folgt dann auch, dass BIBO-Stabilität ebenfalls durch die Existenz der DTFT charakterisiert wird.

In Codeschnipsel 20 zeigen wir diesen Zusammenhang für das gleiche Signal, wie in Codeschnipsel 18, interpretieren es hier aber einmal als aperiodisches Signal bei der Berechnung der DTFT und als periodisches Signal für die Berechnung der DFT. Wir wissen aus den Eigenschaften der z -Transformation, dass das

Signal $x[n] = u[n] - u[n - k]$ im z -Bereich einen k -fachen Pol bei $z = 0$ hat und $k - 1$ Nullstellen auf dem Einheitskreis.

Wir können in dem Plot von Codeschnipsel 20 direkt sehen, dass die DTFT auf dem Einheitskreis in der Tat mit der z -Transformation übereinstimmt. Außerdem sehen wir gut den Pol bei $z = 0$ und die 9 Nullstellen auf dem Einheitskreis. In den Plots der DTFT und der DFT sehen wir auch, dass sich die DFT durch Auswertung der DTFT ergibt. Wir hatten vorher in Gleichung (4.1.6) schon gesehen, dass sich die Fourier-Koeffizienten $c[k]$ eines periodischen kontinuierlichen Signals aus der Auswertung der Fourier-Transformation an den richtigen Stellen ergeben. denselben Zusammenhang finden wir hier wieder, denn zwischen der DFT und der DTFT besteht derselbe Zusammenhang.

```

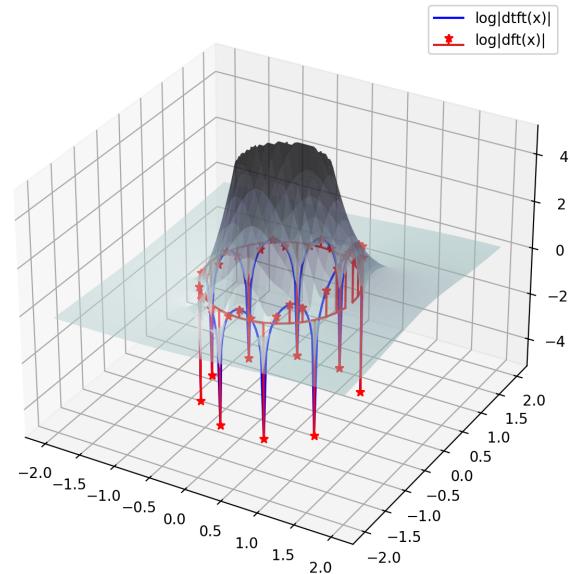
N = 40
x = np.zeros(N)
k = N // 4
x[:k] += np.ones(k)

res = 128
real = np.linspace(-2, +2, res, endpoint=True)
imag = np.linspace(-2, +2, res, endpoint=True)
zValues = np.concatenate(
    [
        gg.flatten()[None, :]
        for gg in np.meshgrid(
            real,
            imag,
        )
    ]
).T

zTrafo = np.array(
    [
        (
            x
            * (zz[0] + 1j * zz[1])
            ** (-np.arange(N))
        ).sum()
        for zz in zValues
    ]
).reshape(res, res)

dtftValues = np.exp(
    1j
    * 2
    * np.pi
    * np.linspace(0, 1, 201, endpoint=True)
)
dtft = np.array(
    [
        (x * zz ** (-np.arange(N))).sum()
        for zz in dtftValues
    ]
)
dft = np.fft.fft(x)

```



Codeschnipsel 20: Berechnung und Darstellung von (4.3.1), siehe [dsv/code/dtft_z.py](#)

4.3.2 Weitere Eigenschaften der DFT

Wir wollen analysieren, welche Eigenschaften die Beziehung zwischen einem diskreten N -periodischen Signal $x[\cdot]$ und dessen **DFT** $X[\cdot]$ besitzt.

Für die meisten Eigenschaften macht es Sinn die **DFT** eines Signals der Länge N und die entsprechende **Inverse Discrete Fourier Transform (iDFT)** zu definieren durch

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{k \cdot n} \quad \text{und} \quad x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] W_N^{-k \cdot n}, \quad (4.3.2)$$

wobei wir hier

$$W_N = \exp(-j2\pi/N) \quad (4.3.3)$$

definiert haben.

DFT ist periodisch: Wir wissen bereits aus der Definition und den Eigenschaften von komplexen diskreten Harmonischen, dass

$$X[k + \ell \cdot N] = X[k]$$

für alle $\ell \in \mathbb{Z}$ gilt.

Übereinstimmung mit DTFT: Aus Codeschnipsel 20 wissen wir, dass

$$X(k/N) = X[k]$$

gilt – wir die Werte der **DFT** also aus Auswertung der **DTFT** von $x[\cdot]$ erhalten. Es ist wichtig zu erwähnen, dass die Frequenzen an denen die **DTFT** X ausgewertet wird harmonische „Verwandte“ sind, da wir X an den Stellen $f = k/N$ auswerten.

Abgetastete Signale Ist $x[\cdot]$ aus Abtastung eines analogen Signals $x : \mathbb{R} \rightarrow \mathbb{C}$ entstanden, so erhält die Einheit von f bei der **DTFT** eine Bedeutung, da dann „Perioden pro Sample“ eine physikalische Interpretation zulässt. Tasten wir x mit Sampling-Rate F_s ab, so ist der Abstand zwischen zwei Samples in $x[\cdot]$ genau $1/F_s$. Demzufolge erhält die **DTFT** $X(f)$ die Interpretation, dass wir das periodifizierte Spektrum von x an den Stellen $f \cdot F_s$ betrachten. Das heißt beispielsweise bei einer Sample-Rate von 44 kHz entspricht der Bereich $f \in [-1/2, +1/2]$ dem *physikalischen* Frequenzbereich von $F \in [0 \text{ Hz}, 44 \text{ kHz}]$. Es ist jedoch zu beachten, dass wir bei der Betrachtung der **DTFT** *immer* nur das periodifizierte Spektrum erhalten. Das heißt also, nur wenn wir Aliasing-frei abgetastet haben, also F_s groß genug gewählt haben, können wir aufschlussreiche Aussagen über das Spektrum von x basierend auf der **DTFT** von $x[\cdot]$ treffen. Wie bereits erwähnt, ergibt dann die **DFT** des Signals $x[\cdot]$ eben die periodifizierten spektralen Informationen von x an den diskreten Frequenzen kF_s/N .

Reelle Signale Intuitiv sind reelle Signale „einfacher“ als komplexe Signale. Deshalb mag es nicht verwunderlich sein, dass die **DFT** von reellen Signalen Struktur hat, in dem Sinne, dass wir einige Koeffizienten aus anderen schnell berechnen können. Ist ein Signal reell, so gilt $x[n] = x[n]^*$ für alle n . Für W_N gilt

$$W_N^\ell = (W_N^{-\ell})^* \quad \text{und} \quad W_N^{N-\ell} = (W_N^\ell)^*$$

also gilt

$$X[N-k] = \sum_{n=0}^{N-1} x[n] W_N^{(N-k)n} = \sum_{n=0}^{N-1} x[n]^* (W_N^{kn})^* = \sum_{n=0}^{N-1} x[n]^* W_N^{-kn} = \left(\sum_{n=0}^{N-1} x[n] W_N^{kn} \right)^* = X[k]^* = X[-k].$$

Das heißt, dass die **DFT** von reellen Signalen „konjugiert symmetrisch“ um $k = 0$ sind. Dies hat zur Folge, dass man nur die Koeffizienten von $k = 0$ bis $k = \lceil N/2 \rceil$ berechnen und speicher muss, was den Rechenaufwand halbiert. Außerdem wird hierdurch auch die **iDFT** weniger komplex. In Python ist es in solchen Fällen deshalb ratsam auf `np.fft.rfft` und `np.fft.irfft` zurückzugreifen. Es sei noch angemerkt, dass sich noch viele weitere Redundanzen und Symmetrien ausnutzen lassen. Siehe hierzu [1, Kap. 7.2.1].

DC-Komponente und Nyquist-Sequenz Der Wert $X[0]$ wird oft als DC-Komponente, also Gleichanteil, bezeichnet, weil $W_N^0 = 1$, also ergibt sich speziell für $X[0]$, dass

$$X[0] = \sum_{n=0}^{N-1} x[n],$$

was eben dem um Faktor N skalierten Mittelwert des Signals $x[\cdot]$ entspricht. Eine andere Interpretation ergibt sich aus der Betrachtung der **DFT** als Skalarprodukt, wobei wir die „Ähnlichkeit“ von $x[\cdot]$ mit der konstanten Sequenz $x_0[\cdot] = 1$ berechnet haben. Ist ein Signal reell, so ergibt sich aus der Symmetrie des Spektrums, dass der DC-Anteil immer ebenfalls reell sein muss. Glück gehabt!

Der andere Extremfall tritt auf, wenn $k = N/2$. Hierbei wird vorausgesetzt, dass N eine gerade Zahl ist. Dann nennt man $x_{N/2}$ die Nyquist-Sequenz, denn wie wir in **Abgetastete Signale** gesehen haben, entspricht $k = N/2$ der physikalischen Frequenz $F = (N/2)F_s/N = F_s/2$, also genau der *maximalen* Frequenz, die ein Signal beinhalten darf, sodass es noch mit der gewählten Sampling-Rate ohne Aliasing beobachtbar ist.

Man sieht in Codeschnipsel 21 gut, dass die beiden Sequenzen wirklich „Gegensätze“ darstellen, da $x_0[\cdot]$ die Anteile mit „minimaler“ Variation/Frequenz aufsammelt, während die Werte von $x_{N/2}[\cdot]$ genau so liegen, dass ein analoges Signal (und dessen Aliase), siehe Codeschnipsel 4, genau Frequenz $F_s/2$ (und Amplitude 1) besitzen muss, um $x_{N/2}[\cdot]$ als Abtastwerte zu erhalten.

Zyklische Faltung im Zeitbereich Kommen wir nun zur wahrscheinlich wichtigsten Eigenschaft der **DFT** und dem Grund, warum digitale Signalverarbeitung überhaupt ermöglicht wurde. Wir haben bereits gesehen, dass **LTI**-Systeme durch eine Faltung realisiert werden können. Gegeben zwei N -periodische Signale $x_{1,2}[\cdot]$ und betrachten wir deren *zyklische Faltung* \circledast definiert durch

$$x_3[m] = \sum_{n=0}^{N-1} x_1[n] \cdot x_2[(m-n) \mod N] = (x_1 \circledast x_2)[m], \quad (4.3.4)$$

dann kann man zeigen, dass

$$X_3[k] = X_1[k] \cdot X_2[k]$$

gilt. Das heißt, dass die zyklische Faltung von zwei periodischen Signalen wieder ein periodisches Signal ergibt, dessen **DFT** das Produkt der **DFTs** der gefalteten Signale ist. Wenn wir an **LTI**-Systeme denken und uns erinnern, dass viele Filter als **LTI**-Systeme aufgefasst werden können, so ist klar, warum diese Eigenschaft wichtig ist. Die digitalen Filter können also im Frequenzbereich sehr einfach angewendet werden, weil dort nur eine einfache punktweise Multiplikation notwendig ist. Doch dies allein ist nicht ausreichend, denn

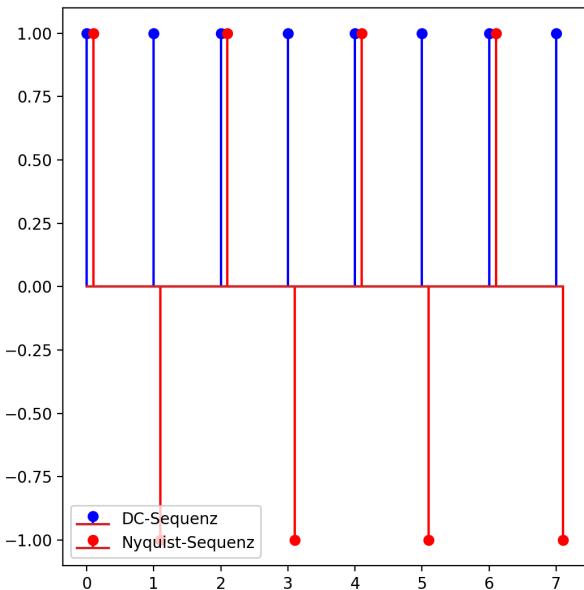
```

N = 8

X = np.fft.fft(np.eye(N))
x_DC = X[:, 0]
x_Nyquist = X[:, N // 2]

plt.stem(
    np.arange(N),
    x_DC,
    linefmt="b",
    label="DC-Sequenz",
)
plt.stem(
    np.arange(N) + 0.1,
    x_Nyquist,
    linefmt="r",
    label="Nyquist-Sequenz",
)
plt.legend()
plt.show()

```



Codeschnipsel 21: Darstellung der DC-Sequenz und der Nyquist-Sequenz für $N = 8$, siehe [dsv/code/nyquist_seq.py](#)

wir benötigen noch einen effizienten weg, um zwischen Zeit- und Freqeunzbereich wechseln zu können. Dies wird uns durch die **Fast Fourier Transform (FFT)** ermöglicht werden. In Codeschnipsel 18 hatten wir bereits erfahren, dass man die **DFT** als Matrix-Vektor-Produkt formulieren kann, wozu etwa **Floating Point Operations (FLOPs)** in der Größenordnung N^2 notwendig sind. Die **FFT** erlaubt es aber eine **DFT** mit **FLOPs** in der Größenordnung von nur $N \log N$ durchzuführen.

Dann beläuft sich eine effiziente Implementierung eines **FIR** Filters mit Impulsantwort $h[\cdot]$ auf folgende Schritte:

1. Transformiere $x[\cdot]$ und $h[\cdot]$ in den Frequenzbereich
2. Punktweise Multiplikation von $Y[\cdot] = X[\cdot] \cdot H[\cdot]$ (NICHT die z -Transformation H_z)
3. Rücktransformation von $Y[\cdot]$ zu $y[\cdot]$.

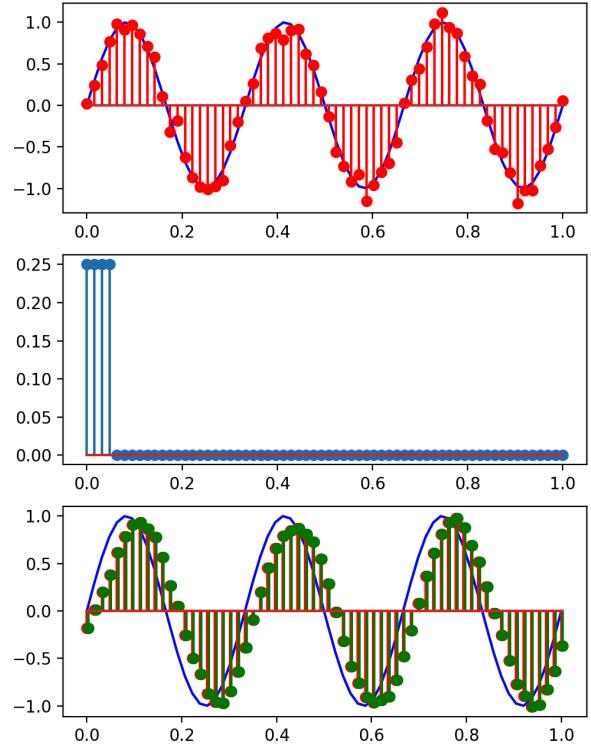
Dieser Prozess wurde in Codeschnipsel 22 nachgestellt und wird mit der zyklischen Faltung im Zeitbereich verglichen. Als Beispiel nehmen wir hier eine abgetastete Sinusfunktion, deren Werte mit Rauschen belegt sind. Um das Signal ein wenig vom Rauschen zu befreien, implementieren wir ein gleitendes Mittel, siehe Codeschnipsel 10, dessen Mittelungsdauer wir mit einem „cut-off“ einstellen können. Wie man sieht, ist das Ergebnis der Faltung im Zeitbreich mit dem im Frequenzbereich identisch. Außerdem stellt sich durch die Mittelung eine Glättung der Werte ein, die mehr dem Verlauf des ursprünglichen Signals ähnelt. Jedoch sieht man auch, dass sich die Phase des Resultates sich von der des Eingangs unterscheidet. Diese Phase wird vom Filter h aufgeprägt und ist somit eine Konsequenz der gewählten Signalverarbeitung. Es gibt Möglichkeiten diesen Einfluss zu reduzieren, die aber über unser aktuelles Interesse hinausgehen.

Es sei hier angemerkt, dass dies der einfachste Fall ist, den man sich für die sogenannte schnelle Faltung vorstellen kann. Es ist beispielsweise nicht direkt klar, was zu tun ist, wenn $x[\cdot]$ und $h[\cdot]$ nicht dieselbe Länge haben, was dazu führt, dass die resultierenden DFTs ebensfalls von verschiedener Länge sind. Darüber hinaus ist unsere Implementierung eine „offline“-Version in dem Sinne, dass die zu verarbeitenden Daten bereits im Speicher liegen. Doch, wenn man an Echtzeitanwendungen, beispielsweise im Audiobereich, denkt so wird es notwendig sein, diese Operation auf Daten anzuwenden, die nicht vollständig im Zugriff liegen. Außerdem will man natürlich nicht immer eine zyklische Faltung, sondern eine „normale“ Faltung implementieren.

Alle diese Probleme können aber mit einigen Tricks so umformuliert werden, dass man am Ende wieder eine zyklische Faltung rechnen kann. Oft müssen hierzu das Signal oder die Impulsantwort zielführend manipuliert werden, oder das Ergebnis nach Faltung noch prozessiert werden.

```
N = 64
F = 3
T = np.linspace(0, 1, N)
cutOff = 0.05
sin = np.sin(2 * np.pi * 3 * T)
x = sin + 0.1 * np.random.randn(N)
h = np.zeros_like(T) + 1 * (T < cutOff)
h /= np.sum(h)

# zirkulare faltung im zeitbereich
# Gleichung (4.3.4)
y1 = np.convolve(x, np.tile(h, 2))[N : 2 * N]
y2 = np.fft.ifft(
    np.fft.fft(x) * np.fft.fft(h))
.y.real
```



Codeschnipsel 22: Zyklische/periodische Faltung im Zeitbereich und im Frequenzbereich., siehe [dsv/code/dft_conv.py](#)

Zyklische Faltung im Frequenzbereich Als Dual zu [Zyklische Faltung im Zeitbereich](#) gilt auch, dass die zyklische Faltung im Frequenzbereich eine Multiplikation der jeweiligen Signale im Zeitbereich zur Folge hat. Es gilt also, für ein Signal $x_3[\cdot]$ mit

$$x_3[n] = x_1[n] \cdot x_2[n],$$

dass

$$X_3[k] = \frac{1}{N} (X_1 \circledast X_2)[k]. \quad (4.3.5)$$

5 Anwendung der Disreten Fourier-Transformation

5.1 Spektralanalyse mit der Short Time Fourier Transformation

Einleitung Als erstes wollen wir uns mit der Fourier-Analyse von Signalen beschäftigen. Hierbei ist das Ziel das Verhalten eines Signals im Frequenzbereich zu charakterisieren. Wir wollen jedoch in unserer Analyse davon ausgehen, dass wir ein Signal $x[\cdot]$ vorliegen haben, dessen spektrale Eigenschaften *zeitvariant* sind. Hiermit ist *nicht* gemeint, dass wir davon ausgehen, dass das Signal zeitvariant ist, denn das ist es natürlich. Es geht darum, dass der Anteil der verschiedenen Frequenzen in einem Signal sich mit der Zeit ändert. Natürlich ist hier eine der am einfachsten zugänglichen Anwendungen die Analyse von Audiosignalen.

Beginnen wir mit einer intuitiven Betrachtung. Zunächst wollen wir einen möglichst großen Bereich im Frequenzbereich abdecken – zumindest den für die jeweilige Anwendung relevanten. Da sich beispielsweise im Audiobereich oft an der menschlichen Hörcharakteristik orientiert wird, die jenseits von 20 kHz nichts mehr leistet, sind Frequenzen jenseits dieser Grenze nicht mehr relevant. Das heißt, dass sinnvollerweise ein Audiosignal mit einer Sample-Rate von 40 kHz bereits absolut ausreichend aufgenommen wurde. Für andere Anwendungen lassen sich oft ähnliche Argumente finden, da die meisten physikalischen Systeme eine Tief- oder Bandpasscharakteristik „versteckt“ haben, und man somit nur sehr selten mit Signalen mit immensen Bandbreiten konfrontiert ist. Zusammenfassend muss also die Sample-Rate so angepasst werden, dass sie entsprechend unserer Anforderungen ausreicht.

Darüber hinaus ist uns aber auch gut daran getan, die Frequenzen, die in dem Signal vorkommen, sehr gut unterscheiden zu können. Wir wollen also möglichst *genau* wissen, welche einzelnen Frequenzen im Signal vorhanden sind. Das heißt, dass wir für eine tatsächlich im Signal vorhandene Frequenz F aus unserer Analyse eine Frequenz erhalten wollen, die im Intervall $[F - \Delta_1, F + \Delta_1]$ für möglichst kleines $\Delta_1 > 0$ liegt. In diesem Sinne wollen wir also möglichst große *Auflösung*. Wenn wir daran denken, dass $\Delta_1 = F_s / N$ gilt, so scheint es zu helfen, dass wir ein Signal möglichst lange „beobachten“ müssen, also N sehr groß wählen müssen.

Doch es gibt noch einen weiteren Auflösungsbegriff, der vom ersten strikt zu unterscheiden ist. Wir wollen außerdem eine maximale Trennschärfe zwischen zwei unterschiedlichen Frequenzen, die gleichzeitig im Signal vorhanden sind, erreichen. Wir wollen also in unserer Analyse zwei Frequenzen F_1 und F_2 mit $|F_1 - F_2| = \Delta_2$ immernoch unterscheiden können. Wiederum ist hier das minimale $\Delta_2 > 0$ von Interesse, für das diese Unterscheidbarkeit noch gilt, da dieses ein weiteres Auflösungslimit unseres Systems ist. Ähnlich wie bei Δ_1 ist auch hier der Beobachtungszeitraum N , welchen wir als Grundlage für unsere Analyse verwenden, von Bedeutung. Größer Zeiträume erlauben eine bessere Auflösung *zwischen* Frequenzen.

Außerdem ist es noch oft interessant, dass man einen hohen Dynamikbereich abdecken kann. Dies meint, dass man Frequanzanteile mit Frequenzen F_1 und F_2 und Amplituden A_1 und A_2 noch als zwei wahrnimmt, solange $\log(A_1 / A_2) > \Delta_3$. Je größer das maximale Δ_3 ist, desto besser, da prominente Frequanzanteile nicht weitere weniger stark ausgeprägte Anteile maskieren.

Schlussendlich haben wir es auch mit zeitvarianten Frequanzanteilen zu tun. Das heißt, dass sich zum Zeitpunkt T_1 die Zusammensetzung der Frequenzen von denen zum Zeitpunkt T_2 unterscheidet. Für $|T_1 - T_2| > \Delta_4$ wollen wir also erkennen, dass $x(T_1)$ andere spektrale Charakteristik hat als $x(T_2)$ und die für möglichst kleine Δ_4 . Diese zeitliche Evolution des Spektrums ist natürlich von immenser Bedeutung, weil man möglichst genau wissen möchte, *wann* sich Frequenzen im Signal verändern. Ideal wäre es also, wenn wir eine Art „instantane“ Frequanzanalyse durchführen könnten, die trennscharf in einem infinitesimalen

Intervall die jeweiligen Frequenzanteile aus dem Signal extrahiert. Doch Schwingungen sind auf inhärent *ausgedehnte* Effekte. Ein kleines Δ_4 steht also in direktem Widerspruch zu den Anforderungen an unsere Analyse für, gute (lies: kleine) Δ_1 und Δ_2 , da wir dort gesehen haben, dass wir das Signal *lange* beobachten müssen. Für diesen Beobachtungszeitraum nehmen wir aber das Signal als „konstant“ im Frequenzbereich – also genau kontraproduktiv für ein kleines Δ_4 . Wir wollen uns hier lediglich mit der diskreten Version der **Short Time Fourier Transformation (STFT)** befassen, weil wir diese uns als Analysetool von abgetasteten Daten vorstellen, die mit Rate F_s aufgenommen wurden. Uns liegen also Samples des Signals $x : \mathbb{R} \rightarrow \mathbb{C}$ vor, die wir zum diskreten Signal $x[\cdot]$ abgetastet haben.

Definition Dann definieren wir eine Fensterbreite $W \in \mathbb{N}$ und eine Verzögerung $S \in \mathbb{N}$ und bilden die W -dimensionalen Vektoren x_s als

$$x_n^\ell = [x[\ell \cdot S + n]]_{n=0}^{W-1} \in \mathbb{C}^W$$

und mit diesen eine zweidimensionale Datenstruktur

$$\mathbf{X} = [x^1, \dots, x^L] \in \mathbb{C}^{W \times L},$$

welche aus L vielen Teilen von $x[\cdot]$ der Länge W besteht. Dann berechnen wir spaltenweise die **DFT** auf \mathbf{X} und erhalten \mathbf{X}_{stft} .

Interpretation Im Licht der Überlegungen aus **Einleitung** können wir nun diese Definition interpretieren und noch etwas aufbohren, um bessere Eigenschaften beziehungsweise mehr Information aus der **STFT** zu erhalten. Dabei machen wir uns zunutze, dass eine diskrete **STFT** „nur“ eine blockweise angewandte **DFT** ist. Das heißt, dass deren Eigenschaften sich ausnutzen lassen, um die **STFT** zu beeinflussen.

Zunächst stellen wir fest, dass wir in einem Block von X_{stft} eine Frequenzauflösung von $\Delta_1 = F_s/NW$, da die Periodendauer eines Blockes im Zeitbereich die Länge WN/F_s hat. Jetzt könnte man auf die Idee kommen, dass man gerne diese Auflösung verbessern will. Eine Möglichkeit besteht darin die Fensterbreite künstlich zu verlängern. Eine beliebte Herangehensweise ist an X eine bestimmte Anzahl von Nullen anzuhängen.

Dieses sogenannte *zero-padding* wird in Codeschnipsel 23 gezeigt. Man sieht, dass man bei hinzufügen von $2N$ Nullen an ein Signal der Länge N eine *Interpolation* der **DFT** erreicht. Diese Technik kann dann „blockweise“ bei der **STFT** angewandt werden, um einen besseren Überblick des Spektrums zu bekommen. Beispielsweise erlaubt dies auch eine genauere Bestimmung des Maximums der **DFT**, was der im Signal „prominentesten“ Frequenz entspricht. Da wir aber nur Nullen im Zeitbereich anhängen, entsteht keine neue Information, sondern wir erhalten nur zusätzliche Zwischenwerte. Für die **STFT** heißt das, dass wir lediglich ein schöner aufgelöstes Bild erhalten, wenn wir zero-padding anwenden.

```

N = 25
F_s = 25

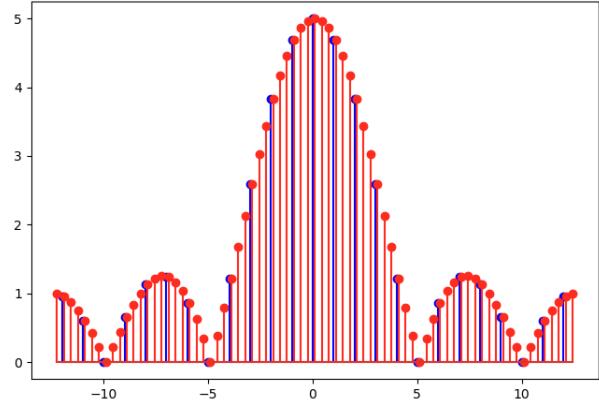
x = np.zeros(N)
x[: N // 5] = 1

freq_x = np.fft.fftshift(
    np.fft.fftfreq(x.size) * F_s
)
X = np.fft.fftshift(np.fft.fft(x))

x_zp = np.pad(x, (0, 2 * N))

freq_x_zp = np.fft.fftshift(
    np.fft.fftfreq(x_zp.size) * F_s
)
X_zp = np.fft.fftshift(np.fft.fft(x_zp))

```



Codeschnipsel 23: Zero-padding im Zeitbereich einer rect-Funktion liefert eine Überabtastung der sinc-Funktion im Frequenzbereich, siehe [dsv/code/stft_zp.py](#)

Wir sehen auch noch einen weiteren Effekt von zero-padding, wenn wir ein harmonisches Signal mit der DFT analysieren. Die FT einer einzelnen Sinusschwingung mit Frequenz F_0 entspricht genau zwei komplexen Dirac-Stößen bei $\pm F_0$. Doch was wir im Plot von Codeschnipsel 24 sehen ist, dass wir dies im Diskreten nicht reproduzieren können. Bei Anwendung der Technik aus Codeschnipsel 23 stellt sich heraus, dass wir eher eine Faltung der Dirac-Stöße bei $\pm F_0$ mit einer sinc-Funktion sehen. Deren Auftreten lässt sich dadurch erklären, dass zero-padding eines Signals der Länge N auf Länge M im Zeitbereich der Multiplikation eines Signals mit einem diskreten rect-Signal entspricht, welches bei N vielen Werten den Wert 1 annimmt. Da dessen DFT, wie in Codeschnipsel 23 gezeigt, einem sinc entspricht, sehen wir im Frequenzbereich die genannte Faltung. Wenn wir im Zeitbereich die Länge des Signals auf $2N$ verlängern, so multiplizieren wir mit einer „breiteren“ rect-Funktion im Zeitbereich und erhalten so einen „schmäleren“ sinc im Frequenzbereich.

Für die STFT heißt das, genau das, was wir oben schon intuitiv analysiert haben. Beobachten wir das Signal in einem längeren Block, so können wir benachbarte Frequenzen besser auseinanderhalten. Dies gilt jedoch nur, wenn sich diese Zusammensetzung im Frequenzbereich nicht stark ändert, während man diese Analyse durchführt.

```

N = 25
F_s = 25

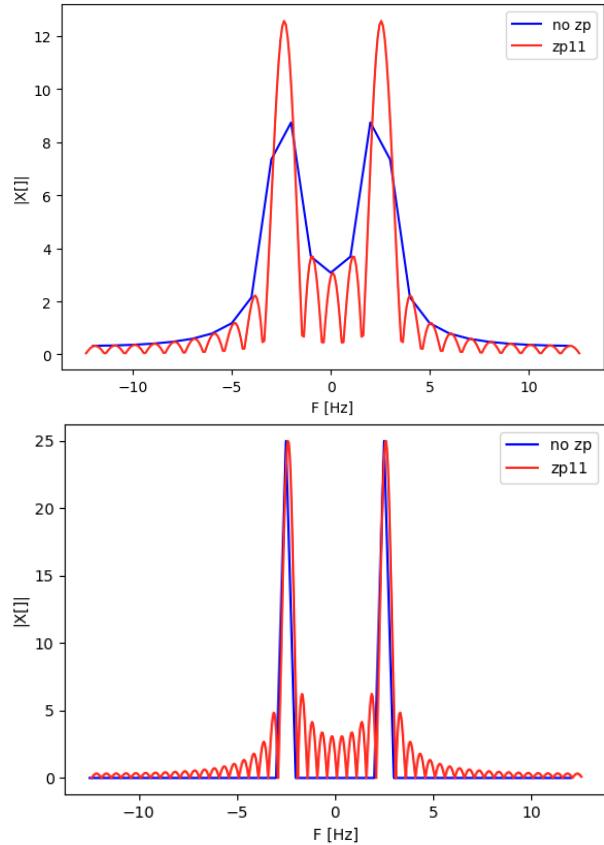
x = np.sin(2 * np.pi * 0.1 * np.arange(N))

freq_x = np.fft.fftshift(
    np.fft.fftfreq(x.size) * F_s
)
X = np.fft.fftshift(np.fft.fft(x))

x_zp = np.pad(x, (0, 10 * N))

freq_x_zp = np.fft.fftshift(
    np.fft.fftfreq(x_zp.size) * F_s
)
X_zp = np.fft.fftshift(np.fft.fft(x_zp))

```



Codeschnipsel 24: Einfluss der Fensterbreite bei der Analyse von harmonischen Schwingungen, siehe [dsv/code/stft_harm.py](#)

Man kann sich diesen Effekt aber auch zunutze machen und somit das Ergebnis der DFT beeinflussen, indem man das Signal vor der DFT noch zusätzlich fenstert, wobei man die Fensterfunktion genau so konzipiert, wie man es gerne hätte¹².

Wenn wir uns beispielsweise an die Diskussion bezüglich der möglichen Dynamik in einem Signal erinnern, so kann es sein, dass man eher an Dynamik als an Auflösung interessiert ist. In solch einem Fall, kann man beispielsweise auf ein Hann-Fenster¹³ zurückgreifen und anstatt einem Vektor \mathbf{x} transformieren wir mit der DFT den Vektor $\mathbf{w} \odot \mathbf{x}$, wobei \odot die punktweise Multiplikation bedeutet. Dies führt dazu, dass wir im Frequenzbereich eine Faltung mit der DFT von \mathbf{w} erhalten anstatt einem sinc.

In Codeschnipsel 25 sehen wir nun, dass dies erlaubt Frequenzanteile zu finden, die sich ohne explizite Fensterung in den sogenannten Nebenwellen der sinc-Funktion „verstecken“. Die Nebenwellen der sinc-Funktion fallen ab gemäß

$$|\text{sinc}(x)| \leq \frac{1}{x},$$

wobei andere Fensterfunktionen im Frequenzbereich deutlich schneller abfallen können. Wir sehen aber auch, dass sich hier wiederum ein Trade-Off aufzeigt, da die sogenannte Hauptkeule der DFT von \mathbf{w} deutlich breiter ist als die der sinc-Funktion. Das heißt, dass man durch Auswahl der Fensterfunktion zwischen Auflösung im Sinne von $\Delta_{1,2}$ und Dynamikbereich wählen kann, beziehungsweise muss. Denn, wie wir gesehen haben, impliziert eine DFT immer eine Multiplikation mit der entsprechenden rect-Funktion und man kann diese nur gegen eine andere ersetzen, aber nicht vermeiden! Für die STFT bedeutet dies, dass wir hier mit derselben Fragestellung konfrontiert sind und entsprechend der gewünschten Anwendung wählen müssen.

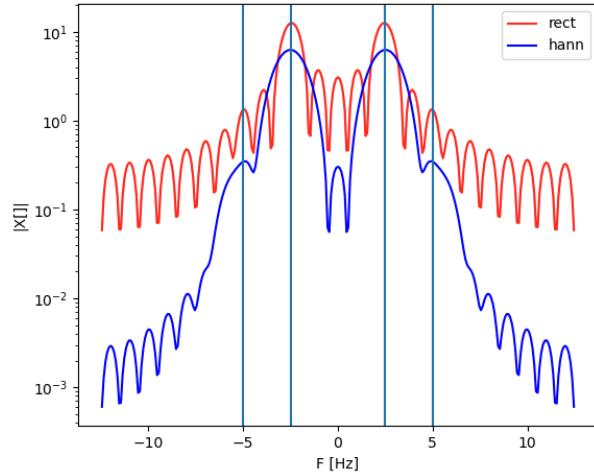
```
N = 25
F_s = 25

x = np.sin(
    2 * np.pi * 0.1 * np.arange(N)
) + 0.05 * np.sin(2 * np.pi * 0.2 * np.arange(N))

x_1 = np.pad(x, (0, 10 * N))

freq_x = np.fft.fftshift(
    np.fft.fftfreq(x_1.size) * F_s
)
x_1 = np.fft.fftshift(np.fft.fft(x_1))

x_2 = np.pad(
    x * hann(x.size, sym=False), (0, 10 * N)
)
x_2 = np.fft.fftshift(np.fft.fft(x_2))
```



Codeschnipsel 25: Geschickte Fensterung kann den Dynamikbereich erhöhen, siehe [dsv/code/stft_win.py](#).

¹²<https://docs.scipy.org/doc/scipy/reference/signal.windows.html>

¹³<https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.hann.html>

Als nächstes wollen wir mit Codeschnipsel 26 den Punkt machen, dass bei der STFT direkt Auflösung im Frequenzbereich gegen Auflösung im Zeitbereich getauscht wird. Wollen wir genau Änderungen im Frequenzbereich detektieren, brauchen wir kurze Fenster, opfern dafür aber Genauigkeit bei der Loaklisierung der Frequenz. Andererseits können wir ein Signal länger beobachten und haben sehr gute Auflösung im Frequenzbereich, was aber impliziert, dass wir nicht genau sagen können, *wann* sich Dinge im Frequenzbereich ändern.

```

def f(t):
    if t < 5:
        return np.cos(2 * np.pi * t * 10)
    elif t < 10:
        return np.cos(2 * np.pi * t * 25)
    elif t < 15:
        return np.cos(2 * np.pi * t * 50)
    else:
        return np.cos(2 * np.pi * t * 100)

F_s = 400.0 # Hz
T_max = 20.0 # s
T = np.linspace(0, T_max, int(T_max * F_s)) # s

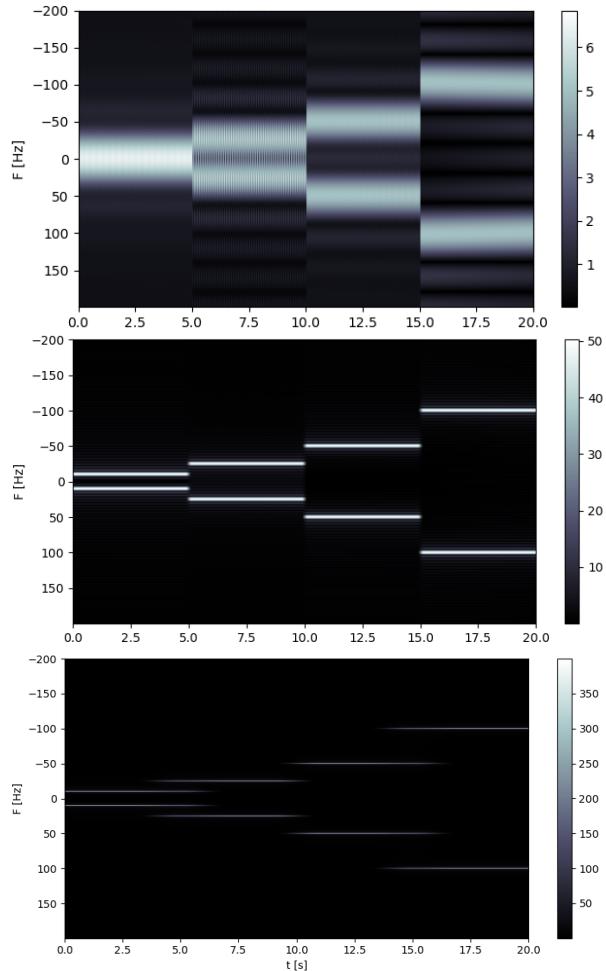
W = 2.0 # s
W_N = int(F_s * W) # samples

x = np.array([f(tt) for tt in T])
X = x.reshape((-1, W_N)).T

F = np.fft.fftshift(np.fft.fftfreq(1024)) * F_s

STFT = np.fft.fftshift(
    np.fft.fft(X, n=1024, axis=0), axes=0
)

```



Codeschnipsel 26: STFT für verschiedene Fensterlängen, siehe [dsv/code/stft_length.py](#)

Als letztes wollen wir ein Beispiel für eine Anwendung der STFT zeigen. In Codeschnipsel 27 wurde mit einer Bassgitarre eine G-Dur-Tonleiter eingespielt und in entsprechende Blöcke zerteilt, die in etwa den einzelnen Tönen entsprechen. Im Frequenzbereich wurde dann eine „peak search“ genutzt, um die lokalen Maxima der einzelnen DFT-Blöcke zu finden. Die ermittelten Frequenzen der einzelnen Töne passen sehr gut zu denen, die man online in Tabellen¹⁴ finden kann.

```

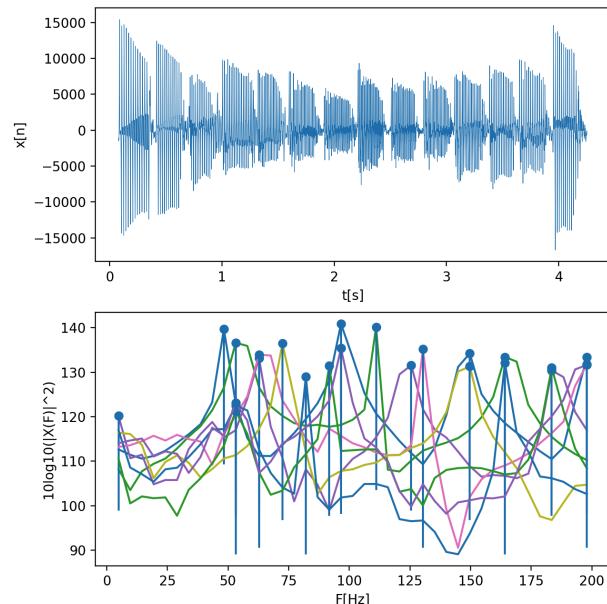
sliceRun = slice(
    int(0.08 * F_s), int(4.25 * F_s)
)
T = np.linspace(
    0, x.size / F_s, x.size, endpoint=False
)

y = x[sliceRun>[
    :int(np.floor(x[sliceRun].size / 14) * 14)
].reshape((14, -1))[:, 1000:-1000]
N = y.shape[1]

dfts = np.fft.fft(y, axis=1)
freqs = np.fft.fftfreq(N) * F_s
plotInd = np.logical_and(
    (np.abs(freqs)) < 200, freqs > 0
)

plt.figure()
plt.subplot(211)
plt.plot(T[sliceRun], x[sliceRun], linewidth=0.4)
plt.xlabel("t[s]")
plt.ylabel("x[n]")
plt.subplot(212)
for dd in range(8):
    spectrum_dB = 10 * np.log10(
        np.abs(dfts[dd, :]) ** 2
    )
    plt.plot(
        freqs[plotInd],
        spectrum_dB[plotInd],
    )
    peakSearch = scipy.signal.find_peaks(
        spectrum_dB, height=120
)

```



```

[ 48.2811  96.5622 149.6715 197.9527]
[ 53.109   111.0466 164.1559]
[ 4.8281   62.7654 125.5309 183.4683]
[ 62.7654 130.3590 197.9527]
[ 72.4217 149.6715]
[ 53.109   82.0779 164.1559]
[ 53.109   91.7341 183.4683]
[ 53.109   96.5622 197.95270418]

```

Codeschnipsel 27: Analyse einer „Bassline“ mittels STFT, die eine G-Dur-Tonleiter spielt, siehe [dsv/code/stft_1.py](#)

¹⁴<https://mixbutton.com/mixing-articles/music-note-to-frequency-chart/>

5.2 Interpolation mittels DFT

5.2.1 Theoretische Grundlagen

Gegeben sei ein periodisches, analoges Signal $x : \mathbb{R} \rightarrow \mathbb{C}$, mit Periode $T_p = 1/F_0$. Wir beobachten dieses Signal auf einem uniformen Raster von Punkten, via $x[n] = x(nT)$ und wollen eine Funktion $y(t)$ herleiten, für welche die Interpolationsbedingung

$$y(nT) = x[n] = x(nT) \quad (5.2.1)$$

erfüllt ist. Hierzu entwickeln wir das Signal x in seine Fourier-Reihe via

$$x(t) = \sum_{k=-\infty}^{+\infty} c[k] \exp(j2\pi ktF_0). \quad (5.2.2)$$

Nun tasten wir dieses Signal uniform mit Samplerate $F_s = N/T_p = 1/T$ (also passend zur Periodendauer) ab und erhalten die Folge

$$x[n] = x(nT) = \sum_{k=-\infty}^{+\infty} c[k] \exp(j2\pi knTF_0) = \sum_{k=-\infty}^{+\infty} c[k] \exp\left(j2\pi k \frac{n}{N}\right) \quad \text{für } n \in \mathbb{N} \quad (5.2.3)$$

bestehend aus den Samples von x . Mit der Periodizität von $\exp(j2\pi t)$ und der Abtastung erhalten wir außerdem noch

$$x[n] = \sum_{k=0}^{N-1} \left[\sum_{\ell=-\infty}^{+\infty} c[k-\ell N] \right] \exp\left(j2\pi k \frac{n}{N}\right) = \sum_{k=0}^{N-1} \tilde{c}[k] \exp\left(j2\pi k \frac{n}{N}\right), \quad (5.2.4)$$

wobei wir

$$\tilde{c}[k] = \sum_{\ell=-\infty}^{+\infty} c[k-\ell N] \quad (5.2.5)$$

als Abkürzung benutzt haben. Wir nehmen nun aus guten Grund noch an, dass die Funktion x auch bandbegrenzt ist. Das heißt, ihre Fourier-Transformierte $X : \mathbb{R} \rightarrow \mathbb{C}$ verschwindet außerhalb eines gewissen Bandes, also

$$X(F) = \int_{-\infty}^{+\infty} x(t) \exp(-j2\pi Ft) dt = 0 \quad \text{für } |F| > B. \quad (5.2.6)$$

Außerdem wissen wir, dass die Fourier-Transformation X und die Folge $c[k]$ verknüpft sind via

$$c[k] = \frac{1}{T_p} X(kF_0), \quad (5.2.7)$$

was impliziert, dass die Folge $c[k]$ ab einem gewissen $k > K_0$ verschwindet. Es gilt also

$$c[k] = 0 \quad \text{für } |k| > \left\lceil \frac{B}{F_0} \right\rceil. \quad (5.2.8)$$

Das heißt, dass wir nun $F_s = N/T_p$ so groß wählen müssen, dass sich in (5.2.5) kein Aliasing für $\tilde{c}[k]$ ergeben kann. Es muss also gelten

$$N > \lceil B/F_0 \rceil \quad \text{bzw. } F_s > \lceil B/(F_0 T_p) \rceil. \quad (5.2.9)$$

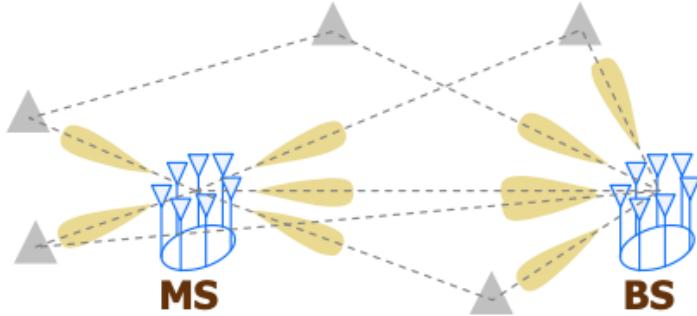


Abbildung 7: Schematische Darstellung einer **MIMO** Kanalmessung. Grafik aus [5].

Es folgt dann, dass $c[k] = X[k]$, wobei $X[k]$ die **DFT** der Folge $x[n]$ darstellt. Das heißt, dass wir die Fourier-Koeffizienten der kontinuierlichen Funktion x durch die **DFT** der Abtastwerte $x[n]$ bestimmen können. Mit (5.2.2) können wir also die Folge $x[n]$ interpolieren, indem wir

$$y(t) = \frac{1}{T_p} \sum_{k=-\frac{B}{F_0}}^{+\frac{B}{F_0}} X[k] \exp(j2\pi ktF_0) \quad (5.2.10)$$

schreiben. Dieses y erfüllt die Interpolationsbedingung (5.2.1), weil wegen der Bandbegrenzung von x und der Periodizität sogar $y(t) = x(t)$ für alle t gilt.

Man beachte hier, dass nun aus der Folge von diskreten Werten $x[n]$ eine analytische Formel in Form einer *endlichen* Summation entstanden ist. Unter der Annahme der Bandlimitierung von x ist diese Interpolation *exakt* und kann effizient implementiert werden, durch die Vorberechnung der Folge $X[k]$ durch die **FFT** [3] der Folge $x[n]$. Eine Anwendung der hier vorgestellten Methode zur Interpolation wird in Abschnitt 5.2.2 aufgezeigt.

5.2.2 Interpolation von Antennen-Richtcharakteristiken

Motivation Mit jeder Erschließung von neuen Frequenzbereichen für die Kommunikation ist es von Interesse das Ausbreitungsverhalten der Elektro-Magnetischen Wellen für verschiedene Umgebungen zu charakterisieren, beispielsweise innerstädtisch, auf der Autobahn, etc. Zwar können solche Umgebungen auch computerbasiert simuliert werden, doch für eine empirisch abgeleitete Statistik solcher sogenannter Kanalmodelle [4] sind repräsentative Messungen unerlässlich. Diese Charakteristiken werden genutzt, um in realistischen Szenarien Kanalkapazitäten, Datenraten und dergleichen zu bestimmen. Schlussendlich fließen solche Statistiken dann in neue Mobilfunkstandards ein.

Das FG EMS hat sich deshalb unter anderem auf solche Messungen und deren Auswertung, das sog. Channel Sounding [6], spezialisiert. Hierbei kommen meist breitbandige **Multiple Input Multiple Output (MIMO)** Messsysteme zum Einsatz, die den Funkkanal in Frequenz, Raum und Zeit kohärent vermessen können, wie in Abbildung 7 dargestellt. Anschließend nutzt man spezielle Signalverarbeitungstechniken [7], die einerseits unter gewissen physikalischen Annahmen das Ausbreitungsverhalten aus den gemessenen Daten ableiten können, und andererseits gleichzeitig den Einfluss des Messsystems so weit wie möglich aus den geschätzten Kanalstatistiken entfernen. Schließlich ist man an der Realität außerhalb des Messaufbaus interessiert.

Natürlich sind hierzu vor der Messung präzise Kalibriermessungen des Systems notwendig. Wir wollen uns im folgenden auf die Wirkung der benutzten Antennenarrays konzentrieren, da diese – wie wir sehen werden – eine gewisse Sonderbehandlung benötigen. Zunächst stellt man bei der Konzipierung und Benutzung des Messsystems sicher, dass es sich um ein LTI System handelt. Betrachtet man nun das Verhalten des Systems im Frequenzbereich für ein einzelnes Paar von Sende- und Empfangsantenne, dann gilt demnach zunächst

$$Y(f) = G_{\text{rx}}(f) \cdot H(f) \cdot G_{\text{tx}}(f) \cdot X(f). \quad (5.2.11)$$

Hierbei steht X für die Anregung des Systems durch ein eingegebenes Signal, $G_{\text{tx}/\text{rx}}$ für die Transferfunktion der Sender- bzw. Empfängerhardware, und H für die Transferfunktion des Funkkanals, der demnach auch als ein LTI System modelliert wird. Es stellt sich aber heraus, dass jede Antenne eine *winkelabhängige* Richtcharakteristik besitzt. Das heißt, dass die Systemantworten $G_{\text{tx}/\text{rx}}$ davon abhängig sind, in welche Richtungen sich die Wellen vom Sender tx ausbreiten und aus welchen Richtungen, sie am Empfänger rx eintreffen.

Um dies korrekt zu modellieren, muss man sich also zunächst auf einzelne sog. *Ausbreitungspfade* konzentrieren. Das heißt wir nehmen an, dass eine ebene Welle sich in die normierte Richtung Ω_{tx} ausbreitet und nach ihrem Weg durch den Funkkanal am Empfänger aus normierter Richtung Ω_{rx} eintrifft. Folglich ergibt sich für dieses Verhalten

$$Y(f, \Omega_{\text{tx}}, \Omega_{\text{rx}}) = G_{\text{rx}}(f) \cdot a_{\text{rx}}(f, \Omega_{\text{rx}}) \cdot H(f, \Omega_{\text{tx}}, \Omega_{\text{rx}}) \cdot a_{\text{tx}}(f, \Omega_{\text{tx}}) \cdot G_{\text{tx}}(f) \cdot X(f), \quad (5.2.12)$$

wobei $a_{\text{tx}/\text{rx}}$ für die richtungs- und frequenzabhängige Antwort der Sende- und Empfangsantenne stehen. In diesem Fall bezeichnet also $H(f, \Omega_{\text{tx}}, \Omega_{\text{rx}})$ die Transferfunktion eines einzelnen Pfades, der den Sender in Richtung Ω_{tx} verlässt und am Empfänger aus Richtung Ω_{rx} eintrifft. Das heißt, wir haben in diesem Fall das Verhalten der Antennen vom Rest des Systems isoliert.

Die Transferfunktion für die gesamte Messung wird dann als Summe der Transferfunktionen solcher ebenen Wellen modelliert, also via

$$Y(f) = \sum_{s=1}^S Y(f, \Omega_{\text{tx},s}, \Omega_{\text{rx},s}), \quad (5.2.13)$$

was sich dadurch rechtfertigt, dass die Transferfunktion des Kanals sich aus der Lösung einer partiellen Differentialgleichung ergibt, deren Lösungsraum lineare Struktur hat. Es zeigt sich aus (5.2.12), dass wir eine möglichst präzise Formulierung für $a_{\text{tx}/\text{rx}}$ benötigen, um die Transferfunktion des Kanals H korrekt bestimmen zu können.

Messvorgang Es ist also unsere Aufgabe für eine gegebene Antenne ein parametrisches Modell $a : [0, \pi] \times [0, 2\pi] \rightarrow \mathbb{C}$ der Form $a(\varphi, \theta) \in \mathbb{C}$, also in Betrag und Phase, herzuleiten. Aus Gründen der Einfachheit und der Physik vernachlässigen wir die Frequenzabhängigkeit der Antenne und konzentrieren uns auf ihr Verhalten für die Anregung mit einer einzelnen Frequenz. Auch die Polarisation von ebenen Wellen und das davon abhängige Verhalten einer Antenne vernachlässigen wir hier. Wir konzentrieren uns also auf die *Winkelabhängigkeit* der Antennenantwort.

Wie oben motiviert benötigen wir eine kontinuierliche Beschreibung der Antennenantwort. Doch diese ist uns wegen endlichem Speicherplatz auf Festplatten und angepeilter endlicher Messzeit nicht direkt zugänglich. Man weiß jedoch, dass es einen Zusammenhang zwischen der elektrischen Größe einer Antenne und deren winkelabhängigen Verhalten gibt [4, Kapitel 4]. Das heißt, man kann zeigen, dass die Funktion

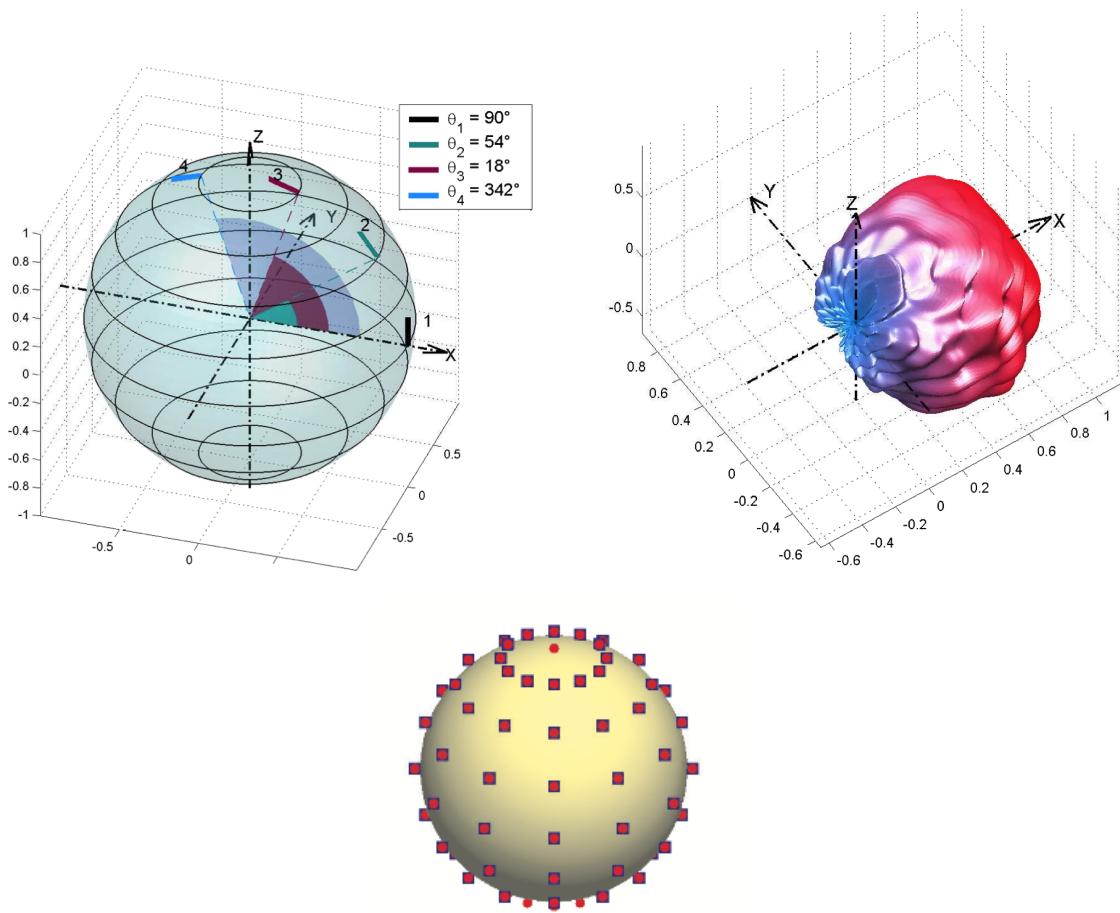


Abbildung 8: Links Oben: Darstellung der Messpositionen für eine Antenne, oder eine Antennenstruktur, welche sich im Ursprung des abgebildeten Koordinatensystems befindet. Rechts Oben: Darstellung der winkelabhängigen Amplitude eines einzelnen Patch-Elements. Unten: Messpunkte für die Abtastung der Funktion a . Grafiken aus [8, 4].

a bandbegrenzt ist, beziehungsweise sich sehr gut durch eine bandbegrenzte Funktion approximieren lässt. Weiterhin ist durch die Stetigkeit der Physik jede Antennencharakteristik periodisch. *Fourier-Reihen-Sound intensifies*

Das heißt weiterhin, dass es uns möglich ist, die Antenne an diskreten Stellen abzutasten, sodass wir mit der Annahme der Bandbegrenzung und der Aussage des Nyquist-Theorems ein Modell ableiten können, welches die Antenne vollständig charakterisiert. In diesem Sinne geht es darum die kontinuierliche Antennenantwort a zu "digitalisieren". Die "Abtastung" erfolgt demnach im Winkelbereich. Der zugehörige "Frequenzbereich" ist entsprechend der *räumliche Frequenzbereich*. Abbildung 8 zeigt einerseits schematisch den Messaufbau, das genutzte Koordinatensystem und beispielhaft die 3D-Darstellung einer Antennenantwort eines einzelnen Patch-Elements.

Die Messung selbst erfolgt in einer echofreien Messkammer, in welcher es möglich ist, die **Antenna Under Test (AUT)** beliebig relativ zu einer bereits kalibrierten Referenzantenne zu verdrehen, sodass ein

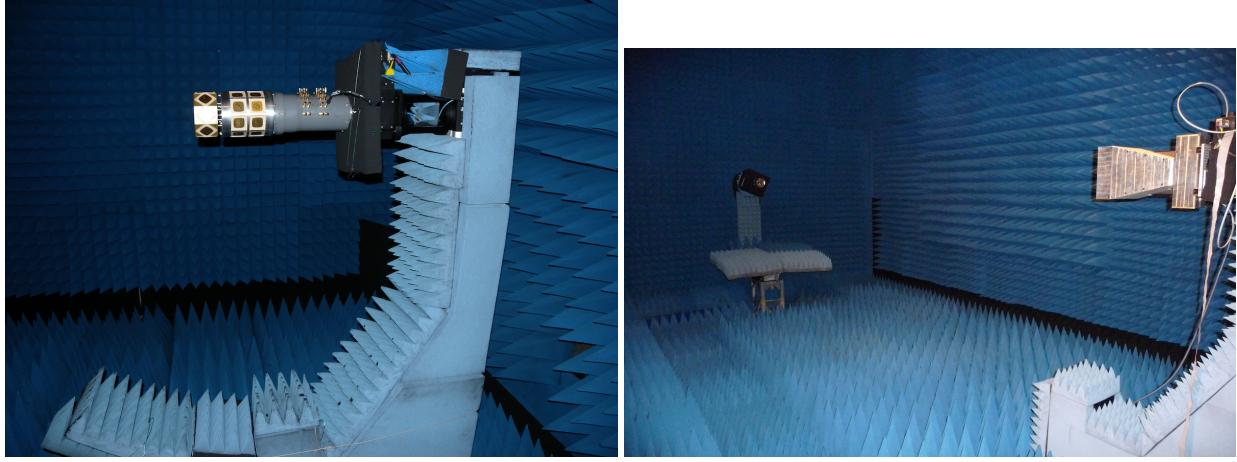


Abbildung 9: Messaufbau zur Kalibrierung eines Antennenarrays. Links: Drehteller für die Positionierung der **AUT**. Rechts: Weiterer Blickwinkel mit Referenzantenne.

Abtastraster, wie in Abbildung 8 unten gezeigt, entsteht. Pro Ausrichtung wird die **AUT** für gewöhnlich für mehrere Frequenzen, zwei orthogonale Polarisationen und alle ihre Elemente vermessen, bevor die nächste Ausrichtung angefahren wird. Wie erwähnt konzentrieren wir uns auf eine einzelne Frequenz, ein einzelnes Element und eine seiner Polarisationen. In Abbildung 9 sieht man den Messaufbau, der von unserem FG benutzt wurde, um ein Antennenarray mit 32 Elementen zu vermessen.

Für ein einzelnes Antennenelement beobachten wir also die Funktion a auf einem Gitter, das aus allen Kombinationen der Punkte

$$\varphi_0, \dots, \varphi_{N_\varphi}, \varphi_i = \frac{i\pi}{N_\varphi} \quad \text{und} \quad \vartheta_0, \dots, \vartheta_{N_\vartheta-1}, \vartheta_j = \frac{j2\pi}{N_\vartheta}$$

besteht. Wir erhalten also ein zweidimensionales Array $a[i, j] \in \mathbb{C}^{N_\varphi \times N_\vartheta - 1}$, welches wir noch durch einen Trick geeignet periodifizieren müssen, wie in Abbildung 10 links dargestellt. Diese Abtastwerte entspringen also einer zweidimensionalen bandbegrenzten, periodischen Funktion. Aufgabe ist es nun aus diesen Werten eine geeignete Interpolante herzuleiten, die sich diese beiden Eigenschaften zunutze macht.

Ableitung der EADF Wir wollen nun (5.2.10) aus Abschnitt 5.2 benutzen und auf zwei Dimensionen erweitern. Wir folgen damit effektiv [8]. Außerdem ändern wir das Argument der Funktion x und deren Namen zu der oben eingeführten Schreibweise $a : [0, 2\pi] \times [0, 2\pi] \rightarrow \mathbb{C}$ mit Werten $a(\varphi, \vartheta)$. In unserem Fall der Interpolation von Antennenantworten wissen wir, dass a in *beiden* Argumenten 2π -periodisch ist.

Die Bandlimitierung von a lässt sich formulieren, indem man fordert, dass die Bedingungen

$$A_\varphi(F_\varphi, \vartheta) = \int_{-\infty}^{+\infty} a(\varphi, \vartheta) \exp(-j2\pi F_\varphi \varphi) d\varphi = 0 \quad \text{für } F_\varphi > B_\varphi \quad \text{und alle } \vartheta \in [0, 2\pi] \quad \text{und} \quad (5.2.14)$$

$$A_\vartheta(\varphi, F_\vartheta) = \int_{-\infty}^{+\infty} a(\varphi, \vartheta) \exp(-j2\pi F_\vartheta \vartheta) d\vartheta = 0 \quad \text{für } F_\vartheta > B_\vartheta \quad \text{und alle } \varphi \in [0, 2\pi] \quad (5.2.15)$$

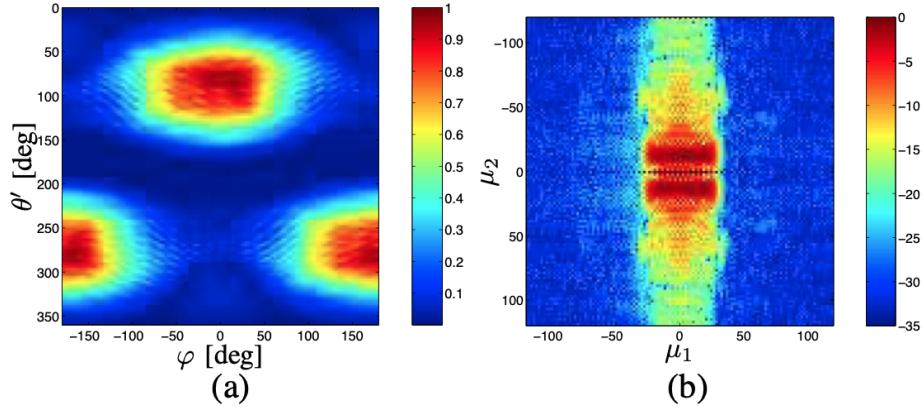


Abbildung 10: Links: Periodifiziertes 2D Array $a[n_\varphi, n_\theta]$ der gemessenen Amplituden eines einzelnen Patch-Elements. Rechts: Der Betrag der zugehörigen EADF, wobei hier $\mu_1 = k_\varphi$ und $\mu_2 = k_\theta$. Grafik aus [8]

erfüllt sein müssen. Nun lassen sich alle obigen Argumente in Abschnitt 5.2.1 „schnittweise“ auf eine abgetastete Version von a in der Form $a[n_\varphi, n_\theta]$ anwenden. Das heißt, wir landen schlussendlich bei einer Interpolations-Formel

$$a(\varphi, \theta) = \sum_{k_\varphi = -\frac{B_\varphi}{F_\varphi}}^{+\frac{B_\varphi}{F_\varphi}} \sum_{k_\theta = -\frac{B_\theta}{F_\theta}}^{+\frac{B_\theta}{F_\theta}} A[k_\varphi, k_\theta] \cdot \exp(j2\pi k_\varphi \varphi F_\varphi) \cdot \exp(j2\pi k_\theta \theta F_\theta), \quad (5.2.16)$$

welche eine absolut analoge (nicht als Gegenteil zu digitale) 2D-Version zu (5.2.10) darstellt. Auch in diesem Fall, können wir das 2D-Array $A[k_\varphi, k_\theta]$ durch eine 2D-DFT, bzw. der FFT [3], von den uniformen Samples $a[n_\varphi, n_\theta]$ der Funktion a effizient vorberechnen.

Um einen möglichst effizienten Algorithmus für die Auswertung der Interpolante zu erhalten, sollte man (5.2.16) geeignet umschreiben. Moderne Rechenarchitekturen und Scientific-Computing-Libraries sind auf schnelle Matrix-Vektor-Produkte optimiert. Nehmen wir an, wir wollen (5.2.16) für mehrere Winkelpaare $(\varphi_1, \theta_1), \dots, (\varphi_L, \theta_L)$ auswerten. Dann berechnen wir zunächst zwei 2D Arrays

$$D_\varphi = [\exp(j2\pi k_\varphi \varphi F_\varphi)]_{\ell=1, k_\varphi = -\frac{B_\varphi}{F_\varphi}}^{\ell=L, k_\varphi = \frac{B_\varphi}{F_\varphi}} \in \mathbb{C}^{L \times 2\frac{B_\varphi}{F_\varphi} + 1} \quad \text{und} \quad (5.2.17)$$

$$D_\theta = [\exp(j2\pi k_\theta \theta F_\theta)]_{\ell=1, k_\theta = -\frac{B_\theta}{F_\theta}}^{\ell=L, k_\theta = \frac{B_\theta}{F_\theta}} \in \mathbb{C}^{L \times 2\frac{B_\theta}{F_\theta} + 1}, \quad (5.2.18)$$

was uns erlaubt (5.2.16) in das folgende Vektor-Matrix-Vektor-Produkt

$$a(\varphi_\ell, \theta_\ell) = D_\varphi[\ell, :] \cdot A[:, :] \cdot D_\theta[\ell, :]^\top \quad (5.2.19)$$

umzuschreiben. Damit besteht der Interpolations-Algorithmus zunächst aus der Vorberechnung des Arrays A , sowie bei Ausführung dann aus der Berechnung von D_φ und D_θ , sowie der Auswertung von (5.2.19).

Man sieht hier der schön, dass die Laufzeitkomplexität von (5.2.19) maßgeblich von der räumlichen Bandbegrenzung der Antennen-Richtcharakteristik beeinflusst wird. Je höher die Bandbreite, desto höher ist nicht nur der Aufwand bei der Messung, sondern auch bei der Berechnung Interpolation. Eine alternative Form der Interpolation, welche diese eventuell nachteilige Eigenschaft nicht hat, ist in Abschnitt 6 dargestellt.

5.3 Schnelle Korrelation für Radar-Signalverarbeitung

Wir wollen noch eine weitere wichtige Anwendung für die in dieser Vorlesung entwickelten Signalverarbeitungstechniken betrachten. Für die Konzipierung eines RADAR-Systems macht man sich zu Nutze, dass sich ausbreitende elektromagnetische Wellen an metallischen Objekten gut reflektiert werden. Mit einer Antenne kann man beispielsweise ein Signal $x[\cdot]$ mit einem **transmitter (TX)** aussenden und mit einer zweiten Antenne am **receiver (RX)** auf das eventuelle Echo warten. Wird ein Objekt mit Entfernung d nun von dieser Welle getroffen und reflektiert einen Teil der Energie in Richtung **RX**, so enthält das empfangene Signal $y[\cdot]$ eine zeitlich verzögerte Kopie von $x[\cdot]$ und eine gewisse Menge an Rauschen, das von den Schaltkreisen an **TX** und **RX** eingeprägt wird. Es gilt also

$$y[\cdot] = \gamma x[\cdot - \tau] + w[\cdot],$$

wobei sich τ aus der Samplerate F_s , dem Abstand d und der Lichtgeschwindigkeit c via

$$\tau = F_s \cdot \frac{d}{c}$$

ergibt. Die Zahl $\gamma \in \mathbb{R}$ repräsentiert einerseits Verluste, die sich durch die Ausbreitung im Raum ergeben und gleichzeitig aus der „Reflektivität“ des Objektes. Für gewöhnlich hat ein RADAR zwei Aufgaben. Einerseits soll detektiert werden, ob ein Objekt von unserem RADAR angeleuchtet wurde und im Idealfall, wie groß d ist. Wir wollen uns also Signalverarbeitungstechniken ansehen, die die Bestimmung von d erlauben.

5.3.1 Zyklische Korrelationen

Eine Möglichkeit, um d zu erhalten basiert auf Korrelation. Hierzu schränken wir zunächst die Menge der Sendesequenzen ein, indem wir fordern, dass $x[\cdot]$ periodisch mit Periodendauer N ist. Die Korrelation $r_{x,y}$ von zwei periodischen Sequenzen $x[\cdot]$ und $y[\cdot]$ ist gegeben durch

$$r_{x,y}[\ell] = \frac{1}{N} \sum_{n=0}^{N-1} x[n]y[n - \ell]. \quad (5.3.1)$$

Stellen wir uns vor, dass unser RADAR-System die Sendesequenz $x[\cdot]$ erzeugt, indem es eine endliche Sequenz $x \in \mathbb{R}^N$ wiederholt aussendet, so ist $x[\cdot]$ periodisch mit Periodendauer N und auch $y[\cdot]$ – nach den ersten N Samples. Die Frage ist nun, was uns diese Korrelation nützt. Man kann sich überlegen, dass

$$|r_{x,y}[\ell]| \leq \sqrt{r_{x,x}[0] \cdot r_{y,y}[0]} = \sqrt{\mathcal{E}(x)\mathcal{E}(y)},$$

siehe Gleichung (3.1.1). Das heißt, dass die sog. *Kreuzkorrelation* immer durch die Werte der sog. *Autokorrelation* wie oben angegeben beschränkt ist. Im Spezialfall $y = x$ ergibt sich

$$|r_{x,x}[\ell]| \leq r_{x,x}[0],$$

was heißt, dass die **Auto-Correlation Function (ACF)** ihr Maximum immer am Wert $\ell = 0$ annimmt.

Betrachten wir nun wieder unser RADAR-System, so könnte es sich als sinnvoll erweisen, am Empfänger das Signal $y[\cdot]$ mit dem Sendesignal zu korrelieren, also

$$r_{y,x}[\ell] = \gamma r_{x,x}[\cdot - \tau][\ell] + r_{w,x}[\ell] = \gamma r_{x,x}[\ell - \tau] + r_{w,x}[\ell]$$

berechnen. Da $r_{x,x}[\cdot]$ bei 0 ein lokales Maximum hat, so hat $r_{x,x[-\tau]}[\cdot]$ ein lokales Maximum bei τ ! Wir könnten also einfach den Maximalen Wert in $r_{y,x}[\cdot]$ suchen und dessen Stelle in d umrechnen.

Die Berechnung von $r_{y,x}[\cdot]$ für N Werte erfordert in der Größenordnung N^2 viele **FLOPs**. Man kann aber ziemlich einfach einsehen, dass man die Korrelation als Faltung via

$$r_{x,y} = x[\cdot] \circledast y[-\cdot] \quad (5.3.2)$$

ausdrücken kann. Außerdem gilt für eine Sequenz $x[\cdot]$ und deren **DFT** $X[\cdot]$, dass

$$\text{DFT}(x[-\cdot]) = X[\cdot]^*, \quad (5.3.3)$$

also Umkehrung im Zeitbereich entspricht Konjugation im Frequenzbereich. Demzufolge können wir Abschnitt 4.3.2 zusammen mit (5.3.2) verwenden, um $r_{x,y}[\cdot]$ mit $N \log(N)$ vielen **FLOPs** zu berechnen. Das heißt, dass für die Signalverarbeitung effiziente Methoden bereitstehen, solange der Ansatz der Berechnung der Kreuzkorrelation des Empfangssignals mit der Sendesequenz als theoretisch fundiert herausstellt.

In Codeschnipsel 28 sind einige verschiedene Sendesignale miteinander verglichen, indem zwei Reflexionen mit verschiedenen Abständen simuliert werden und aus dem resultierenden Empfangssignal $y[\cdot]$ wiederum $r_{x,y}$ bestimmt wird. Wie man sieht, hat das Sendesignal einen signifikanten Einfluss auf die „Form“ der **ACF**, welche wiederum einen Einfluss auf die visuelle Qualität der Funktion $r_{x,y}[\cdot]$ hat. Nutzt man nur einen einfachen Sinus als Sendesignal (Codeschnipsel 28, oben), so kann man gar keine Information über den Abstand von eventuellen Reflexionen finden. Der Grund ist, dass sich Zeitversatz nur mit Signalen schätzen lässt, die eine gewisse Bandbreite abdecken, also mehrere Frequenzen gleichzeitig belegen.

Mit einer sinc-Funktion (Codeschnipsel 28 mitte) belegen wir bekanntermaßen im Frequenzbereich einen kontinuierlichen Bereich, was dazu führt, dass wir in der Tat sehr ausgeprägte lokale Maxima bei den wahren τ erhalten. Je schmäler die Hauptkeule der sinc-Funktion im Zeitbereich, desto breiter ist der abgedeckte Frequenzbereich, was leicht durch Modifikation von Codeschnipsel 28 überprüft werden kann. Es ist jedoch zu bemerken, dass ein im Zeitbereich „schmäler“ sinc an die Hardware, die ihn erzeugt einige unangenehme Herausforderungen stellt, weshalb solch ein Sendesignal in der Praxis einige Nachteile aufweist.

Schlussendlich sehen wir (Codeschnipsel 28, unten), dass zufällige Sendesignale auch gute Eigenschaften bezüglich $r_{x,y}[\cdot]$ liefern. Gleichzeitig haben diese den Vorteil, dass keine großen Sprünge in der Amplitude notwendig sind, da sich das Signal bei maximaler Amplitude nicht sehr weit von seinem Mittelwert entfernt¹⁵.

Speziell für unsere Anwendung im RADAR streben wir also an, dass die **ACF** $r_{x,y}[\cdot]$ ein ausgeprägtes lokales Maximum hat, also einerseits sehr schnell abfällt, und andererseits auch wenig bis keine Nebenmaxima erzeugt, weil sich dies beispielsweise in Codeschnipsel 24 als Problem herausgestellt hat. Im Idealfall finden wir eine Sendesequenz, die sich einerseits gut für RADAR eignet und gleichzeitig sich beispielsweise effizient in Hardware erzeugen lässt.

¹⁵https://en.wikipedia.org/wiki/Crest_factor

```

def gen_receive(x, tau, gamma):
    return np.sum(
        [
            np.roll(x, tt) * gg
            for tt, gg in zip(tau, gamma)
        ],
        axis=0,
    )

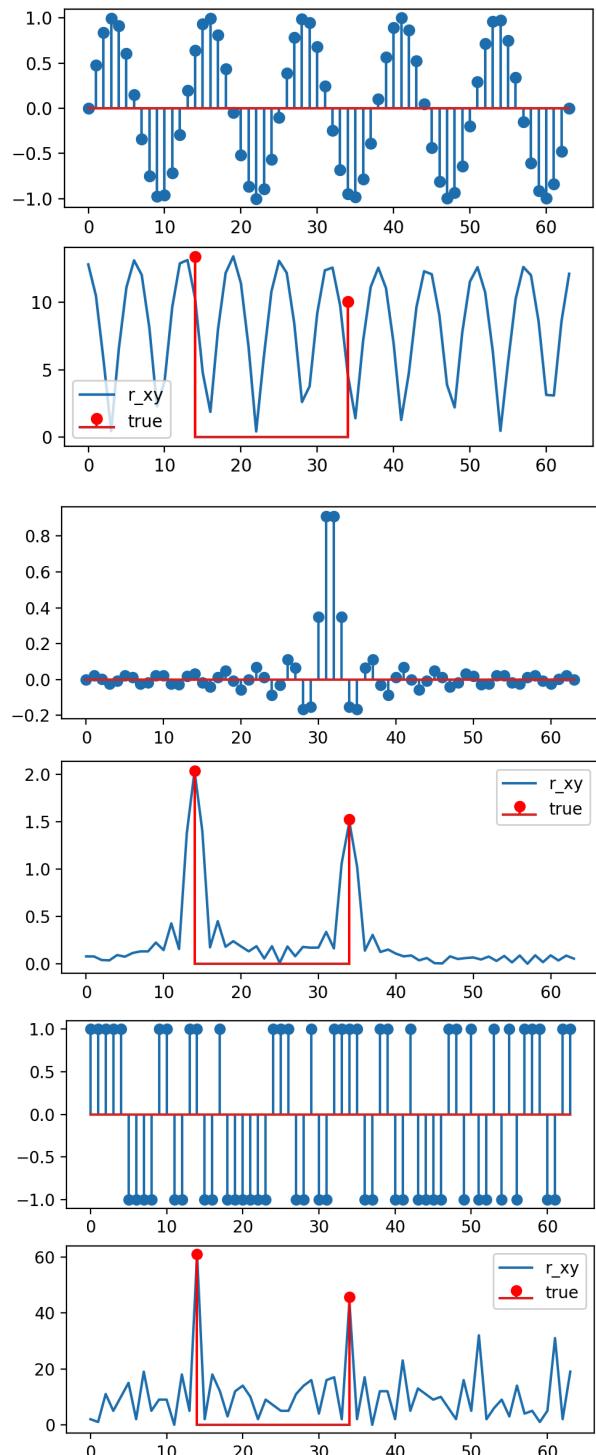
N = 64
tau = [14, 34]
gamma = [1, 0.75]

# x = np.sin(np.linspace(0, 10 * np.pi, N))
# x = np.sinc(np.linspace(-15, +15, N))
x = np.sign(np.random.randn(N))
X = np.fft.fft(x)

y = gen_receive(x, tau, gamma)
Y = np.fft.fft(y)

# Gleichung (5.3.2)
r_xy = np.fft.ifft(Y * X.conj())

```



Codeschnipsel 28: Vergleich verschiedener Sendesignale $x[\cdot]$ bezüglich der resultierenden Korrelation $r_{xy}[\cdot]$, siehe [dsv/code/radar_1.py](#)

5.3.2 Erzeugung der Sendesequenz

Wir wollen im Folgenden eine mögliche Art der Sequenzerzeugung genauer untersuchen, da sie beide obigen Anforderungen erfüllt. Hierzu betrachten wir sogenannte **Linear Feedback Shift Registers (LFSRs)**, wie in Abbildung 11 dargestellt. Diese bestehen aus L Registern/Taps, welche jeweils ein Bit beinhalten. Die Idee ist nun dieses LFSR als diskretes System zu betrachten, dessen Werte in der Menge $\mathbb{Z}_2^L = \{0, 1\}^L$ liegen können und diese Menge \mathbb{Z}_2 mit der binären Addition und Multiplikation ausgestattet wurde. In jedem Takt werden die Werte von Links nach Rechts verschoben, wobei der Zustand des letzten Registers als Ausgabe fungiert. Weiterhin werden einige der Zustände durch binäre Addition miteinander verknüpft und an den Eingang, also das erste Register, als Feedback zurückgeführt. Es ist demnach auch leicht zu sehen, dass solch ein System sehr leicht und effizient in diskreter Hardware implementiert werden kann.

Zum Zeitpunkt n ist damit die Ausgabe

$$y[n] = x[n] + a_1x[n - 1] + \dots + a_{L-1}x[n - (L - 1)] + x[n - L],$$

wobei die Koeffizienten $a_1, \dots, a_{L-1} \in \mathbb{Z}_2$, also definieren, welche Werte der Register für die Berechnung des Feedbacks genutzt werden. Man kann das System zum Zeitpunkt n durch die Werte $x[n], \dots, x[n - L]$ also eine Binärzahl mit L Stellen/Bits beschreiben. Kennt man all diese Zustände, ist es auch möglich den Nächsten zu berechnen, indem auf die Zahl ein Bitshift und eine „maskierte“ Addition über die Bits des aktuellen Zustands angewandt wird.

Man sieht nun, dass sich mit L binären Registern 2^L Zustände abbilden lassen, und dass das System maximal $2^L - 1$ von diesen durchlaufen kann, wenn man nicht im Zustand $0 \dots 0$ beginnt. Die Werte $y[n]$ müssen demnach periodisch sein, denn sobald sich ein einzelner der $2^L - 1$ möglichen Zustände ein zweites Mal einstellt, folgen ihm wieder dieselben wie beim ersten Auftreten. Demnach ist auch die maximale Periode mit $2^L - 1$ beschränkt. Die Frage ist nun, ob es Koeffizienten-Konfigurationen gibt, für welche die maximale Periodenlänge erreicht wird.

Hierzu betrachten wir die z -Transformation des Systems, dann finden wir

$$H(z) = 1 + a_1z^1 + a_2z^2 + \dots + a_{L-1}z^{L-1} + z^L,$$

woran wir wiederum sehen, dass das System nur durch die Wahl der Koeffizienten $a_1, \dots, a_{L-1} \in \mathbb{Z}_2$ bestimmt wird. Man kann zeigen, dass falls die Transferfunktion H ein sogenanntes irreduzibles Polynom

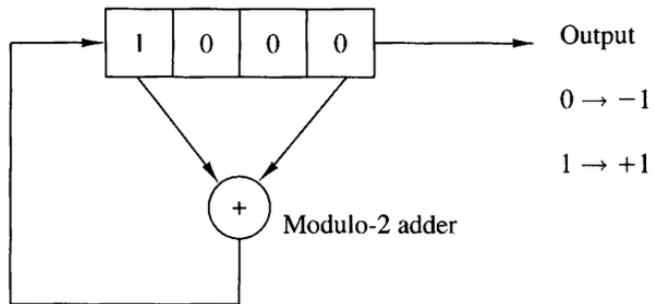


Abbildung 11: Lineares Feedback Shift Register, Quelle [1]

¹⁶ ist, so hat die von diesem LFSR erzeugte Sequenz maximale Länge, also $2^L - 1$. Das heißt, dass die algebraischen Eigenschaften der z-Transformierten der Transferfunktion H wieder Aufschluss darüber geben, welche Eigenschaften das LFSR als System betrachtet besitzt. In Codeschnipsel 29 wird ausgehend von einem Startwert des Registers (`0x1`) und einer sogenannten Tapkonfiguration (`0xD`) eine **Maximum Length Binary Sequence (MLBS)** durch einmaliges Simulieren des LFSR erzeugt. Die entstehende Folge hat ähnliche Eigenschaften, wie eine „wirklich“ zufällige Zahlenfolge, doch entsteht aus einem vollständig deterministischem Prozess.

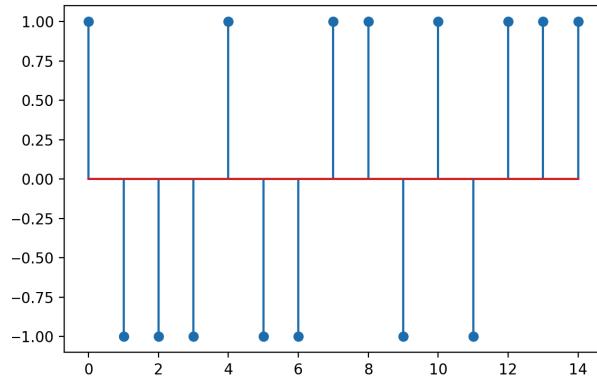
```
def gen_MLBS(taps: int, start: int):
    numTaps = int(
        np.ceil(np.log2(taps)).astype(int)
    )
    length = 2**numTaps - 1
    state = start

    sequence = []
    for ll in range(length):
        # move last bit to output
        sequence.append(state & 1)
        feedback = state
        for tt in range(1, numTaps - 1):
            if (taps >> (numTaps - tt - 1)) & 1:
                feedback ^= state >> tt
        # keep only last bit of combined XOR
        feedback &= 1
        # shift registers
        state = state >> 1
        # feedback from back to front
        state |= feedback << (numTaps - 1)

    sequence = 2 * (np.array(sequence) - 0.5)
    return sequence

if __name__ == "__main__":
    sequence = gen_MLBS(0xC + 1, 1)
```

Codeschnipsel 29: Simulation eines LFSR mittels binären Operationen, siehe dsv/code/mlbs.py



5.3.3 MLBS als Sendesequenz

Zum Abschluss wollen wir nun noch untersuchen, wie gut sich die so gewonnenen Sequenzen für RADAR verwenden lassen. Hierzu betrachten wir die Ausgabe von Codeschnipsel 30, wo wir das Wissen aus Codeschnipsel 28 und Codeschnipsel 29 kombiniert haben, um einerseits „effizient“ eine Sendesequenz zu erzeugen und andererseits, um die RADAR-Signalverarbeitung effizient zu gestalten.

¹⁶Siehe, https://de.wikipedia.org/wiki/Irreduzibles_Polynom. Irreduzible Polynome treten im Zusammenhang mit endlichen Körpern auf, beispielsweise \mathbb{Z}_2 , wo eine andere Art Polynomdivision angewandt werden kann/muss. Irreduzible Polynome nehmen im Grunde hier die Rolle von Primzahlen ein, da sich jedes Polynom in ein Produkt aus irreduziblen Polynomen zerlegen lässt.

Wir sehen, dass das LFSR eine Sequenz erzeugt, welche sehr gut zu unseren Anforderungen passt. Einerseits ist die Aussteuerung des Signals sehr begrenzt und andererseits hat die ACF ideale Eigenschaften, um Abstände von Reflexionen zu bestimmen. Man kann sogar beweisen, dass für eine Sequenz maximaler Länge $x[\cdot]$ gilt, dass

$$r_{x,x}[\ell] = \begin{cases} 1 & \text{für } \ell = 0, \\ -\frac{1}{2^L-1} & \text{sonst.} \end{cases} \quad (5.3.4)$$

Damit ist das lokale Maximum immer um den Faktor $N = 2^L - 1$ höher als die Werte im Rest der Korrelationsfunktion $r_{x,x}$. Dies bedeutet einerseits, dass dieses Verfahren auch mit eventuellem Messrauschen sehr gut umgehen kann. Andererseits kann man auch durch den Einsatz von sehr langen Sequenzen, also mehr Registern im LFSR, mit sehr wenig Sendeleistung durch die Transformation von $y[\cdot]$ nach $r_{x,y}[\cdot]$ immer noch Reflexionen detektieren.

```

from mlbs import gen_MLBS
from radar1 import gen_receive

x = gen_MLBS(0x44, 1)

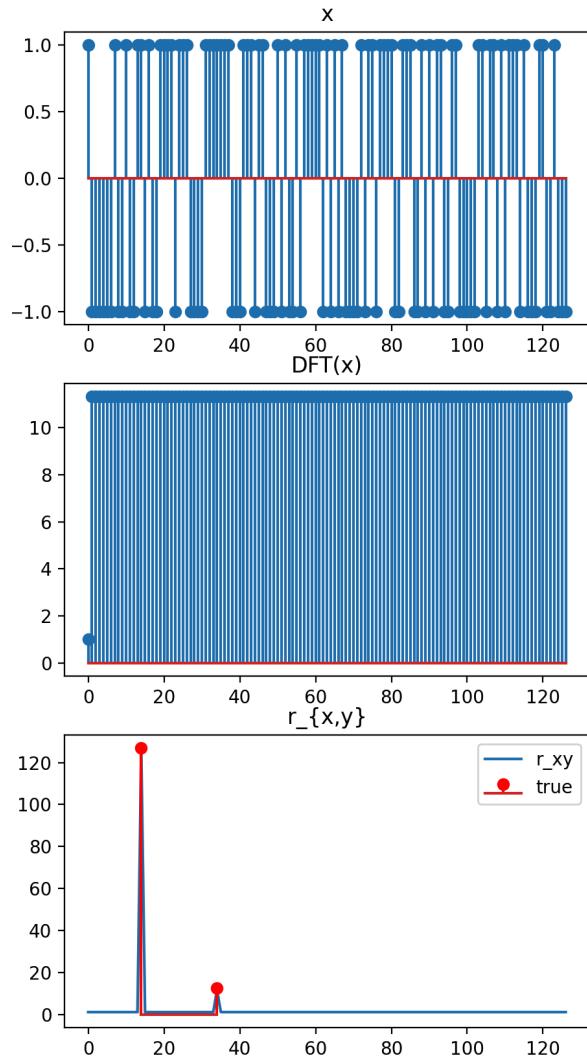
tau = [14, 34]
gamma = [1, 0.3]

N = x.size

x = x
X = np.fft.fft(x)

y = gen_receive(x, tau, gamma) + (
    0.5 * np.random.randn(x.size)
)
Y = np.fft.fft(y)

```



Codeschnipsel 30: Simulation eines RADAR-Systems basierend auf MLBS, die von LFSR erzeugt wurden.
Oben verrauschtet Empfangssignal $y[\cdot]$, mitte DFT $Y[\cdot]$, unten $r_{x,y}$, siehe [dsv/code/radar2.py](#)

6 Signalverarbeitung mit B-Splines

Eine Grundvoraussetzung für eine praktisch nützliche digitale Signalverarbeitung ist die Möglichkeit zwischen dem analogen und digitalen Bereich wechseln zu können. Hierbei sollte man auch genau quantifizieren können, ob bei diesem Prozess Informationen verloren gehen, oder wie man garantieren kann, dass diese Umwandlung verlustfrei vonstatten geht. Meist nutzt man hierfür das Sampling Theorem 2.1. Die hieraus resultierende sogenannte Nyquist-Sampling-Theorie fußt bekanntermaßen auf der Repräsentation von bandbegrenzten Signalen durch hinreichend dichte äquidistante Abtastwerte.

Es gibt jedoch auch einige Nachteile von Nyquist-Sampling, die aus dessen Annahmen und der daraus folgenden Verarbeitung entstehen. Einerseits kann ein endliches Signal im Allgemeinen *nicht* bandbegrenzt sein. Weiterhin entstehen durch die Bandlimitierung von Signalen Gibbs-Artefakte, siehe Codeschnipsel 19, die besonders bei der Bilderverarbeitung nicht erwünscht sind. Geht es um die Auswertung $x(t)$ eines Signals x zwischen den aufgenommenen Samples $x[n]$, also um Interpolation, hat man das Problem, dass die sinc-Funktion nur sehr langsam mit Rate $1/t$ abfällt. Diese Eigenschaft führt dazu, dass man für die Bestimmung eines Wertes $x(t)$ mit einer Genauigkeit von 1% etwa 100 um t benachbarte Samples betrachten muss. Das heißt, vor allem bei 2D-Interpolation, siehe Abschnitt 5.2, skaliert der resultierende Rechenaufwand nicht sehr günstig, falls hohe Genauigkeit benötigt wird.

Aus diesem Grund möchten wir uns eine alternative Sampling-Theorie genauer ansehen – die B-Splines [9]. Wir führen zunächst die auf Polynomen basierende Signalverarbeitung ein und vergleichen sie anschließend zur bereits bekannten Nyquist-Theorie.

6.1 B-Splines als Polynome

Allgemein bezeichnet man stückweise definierte und stetig differenzierbare Polynome als Splines. Man nennt die Stellen an denen zwei unterschiedliche Polynome zusammenstoßen als Knoten. Ein Spline der Ordnung $\ell \in \mathbb{N}$ ist ein Polynom vom Grad ℓ , ist also von der Form

$$p(t) = a_\ell t^\ell + a_{\ell-1} t^{\ell-1} + \cdots + a_1 t + a_0. \quad (6.1.1)$$

Ein Spline ist nun eine Funktion $s(t)$, welche für Knoten $n = 1, 2, \dots$ definiert ist durch

$$s(t) = \begin{cases} p_1(t) & \text{für } x \in [1, 2], \\ p_2(t) & \text{für } x \in [2, 3], \\ \vdots & \end{cases} \quad (6.1.2)$$

wobei sich die Glattheit durch die Forderung ergibt, dass die Funktion und ihre Ableitungen an den Knoten stetig sei, also

$$\lim_{t \rightarrow n^-} s^{(m)}(t) = \lim_{t \rightarrow n^+} s^{(m)}(t) \quad (6.1.3)$$

erfüllt ist, wobei $s^{(m)}$ für $m \geq 0$ die m -te Ableitung des Splines s repräsentiert. In einer Arbeit [10], die sogar dem berühmten Paper von Shannon vorausgeht, beschreibt Schoenberg, dass sich diese Splines der Ordnung ℓ via

$$s(t) = \sum_{k \in \mathbb{Z}} c[k] \beta^\ell(t - k) \quad (6.1.4)$$

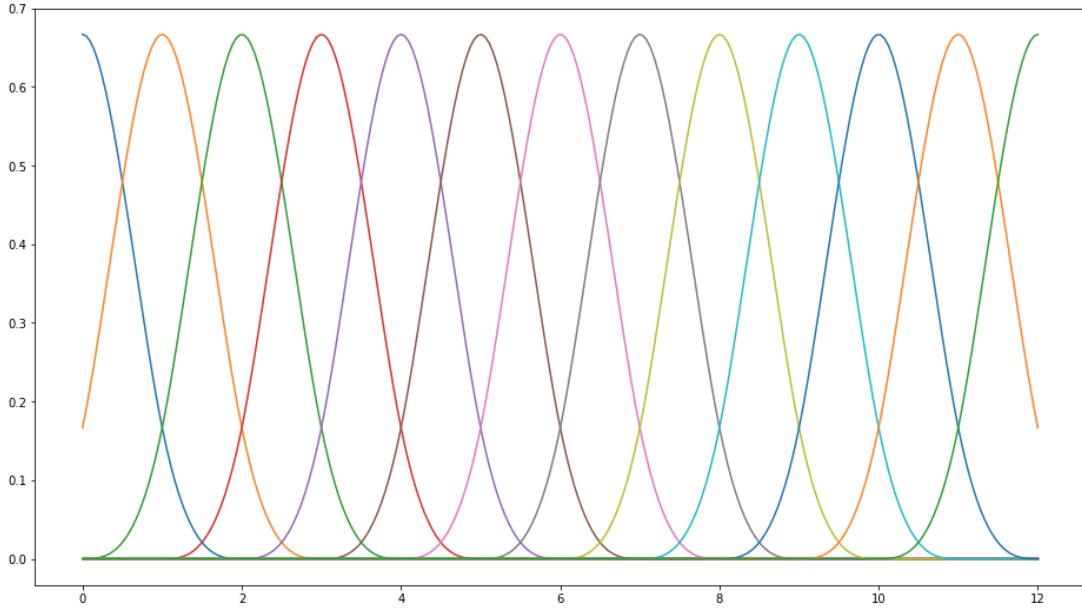


Abbildung 12: Kubische B-Splines für Abtastung an den Werten $n = 0, \dots, 12$.

darstellen lassen. Hierbei ist die Funktion $\beta^\ell : \mathbb{R} \rightarrow \mathbb{R}$ definiert als eine iterierte Faltung einer Rechteckfunktion via

$$\beta^\ell = \underbrace{\beta^0 * \dots * \beta^0}_{(\ell+1) \text{ mal}}, \quad \text{wobei } \beta^0(t) = \begin{cases} 1, & |t| < \frac{1}{2} \\ \frac{1}{2}, & |t| = \frac{1}{2} \\ 0, & \text{sonst.} \end{cases} \quad (6.1.5)$$

In Abbildung 12 sind die Funktionen β^ℓ für $\ell = 0, \dots, 3$ dargestellt. Man erkennt sehr gut, dass der Grad der Glattheit von der Ordnung des Splines abhängt und dass die Funktionswerte $\beta^\ell(t)$ für $|t| > \ell + 1/2$ verschwinden. Man spricht von Funktionen mit *kompaktem Träger*. Das heißt die Summation in (6.1.4) ist für fixes $t \in \mathbb{R}$ *endlich* und ist auf $\ell + 1$ Summanden beschränkt! Wir wollen nun eine explizite Formel für β^ℓ entwickeln. Hierzu betrachten wir die Fourier-Transformation

$$B^\ell(\omega) = \left(\frac{\sin(\omega/2)}{(\omega/2)} \right)^{\ell+1} = \frac{(\exp(j\omega/2) - \exp(-j\omega/2))^{\ell+1}}{(j\omega)^{\ell+1}} \quad (6.1.6)$$

mit einigen Rechentricks (siehe [9, Box 1.]) kann man dies so lange umformen, bis man

$$\beta^\ell(t) = \frac{1}{\ell!} \sum_{p=0}^{\ell+1} \binom{\ell+1}{p} (-1)^p \left(t - p + \frac{\ell+1}{2} \right)_+^\ell \quad \text{mit } (x)_+ = \begin{cases} x, & \text{für } x \geq 0, \\ 0, & \text{sonst,} \end{cases} \quad (6.1.7)$$

erhält. Damit ist β^ℓ wirklich ein Polynom ℓ -ten Grades. Die Stetig- und Differenzierbarkeit muss man sich aber noch separat überlegen.

Weiterhin kann man zeigen, dass folgende Formeln für Differentiation und Integration von B-Splines gelten:

$$(\beta^\ell)'(t) = \beta^{\ell-1}(t + 1/2) - \beta^{\ell-1}(t - 1/2), \quad \int_{-\infty}^t \beta^\ell(s) ds = \sum_{p=0}^{+\infty} \beta^{\ell+1}(t - 1/2 - p). \quad (6.1.8)$$

Das heißt, dass man auch einen kompletten Spline s differenzieren und integrieren kann, indem man nutzt, dass sowohl Differentiation, als auch Integration lineare Operationen sind. Es gilt also mit (6.1.4) und (6.1.8) beispielsweise für die Differentiation, dass

$$s'(t) = \sum_{k \in \mathbb{Z}} c[k] (\beta^\ell)'(t - k) = \sum_{k \in \mathbb{Z}} c[k] (\beta^{\ell-1}(t + 1/2 - k) - \beta^{\ell-1}(t - 1/2 - k)). \quad (6.1.9)$$

Diese Eigenschaft macht man sich auch für kompliziertere Operationen, wie Rotationen und Verzerrungen auf einem Spline s zunutze.

6.2 Kubische B-Spline Interpolation

Wir möchten uns eine spezielle Version der B-Splines genauer Ansehen, da diese in der Anwendung den Spagat zwischen Komplexität und Approximationsgüte sehr gut hinbekommen. Wir setzen hierzu $\ell = 3$ und erhalten somit ein Polynom dritten Grades der Form

$$\beta^3(t) = \begin{cases} \frac{2}{3} - |t|^2 + \frac{|t|^3}{2} & \text{für } |t| < 1 \\ \frac{(2-|t|)^3}{6}, & \text{für } |t| \in [1, 2) \\ 0, & \text{für } |t| > 2, \end{cases} \quad (6.2.1)$$

welche in Codeschnipsel 31 auch einmal implementiert wurde.

In Analogie zum bekannten Nyquist-Sampling wollen wir untersuchen, wie wir aus endlich vielen gegebenen Abtastwerten $x[n]$ mit $n \leq N$ eine Darstellung wie in (6.1.4) herleiten können, welche die abgetasteten Werte exakt interpoliert. Aufgabe ist es also aus $x[n]$ die Folge $c[k]$ zu bestimmen.

Hierzu benötigen wir die sogenannte Interpolationsbedingung, siehe (5.2.1), welche für eine zu interpolierende Funktion x und ihre Abtastwerte $x[n] = x(n)$ fordert, dass

$$x(n) = x[n] = s(n) = \sum_{k \in \mathbb{Z}} c[k] \beta^3(n - k) \quad (6.2.2)$$

gilt. Wir fordern also *exakte* Interpolation. Nun könnte man für die Bestimmung der Folge $c[k]$ ein lineares Gleichungssystem aufstellen, welches die Form

$$\mathbf{B} \cdot \mathbf{c} = \mathbf{x} \quad (6.2.3)$$

```
def cubic_spline(t: np.ndarray) -> np.ndarray:
    s1: np.ndarray = (
        2.0 / 3.0
        - np.abs(t) ** 2
        + 0.5 * np.abs(t) ** 3
    )
    s2: np.ndarray = (2 - np.abs(t)) ** 3 / 6
    s3: np.ndarray = np.zeros_like(t)
    return (
        s3
        + s1 * (np.abs(t) < 1)
        + s2 * (np.abs(np.abs(t) - 1.5) <= 0.5)
    )
```

Codeschnipsel 31: Implementierung von (6.2.1), siehe [dsv/code/bsplines_eval.py](#)

hat, wobei die Systemmatrix B durch die Auswertung der B-Splines an den Stellen n bestimmt ist und der Vektor x den Werten $x[n]$ entspricht. Dem endlichen Träger der Funktionen β^ℓ ist es zu verdanken, dass die Matrix B mit nur wenigen von 0 verschiedenen Werten besetzt ist (genauer: band-diagonal) und demzufolge effizient invertierbar ist. Es ist also nicht „schwer“ $c = B^{-1}y$ zu berechnen. Doch die Anwendung von B^{-1} ist numerisch instabil, weshalb wir einen alternativen Weg einschlagen, der auf inverser Filterung beruht.

Sehen wir uns (6.2.2) genauer an. Wir finden, dass sich diese Gleichung nach Definition von $\beta[k] = \beta^3(k)$ als Faltung via

$$x[n] = (c * b)[n] \quad (6.2.4)$$

schreiben lässt. Nach Transformation in den z -Bereich erhalten wir

$$X(z) = C(z) \cdot B(z) \Rightarrow C(z) = \frac{X(z)}{B(z)}, \quad (6.2.5)$$

was uns motiviert eine Darstellung von $1/B(z)$ im Zeitbereich herzuleiten. Für die kubischen B-Splines folgt, dass

$$B(z) = \frac{z + 4 + z^{-1}}{6} \Rightarrow \frac{1}{B(z)} = 6 \left(\frac{1}{1 - z_1 z^{-1}} \right) \left(\frac{-z_1}{1 - z_1 z} \right) \quad (6.2.6)$$

gilt. Wobei man zeigen kann, dass $z_1 = \sqrt{3} - 2 < 1$ gilt. Wir betrachten nun $1/B(z)$ als einen Filter, der auf die Abtastwerte $x[n]$ angewandt werden soll. Aus (6.2.6) erkennen wir, dass $1/B(z)$ ein Filter ohne Nullstellen ist und als Hintereinanderausführung von zwei rekursiven Filtern betrachtet werden kann. Wir erhalten mit $c^-[k] = c[k]/6$, dass sich $1/B(z)$ durch

$$c^+[k] = x[n] + z_1 c^+[k-1] \quad \text{für } k = 1, \dots, N-1 \quad (6.2.7)$$

$$-c[k]/6 = c^-[k] = z_1 (c^-[k+1] - c^+[k]) \quad \text{für } k = N-2, \dots, 0 \quad (6.2.8)$$

ausdrücken lässt. Diese Methode der kausalen und anti-kausalen Filterung ist deutlich effizienter und stabiler, als (6.2.3), da beispielsweise keine Divisionen notwendig sind. Nun ist es noch notwendig Anfangswerte für $c^+[k]$ und $c^-[k]$ zu finden. Dies ist aufgrund der Endlichkeit von x nicht ohne Weiteres möglich. Man sieht, dass die Impulsantwort von c^+ eine abklingende Exponentialfunktion ist, also gilt

$$c^+[0] = \sum_{k=0}^{\infty} x[-k] z_1^k \approx \sum_{k=0}^K x[-k] z_1^k, \quad (6.2.9)$$

wobei man $K \in \mathbb{N}$ so wählen kann, dass $z_1^K \leq \varepsilon$ erfüllt ist, siehe Codeschnipsel 14. Nach Ausführung von c^+ kann man $c^-[N-1]$ durch

$$c^-[N-1] = \frac{z_1}{1 - z_1^2} (c^+[N-1] + z_1 c^+[N-2]) \quad (6.2.10)$$

effizient und exakt initialisieren. Beides wurde in Codeschnipsel 32 beispielhaft implementiert und eine mögliche Ausgabe nach Auswertung von (6.1.4) ist in Abbildung 13 dargestellt.

```

def filter_coeffs(x_n: np.ndarray) -> np.ndarray:
    z_1: float = np.sqrt(3) - 2.0
    c_k_p: np.ndarray = np.zeros_like(x_n)
    c_k_m: np.ndarray = np.zeros_like(x_n)

    c_k_p[0] = np.sum(
        x_n * np.power(z_1, np.arange(x_n.size)))
    )

    for kk in range(1, x_n.size):
        c_k_p[kk] = x_n[kk] + z_1 * c_k_p[kk - 1]

    c_k_m[-1] = (
        z_1
        * (c_k_p[-1] + z_1 * c_k_p[-2])
        / (1.0 - z_1**2)
    )

    for kk in range(2, x_n.size + 1):
        c_k_m[-kk] = z_1 * (
            c_k_m[-kk + 1] + c_k_p[-kk]
        )

    return -6 * c_k_m

```

Codeschnipsel 32: Berechnung der B-Spline Koeffizienten, siehe [dsv/code/bsplines_coeffs.py](#)

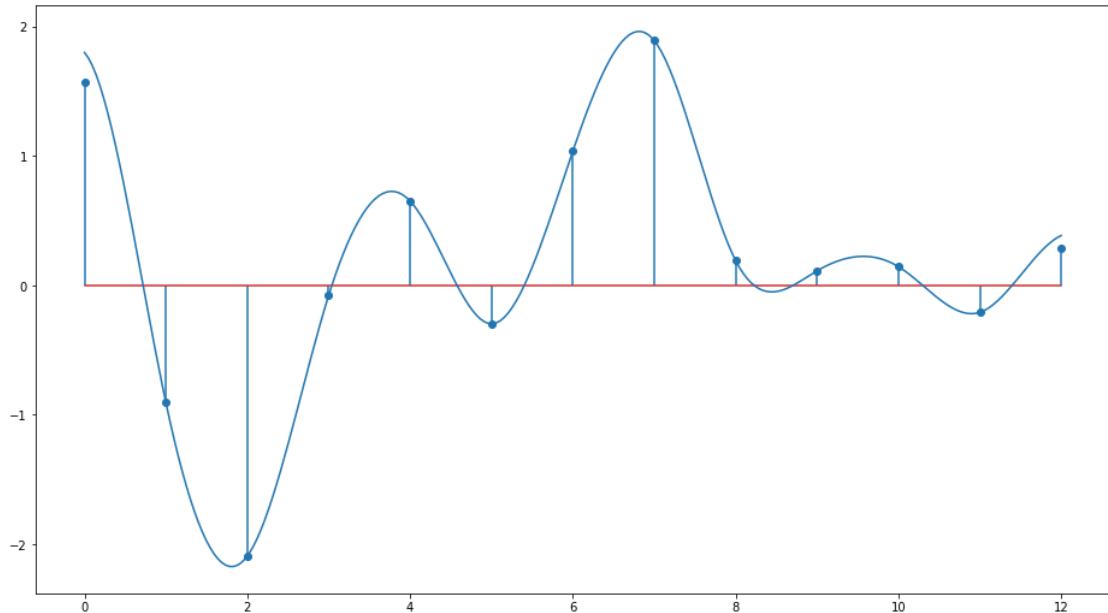


Abbildung 13: Kubische B-Spline-Interpolation für Abtastung an den Werten $n = 0, \dots, 12$.

6.3 Verbindung zur Nyquist-Sampling-Theorie

Wir wollen als Abschluss eine Verbindung zum Sampling und der Interpolation [1, Kapitel 6.1] von bandbegrenzten Funktionen mit endlicher Energie ziehen. Nehmen wir als Wiederholung zunächst an, dass das bandbegrenzte Signal x_a mit endlicher Energie und Fourier-Transformation X_a mindestens kritisch mit Rate F_s zu den Werten $x[n]$ abgetastet wurde. Wir bezeichnen mit X die DTFT von $x[n]$. Dann gilt als Zusammenhang zwischen den beiden Spektren, wie in Abschnitt 2.5, dass

$$X(f) = F_s \sum_{k=-\infty}^{+\infty} X_a((f - k)F_s), \quad (6.3.1)$$

was die Periodifizierung des Frequenzbereiches durch Abtastung ausdrückt. Nach Annahme der kritischen Abtastung findet hier kein Aliasing statt, sodass für $f \in [-F_s/2, +F_s/2]$ gilt, dass $F_s \cdot X_a(f) = X(f)$. Außerdem können wir das Spektrum der abgetasteten Werte X durch

$$X(f) = \sum_{n=-\infty}^{+\infty} x[n] \exp(-j2\pi fn/F_s) \quad (6.3.2)$$

ausdrücken. Nun können wir das analoge Signal x_a in Abhängigkeit von den Abtastwerten darstellen. Es gilt mit $T = 1/F_s$ und nach Theorem 2.1, dass

$$x_a(t) = \sum_{n=-\infty}^{+\infty} x[n] \frac{\sin(\pi(t - nT)/T)}{\pi(t - nT)/T}.$$

Das heißt, dass die Funktion $g : \mathbb{R} \rightarrow \mathbb{R}$ mit

$$g(t) = \frac{\sin(\pi t/T)}{(\pi t/T)} \quad (6.3.3)$$

als *Interpolations-Kernel* von bandbegrenzten und abgetasteten Funktionen betrachtet werden kann. Siehe Abschnitt 5.2.2 für eine Anwendung dieser Art der Interpolation.

Nun können wir eine analoge Rechnung für die B-Splines durchführen, indem wir einen Filter $b^{-1}[k]$ als inverse Z-Transformation von $1/B(z)$ aus (6.2.5) definieren. Dann gilt

$$c[k] = (b^{-1} * x)[k], \quad (6.3.4)$$

was wir in (6.2.2) einsetzen und dann

$$s(t) = \sum_{n \in \mathbb{Z}} (b^{-1} * x)[n] \beta^\ell(t - n) = \sum_{n \in \mathbb{Z}} x[n] \sum_{p \in \mathbb{Z}} b^{-1,\ell}[p] \beta^\ell(t - n - p) = \sum_{n \in \mathbb{Z}} x[n] h^\ell(t - n) \quad (6.3.5)$$

erhalten, wobei wir analog zu (6.3.3) den Interpolationskernel $h : \mathbb{R} \rightarrow \mathbb{R}$ durch

$$h^\ell(t) = \sum_{p \in \mathbb{Z}} b^{-1,\ell}[p] \beta^\ell(t - p) \quad (6.3.6)$$

definiert haben. Man kann zeigen, dass $\lim_{\ell \rightarrow \infty} h^\ell = g$. Das heißt, dass die sinc-Interpolation als Grenzwert der B-Spline Interpolation aufgefasst werden kann – oder andersherum – die B-Spline-Interpolation als Approximation der sinc-Interpolation. Das heißt, dass auch im Frequenzbereich Konvergenz in der Form

$$H^\ell(\omega) = \left(\frac{\sin(\omega/2)}{\omega/2} \right)^{\ell+1} \frac{1}{B^\ell(\exp(j\omega))} \xrightarrow{\ell \rightarrow \infty} \text{rect}(\omega) \quad (6.3.7)$$

gegeben sein muss.

Zusammenfassend kann man sagen, dass B-Splines einen alternativen Zugang zu digitaler Signalverarbeitung bieten, welcher eng mit dem des Nyquist-Samplings verknüpft ist und als Approximation von diesem gesehen werden kann. B-Splines sind wegen ihrer effizienten und stabilen Implementierung sowohl bei der Analyse (6.2.6), als auch der Synthese (6.1.4) vor allem für hochdimensionale Interpolationen sehr interessant¹⁷.

¹⁷<https://developer.nvidia.com/gpugems/gpugems2/part-iii-high-quality-rendering/chapter-20-fast-third-order-texture-filtering>

Akronyme

- ACF** Auto-Correlation Function. 81, 82, 86
- ADC** Analog-to-Digital Converter. 6
- AUT** Antenna Under Test. iii, 78, 79
- BIBO** Bounded-Input-Bounded-Output. 28, 34, 61
- CMOS** Complementary Metal Oxide Semiconductor. 6
- DFT** Discrete Fourier Transform. ii, iv, 57, 61, 62, 64–67, 69–72, 74–76, 80, 82, 87
- DTFT** Discrete Time Fourier Transform. 16, 17, 58, 59, 61, 62, 64, 93
- EADF** Effective Aperture Distribution Function. iii, 80
- FFT** Fast Fourier Transform. 66, 76, 80
- FIR** Finite Impulse Response. ii, iii, 34, 35, 37, 38, 66
- FLOP** Floating Point Operation. 66, 82
- FT** Fourier Transform. 15–17, 50, 53, 54, 71
- iDFT** Inverse Discrete Fourier Transform. 64, 65
- iDTFT** Inverse Discrete Time Fourier Transform. 59
- IIR** Infinite Impulse Response. ii, iii, 34, 35, 37, 38
- LFSR** Linear Feedback Shift Register. iv, 84–87
- LTI** Linear Time-Invariant. i, ii, 5, 30–32, 34, 39, 42, 43, 61, 65, 77
- MIMO** Multiple Input Multiple Output. iii, 76
- MLBS** Maximum Length Binary Sequence. ii, iv, 85, 87
- ROC** Region of Convergence. iii, 39–41, 44, 45, 61
- RX** receiver. 81
- STFT** Short Time Fourier Transformation. ii, iv, 68–74
- TX** transmitter. 81

Literatur

- [1] J. Proakis und D. Manolakis. *Digital Signal Processing*. Pearson Deutschland, 2013. URL: <https://elibrary.pearson.de/book/99.150005/9781292038162> (siehe S. 7, 13, 18, 21, 30, 37, 40, 42, 44, 65, 84, 93).
- [2] T. E. Oliphant. *A guide to NumPy*. Trelgol Publishing USA, 2006 (siehe S. 21).
- [3] M. Frigo und S. Johnson. „The Design and Implementation of FFTW3“. In: *Proc. IEEE* 2 (2005). Special issue on “Program Generation, Optimization, and Platform Adaptation”. DOI: [10.1109/jproc.2004.840301](https://doi.org/10.1109/jproc.2004.840301). URL: <https://doi.org/10.1109/jproc.2004.840301> (siehe S. 76, 80).
- [4] G. Del Galdo Prof. Dr.-Ing. „Geometry-Based Channel Modeling for Multi-User MIMO Systems and Applications“. en. Diss. 2007. URL: https://www.db-thueringen.de/receive/dbt_mods_00011002 (siehe S. 76–78).
- [5] A. Richter. „Estimation of Radio Channel Parameters“. In: (2005). URL: https://www.db-thueringen.de/receive/dbt_mods_00004815 (siehe S. 76).
- [6] M. Haardt, R. Thomä und A. Richter. „Multidimensional High-Resolution Parameter Estimation with Applications to Channel Sounding“. English. In: *High-Resolution and Robust Signal Processing*. Hrsg. von Y. Hua. 2003 (siehe S. 76).
- [7] S. Semper, M. Döbereiner, C. Steinmetz, M. Landmann und R. Thomä. „High Resolution Parameter Estimation for Wideband Radio Channel Sounding“. In: *IEEE Transactions on Antennas and Propagation* (2023). DOI: [10.1109/TAP.2023.3286024](https://doi.org/10.1109/TAP.2023.3286024) (siehe S. 76).
- [8] M. Landmann und G. D. Galdo. „Efficient antenna description for MIMO channel modelling and estimation“. In: *7th European Conference on Wireless Technology, 2004*. 2004 (siehe S. 78–80).
- [9] M. Unser. „Splines: a perfect fit for signal and image processing“. In: *IEEE Signal Processing Magazine* 6 (1999). DOI: [10.1109/79.799930](https://doi.org/10.1109/79.799930) (siehe S. 88, 89).
- [10] I. J. Schoenberg. „Contributions to the Problem of Approximation of Equidistant Data by Analytic Functions“. In: *I. J. Schoenberg Selected Papers*. Hrsg. von C. de Boor. Birkhäuser Boston, 1988. DOI: [10.1007/978-1-4899-0433-1_1](https://doi.org/10.1007/978-1-4899-0433-1_1). URL: https://doi.org/10.1007/978-1-4899-0433-1_1 (siehe S. 88).