← LAB ◊ ↑ TOC ↑

Gruppe P
Alsheikh, Roubeer
Nouri, Nawid
Nguyen, Hoang Son

****

Inhalt

0. Allgemeine Hinweise

0.1 Vor dem Versuch

Bitte bereiten Sie die wie folgt gekennzeichneten Blöcke *vor* dem Praktikumsversuch vor, tragen Sie Ihren Namen und die Ihrer Gruppe oben und die Antworten ins Notebook ein und speichern Sie es lokal unter dem Namen "Lab1_Nachname_Vorname" (File -> Save Notebook As ...) ab.

VORBEREITUNG:

Beispiel

- **Geben Sie die Systemfunktion eines Differenzierers an**

$H(z) = 1 - z^{-1}$

- **Welche Filter sind für die Praxis am wichtigsten?**

Kaffeefilter und *Spamfilter*

Simulationsaufgaben werden gemeinsam im Praktikum bearbeitet, sie sind gekennzeichnet durch

SIMULATION:

0.2 Tipps zum Schreiben und Coden

- **\<SHIFT>-\<RETURN> führt eine Codezelle aus und rendert eine Textzelle.**
- **In Markdown sind Leerzeilen wichtig zum Trennen von Abschnitten!**
- **Sie können LaTeX-Code zwischen $ ... $ einschließen.**
- **Kontexthilfe zu Funktionen etc. bekommen Sie über \<SHIFT>-\<TAB>**

0.3 Nach dem Versuch

**Nach dem Praktikumsversuch exportieren Sie das Notebook mit Textantworten, Codezellen und Plots als HTML (File -> Export Notebook As ... -> Export Notebook to HTML) und reichen es in Moodle ein. **

Abbildungen in diesem Notebook wurden konvertiert mit https://www.base64-image.de/ und in den HTML-Code eingebettet.

In [ ]:
```
import os, sys
module_path = os.path.abspath(os.path.join('..')) # append directory one level up to import path
if module_path not in sys.path:  # ... if it hasn't been appended already
    sys.path.append(module_path)
import dsp_fpga_lib as dsp
dsp.versions() # print versions

%matplotlib inline
import matplotlib.pyplot as plt
size = {"figsize":(12,6)} # Plotgröße in Inch

import numpy as np
import scipy.signal as sig
import wave

from IPython.display import Audio, display
```

```
Python version: 3.10.9
Numpy: 1.23.5
Scipy: 1.10.0
Matplotlib: 3.7.0 module://matplotlib_inline.backend_inline
```

LAB 1: Intro, Faltung zeitdiskreter Signale und LTI-Systeme

1. Einführung

In diesem Abschnitt erstellen Sie verschiedene Signale, plotten sie und machen sie hörbar.

Zunächst generieren wir ein Sinussignal der Länge $T = 2$ s und der Frequenz $f_a = 440$ Hz, das mit $f_S = 8$ kHz abgetastet wird.

Bitte bereiten Sie die wie folgt gekennzeichneten Blöcke vor dem Praktikum vor:

VORBEREITUNG:

- **Wie viele Datenpunkte N und wie viele Perioden L umfasst das Signal?** $L = \ldots$, $N = \ldots$

  $L = T_S \cdot f_a = 2\text{s} \cdot 440\text{kHz} = 880$

  $N = T_S \cdot f_s = 2\text{s} \cdot 8\text{kHz} = 16000$

- **Wie groß ist die Abtastperiode T_S?**

  $T_s = \frac{1}{f_s} = \frac{1}{8\text{kHz}} = 125\mu\text{s}$

1.1 Signalerzeugung

SIMULATION:

Achtung: Bei der Angabe von Bereichen ist in Python das letzte Element immer ausgeschlossen - `a = np.arange(10)` erzeugt ein Array mit den Zahlen 0 ... 9; `a[3:7] = [3, 4, 5, 6]`, Element Nr. 7 ist gerade eben nicht mehr dabei. Im Gegensatz zu Matlab nutzt Python 0-based indexing, d.h. das erste Element eines Arrays ist `a[0]`.

- **Erzeugen Sie ein Index-Array `n` mit N ganzzahligen Elementen mit der numpy Funktion `n = np.arange(end)` Vollständige Syntax: `n = np.arange(start, end, step)` mit den Defaultwerten `start=0` und `step=1` und `end=N`.**
- **Erzeugen Sie ein Zeit-Array `t` mit N reellwertigen Elementen im Abstand T_S über `t = np.linspace(start, end, N, endpoint=False)` (hier wäre `end` defaultmäßig das letzte Element des Arrays, das daher mit `endpoint=False` ausgeschlossen werden muss). Prinzipiell geht das auch mit `t = np.arange(0, N * T_S, T_S)`, hier besteht aber die Gefahr dass das Array wegen Rundungsfehlern ein Element mehr enthält als erwartet.**
- **Stellen Sie mit `len()` sicher, dass die Längen der beiden Vektoren identisch sind, schauen Sie sich mit `n[-1]` und `t[-1]` jeweils den Wert des letzten Arrayelements an.**
- **Stellen Sie die Funktionen `y1 = sin( ... Fa n)` und `y2 = sin( ... fa t)` so auf, dass mit $F_a = f_a/f_S$ beide Ausdrücke die gleichen Zahlenfolgen ergeben. Was liefert der Vergleich `print(y1[:10] == y2[:10])`? Erklären Sie das Ergebnis!**
- **Das Herausschneiden von Bereichen aus einem Array nennt man "Slicing", siehe z.B. https://www.w3schools.com/python/numpy_array_slicing.asp . Mit Slicing kann man ziemlich kranke Dinge tun, `y[2 : -1 : -2]` gibt Ihnen z.B. rückwärts jeden zweiten Wert ("-2") eines Arrays aus, angefangen beim letzten Wert ("-1") bis zum zweiten Wert ("2"), der gerade eben nicht mehr mit dabei ist. Was liefert `a[:10]`? Und was könnte `a[10:]` bedeuten?**

In [ ]:

```
N = 200
T_S = 125E-6
print(T_S)
n = np.arange(0,N,1)
print(N*T_S)
t = np.linspace(0, N * T_S, N, endpoint=False)
print(len(n), len(t))
#print(n)
#print(t)
f = 440
y1 = np.sin(2 * np.pi * f * n * T_S)
y2 = np.sin(2 * np.pi * f * t)
print(np.allclose(y1[:10], y2[:10]))
print(y1[:10])
print(y2[:10])
```

```
0.000125
0.025
200 200
True
[0.         0.33873792 0.63742399 0.86074203 0.98228725 0.98768834
 0.87630668 0.66131187 0.36812455 0.03141076]
[0.         0.33873792 0.63742399 0.86074203 0.98228725 0.98768834
 0.87630668 0.66131187 0.36812455 0.03141076]
```

1.2 Anschauen

In der nächsten Codezelle sollen Sie die Daten plotten mit Hilfe der Funktionen `ax.plot(x,y)` oder `ax.stem(x,y)`. In das Signal "hinein zoomen" können Sie mit `ax.set_xlim([start, stop])`, dabei beziehen sich `start` und `stop` auf die Werte des Zeitvektors, nicht auf die Indizes. Mit `ax.plot(x[10:20], y[10:20])` wählen Sie indexbasiert Slices aus, die fürs Plotten gleiche Länge haben müssen. Sie können die Plots verschönern mit `ax.set_xlabel("mein Label")` (ebenso für die y-Achse) oder `ax.set_title("mein Titel")`.

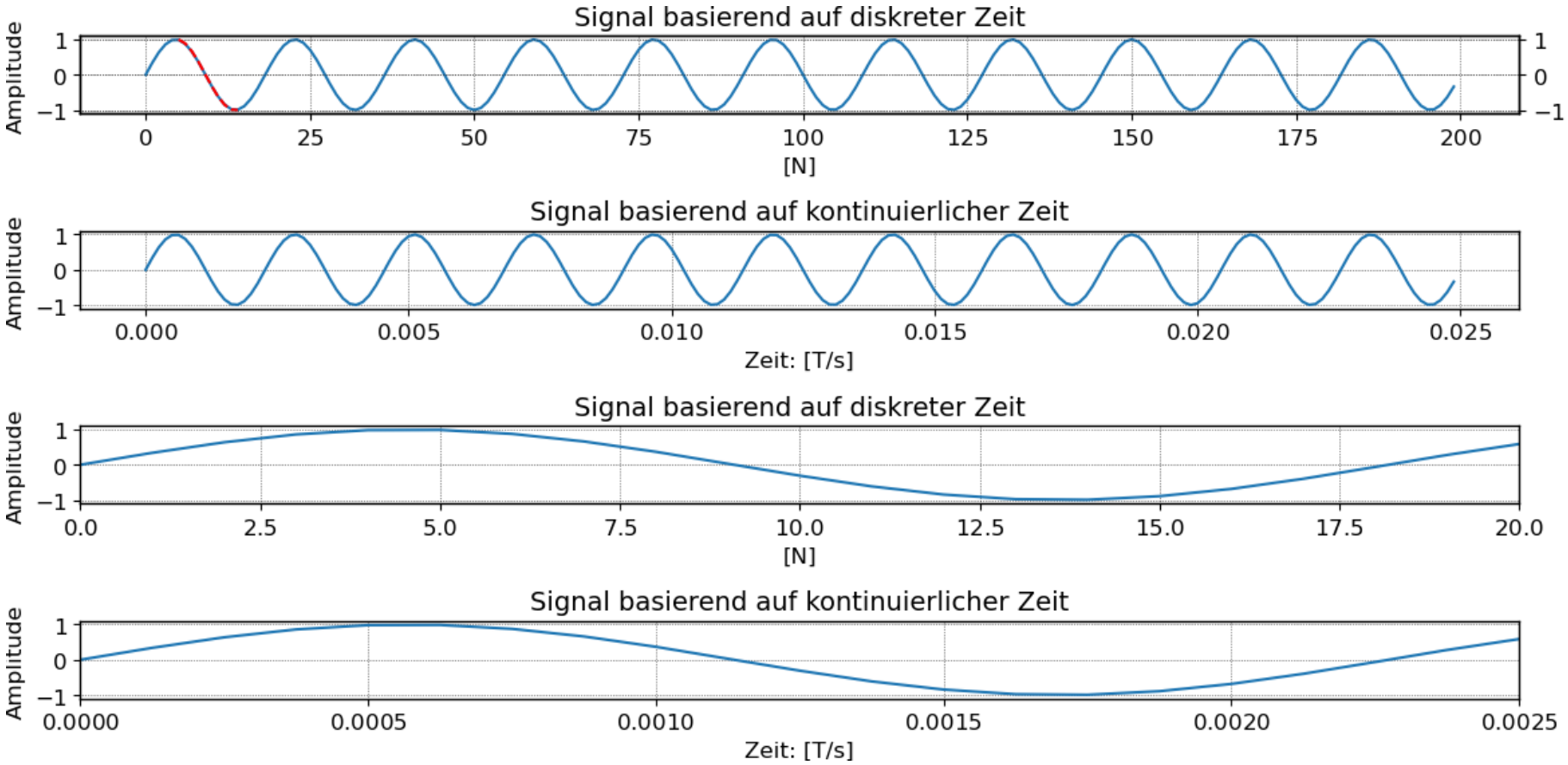
Die etwas seltsam aussehende Syntax `**size` bewirkt übrigens, dass das Dictionary `size` entpackt wird und die einzelnen Schlüssel-Werte Paare an die subplots-Methode übergeben werden.

SIMULATION:

- **Plotten Sie jetzt die Daten, indem Sie den Code der nächsten Zelle vervollständigen.**
- **Fügen Sie Label, Titel oder was immer Ihnen gefällt hinzu**
- **Testen Sie `ax.set_xlim()`, um in die Signale hinein zu zoomen.**
- **Zeigen Sie, dass die Signale identisch sind, indem Sie die gleichen Indizes beider Signale über einen kleinen Bereich darstellen.**

In [ ]:

```
#fig, axs = plt.subplots(2,1,**size)
#ax1.plot(n,...)
#ax2.plot(t,...)

fig, (ax1,ax2,ax3,ax4) = plt.subplots(4, 1, **size)

ax1.plot(n, y1, label='Signal basierend auf diskreter Zeit')
ax1.set_title('Signal basierend auf diskreter Zeit')
ax1.set_xlabel('[N]')
ax1.set_ylabel('Amplitude')
#ax1.legend()

ax2.plot(t, y2, label='Signal basierend auf kontinuierlicher Zeit')
ax2.set_title('Signal basierend auf kontinuierlicher Zeit')
ax2.set_xlabel('Zeit: [T/s]')
ax2.set_ylabel('Amplitude')
#ax2.legend()

ax3.plot(n, y1, label='Signal basierend auf diskreter Zeit')
ax3.set_xlim(n[0],n[20])
ax3.set_title('Signal basierend auf diskreter Zeit')
ax3.set_xlabel('[N]')
ax3.set_ylabel('Amplitude')

ax4.plot(t, y2, label='Signal basierend auf kontinuierlicher Zeit')
ax4.set_xlim([t[0], t[20]])
ax4.set_title('Signal basierend auf kontinuierlicher Zeit')
ax4.set_xlabel('Zeit: [T/s]')
ax4.set_ylabel('Amplitude')

index_i = slice(5,15)
ax1_compare = ax1.twinx() # Zweite y-Achse für den Vergleichplot
ax1_compare.plot(n[index_i], y2[index_i], linestyle='--', color = 'red')

plt.tight_layout()
plt.show()

print(n[10]*T_S,n[30]*T_S)
print([T_S*10, T_S*30])
```




```
0.00125 0.00375
[0.00125, 0.00375]
```

1.3 Anhören

Die Daten kann man sich im Browser anhören mit der `Audio` Klasse aus dem `IPython.display` - Modul:

```
display(Audio((data=None, filename=None, url=None, embed=None, rate=None, autoplay=False)
```

`data` kann dabei ein ein- oder zweidimensionales numpy-Array oder Liste sein, ein Filename oder auch eine URL. Der Parameter `rate` definiert die Abtastrate (nicht bei wav-Files, dort steht die Abtastrate im File.

In [ ]:
```
display(Audio(data=y2, rate=8000))
```

SIMULATION:

Erzeugen Sie ein Stereosignal mit Hilfe eines zweiten Sinussignals mit 445 Hz und hören Sie sich das Resultat mit dem Kopfhörer an.

Sie können zwei Listen oder Arrays zusammenfügen zu einer Liste mit `list12 = [list1, list2]` oder Sie nutzen `a12 = np.vstack((a1, a2))`. Für Stereowiedergabe muss das Array die Form `(2, Anzahl_Samples_pro_Kanal)` haben

Überprüfen Sie mit den Befehlen `np.shape(a)`, `a.size`, `len(a)` oder `np.ndim(a)` Form und Dimension des entstandenen Arrays. Was macht der Befehl `np.hstack()`?

Ein Signal transponiert man (vertauschen der Achsen) ganz einfach mit `a.T`. Falls sich ein Signal gar nicht abspielen lässt, versuchen Sie es einmal zu transponieren.

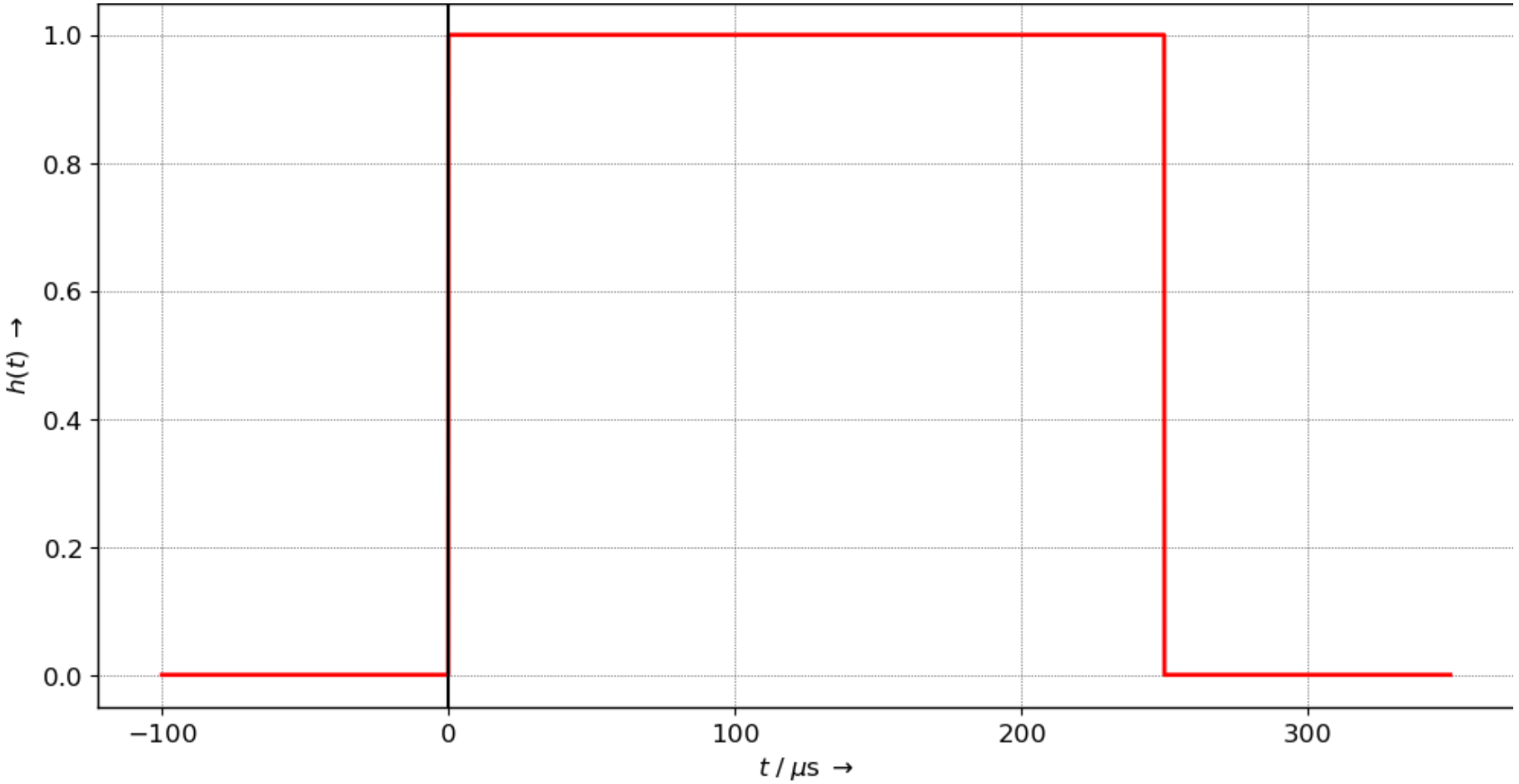
In [ ]:
```
#y3 = np.sin(2 * np.pi * 445 * t)
#y3
#display(Audio(data=y3))

y3 = np.sin(2 * np.pi * 445 * t)
signal = np.vstack((y2, y3))
display(Audio(data=signal, rate=8000))
print("Shape:", np.shape(signal))
print("Size:", signal.size)
print("Len:", len(signal))
print("Dimensions:", np.ndim(signal))
# np.hstack() fügt die Arrays horizontal zusammen
horizontal_stack = np.hstack((y2.reshape(-1, 1), y3.reshape(-1, 1)))
# nochmal Überprüfen
print("\nNach np.hstack():")
print("Shape:", np.shape(horizontal_stack))
print("Size:", horizontal_stack.size)
print("Len:", len(horizontal_stack))
print("Dimensions:", np.ndim(horizontal_stack))
```


```
Shape: (2, 200)
Size: 400
Len: 2
Dimensions: 2

Nach np.hstack():
Shape: (200, 2)
Size: 400
Len: 200
Dimensions: 2
```

2 Diskrete Faltung

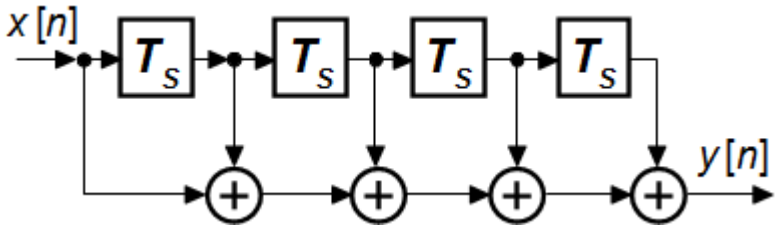
In diesem Versuchsteil entwerfen wir ein zeitdiskretes Filter mit $f_S = 64$ kHz so, dass dessen Impulsantwort $h[n]$ möglichst ähnlich zu einem analogen Rechteckpuls $h(t) = \mathrm{rect}(t/\Delta T)$ mit $\Delta T = 250\,\mu$ s wird. Die Impulsantwort ist also eine Abfolge von L Diracstößen mit gleicher Amplitude. Mit dem so definierten System filtern wir anschließend Rauschsignale.

In [ ]:
```
fig, ax = plt.subplots(**size)
t = np.arange(-100, 350, 0.1)
ax.plot(t, np.where((t >= 0) & (t < 250), 1, 0), 'r', lw=2)
ax.axvline(0, ls='-',color='k')
ax.set_xlabel(r"$t \; / \;\mu\mathrm{s} \;\rightarrow$")
ax.set_ylabel(r"$h(t) \;\rightarrow$");
```



Dieses System wird Moving Average (MA) Filter (gleitender Mittelwert) genannt, das folgende Bild zeigt den Signalflussgraphen eines MA-Filters der Ordnung $N = 4$ bzw. Länge $L = 5$. Die Länge der Impulsantwort entspricht der Anzahl der Taps.

Die Abbildung zeigt nur die prinzipielle Struktur, bitte verwenden Sie im folgenden die von Ihnen berechnete Anzahl von Taps L!



VORBEREITUNG:

- **Wieviele Taps L braucht das Filter, damit seine Impulsantwort dem zeitkontinuierlichen Puls $h(t)$ entspricht?** $L = \ldots$

  > $250/15.625 = 16$
  >
  > $N = L - 1 = 15$ **bei Moving-Average-Filter**

- **Wie groß ist die Abtastperiode T_S?**

  > $1/64$ **kHz =** 15.625μ **s**

- **Wie können Sie näherungsweise den analogen Puls $h(t)$ plotten?**

  > **Dirac Stöße nacheinnander stacken**

2.1 Einschub: Abschnittsweise definierte Funktionen in Python

Allgemein können Sie Funktionen abschnittsweise definieren mit:

2.1.1 List Comprehension

Mit einer sogenannten "List Comprehension" können Sie sehr kompakt eine Liste definieren oder modifzieren. Mit `y=[i*i for i in t]` definieren Sie eine Liste mit einer quadratischen Funktion über dem Zeitvektor t, mit `y=[i < T_0 for i in t]` eine Liste mit einer Funktion, die 1 ist für $t < T_0$ und ansonsten 0. Dabei wird das schlampige Typecasting von Python genutzt (True -> 1, False -> 0).

List comprehensions ähneln der mathematischen Schreibweise:

$$y = t^2 \text{ für } t \in \mathbb{R}$$

bzw.

$$y = 1 \text{ für } t < T_0, \text{ sonst } 0.$$

Mit einer zusätzlichen Bedingung kann man auch kompliziertere Funktionen erzeugen wie die Menge der Quadratzahlen aller geraden Zahlen zwischen 10 und 20: `m = [x**2 for x in range(10,21) if x%2 == 0]`. Mathematisch gesehen:

$$m = \{x^2 \in \mathbb{N} \,|\, 10 \leq x \leq 20,\ x \bmod 2 = 0\}$$

2.1.2. Schneller mit numpy

Für große Datenmengen und / oder hohe Geschwindigkeit sollte man numpy-Funktionen wie `np.where()` verwenden:

In [ ]:
```
t = np.arange(100) # Array 0 ... 99
print(t)
h = np.where(t < 25, 1, 0) # 1 für t < 25, ansonsten 0
print(h)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
 96 97 98 99]
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

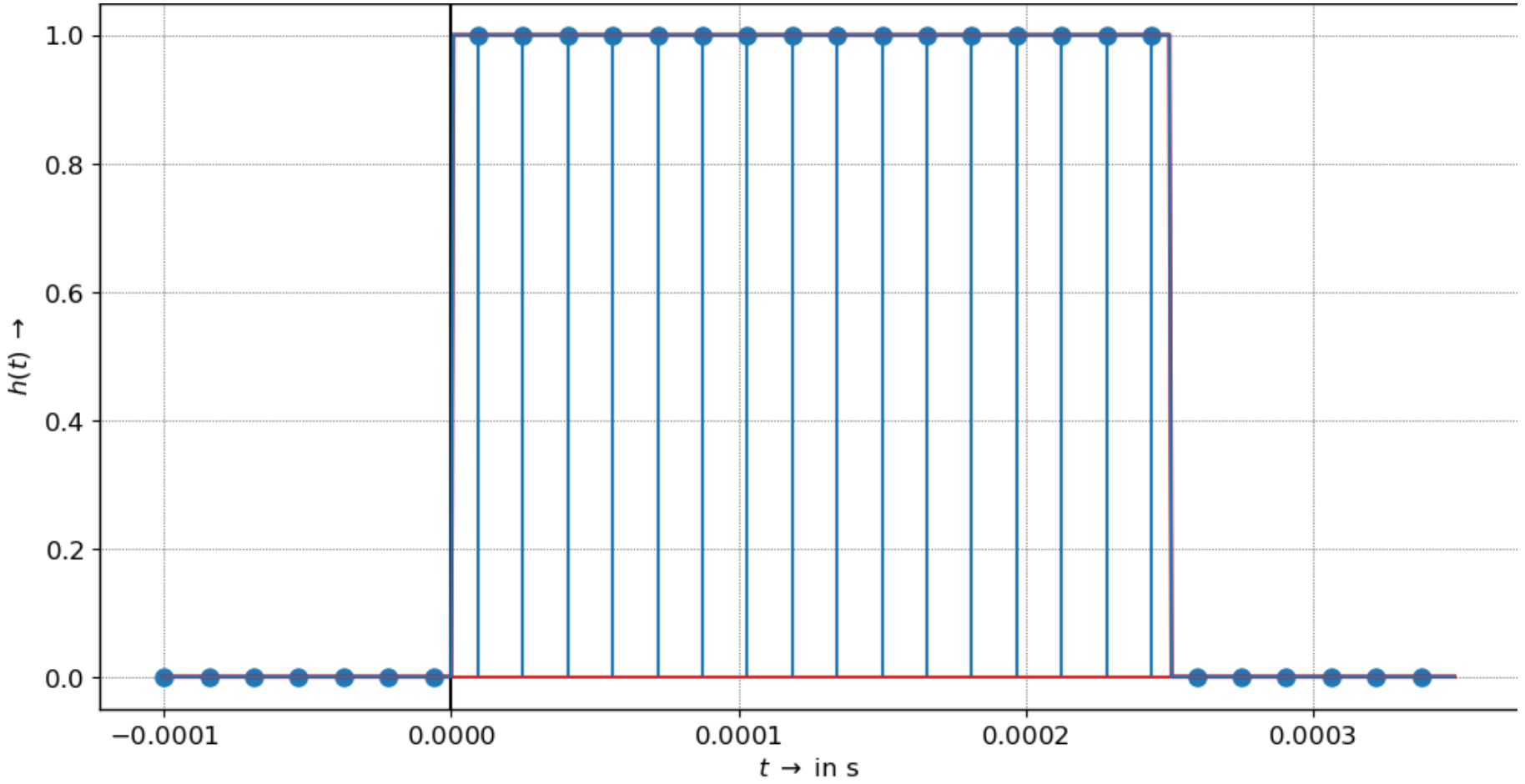
2.2 Plotten der Impulsantwort

SIMULATION:

Definieren Sie jetzt mit Hilfe einer list comprehension oder der `np.where()` Funktion die Impulsantwort des Filters und plotten Sie sie als `stem()` Plot über der Zeit t. Optional überlagern Sie im gleichen Plot den "analogen" Rechteckpuls.

Tipp: Sie können das Aussehen des Stemplots ändern z.B. mit `stem(n,y, linefmt='b-', markerfmt='ro', basefmt='k')` ('b-' blaue Linien, 'ro' rote Punkte als Marker und 'k' schwarze Grundlinie)

In [ ]:
```
fig, ax = plt.subplots(**size)
tt = np.arange(-100e-6, 350e-6, step=15.625e-6)
#print(tt)
ht = np.where( (tt >= 0) & (tt < 250e-6), 1, 0)
ax.stem(tt, ht)

t = np.arange(-100e-6, 350e-6, step=15.625e-6/20)
ax.plot(t, np.where((t >= 0) & (t < 250e-6), 1, 0), 'r', lw=2)
ax.axvline(0, ls='-',color='k')
ax.set_xlabel(r"$t \; / \;\mu\mathrm{s} \;\rightarrow$")
ax.set_ylabel(r"$h(t) \;\rightarrow$");
h = np.where( (t >= 0) & (t < 250e-6), 1, 0)
#h = np.ones(L)
#h = np.where(n < L, 1, 0)

ax.plot(t, h)
ax.set_xlabel(r"$t \; \rightarrow$ in s");
```


1.0
0.8
0.6
0.4
0.2
0.0
h(t) →
−0.0001
0.0000
0.0001
0.0002
0.0003
t → in s

2.3 Rauschen als Testsignal

Als Testsignal für das System verwenden wir gauss- oder normalverteiltes Rauschen: `x_n = np.random.randn(M,N)` **gibt ein Array** `x_n` **zurück mit den Dimensionen** $M \times N$ **und der Varianz** $\sigma^2 = 1$ **sowie dem Mittelwert** $m = 0$. M **ist bei uns die Anzahl der Audiokanäle, also 1 oder 2.**

Addieren Sie einen Gleichspannungswert von 1 V zum Rauschsignal. Durch Multiplikation mit σ^2 **können Sie die Varianz anpassen.**

SIMULATION:

Hören Sie sich zunächst das Stereorauschen in der nächsten Zelle an. Ersetzen Sie versuchsweise das normalverteilte Rauschen durch gleichverteiltes (uniform) Rauschen `np.random.rand(M,N)` **. Hören Sie einen Unterschied?**

In [ ]:
```
k = 16000

x_uniform = np.random.uniform(0.0, 1.0, (2,k))+1
x_normal = np.random.normal(2, 1, (2, k))

#print(np.median(x_uniform))
#print(np.median(x_normal))

display(Audio(data=x_uniform, rate=k))
display(Audio(data=x_normal, rate=k))

#figure, axis = plt.subplots(2)
#axis[0].plot(x_uniform)
#axis[1].plot(x_normal)
```

VORBEREITUNG:

- **Wodurch unterscheiden sich normal- und gleichverteiltes Rauschen? Finden Sie es mit einem Lehrbuch oder Wikipedia o.ä. heraus!**

  Weicher Rauschen Normalverteilt: Parametrisiert nach Mittelwert und Standardabweichung/Varianz und gewünschter Normalverteilung Härter Rauschen Gleichverteilt: Weißes Rauschen, Werte des Rauschens liegen dann gleichverteilt im Intervall [-Rauschamplitude, +Rauschamplitude]

- **Warum ändert sich der "Sound" des Rauschens im vorigen Beispiel, wenn Sie die Abtastrate in der vorigen Codezelle ändern?**

  Tonhöhe/Frequenz ändert sich. Das Signal wird auch länger je niedriger die Frequenz ist.

- **Im nächsten Abschnitt sollen Sie das Rauschsignal `x_n` der Länge `Lx` filtern, indem sie es mit dem vorher definierten Filter (Länge `L`) falten. Wie viele Samples `Ly` enthält das Ergebnis der diskreten Faltung $y[n] = x_n[n] * h[n]$?**

  $(Lx + L - 1) = Ly$

- **Welche Verstärkung $H(f = 0)$ hat unser Moving Average Filter für Gleichsignale?**

  Moving Average Filter ein Tiefpassfilter -> Verstärkung 1. Signal sollte Mittelwert durchgelassen werden.

- **Sie können ein Array `t` mit passender Länge und Skalierung zum Array `y` erzeugen mit `t = np.arange(len(y))*T_S`. Wie funktioniert dieser Befehl (Ausprobieren, keine schriftliche Antwort erforderlich)?**

  array Länge y und skaliert mit T_S

2.3 Falten und Plotten

Mit `np.convolve(a,b)` faltet man zwei eindimensionale Array miteiander., ggf. können Sie ein Array der Form (500,1) mit `np.squeeze(x_n)` in eine "flache Form" bringen oder aus einem mehrdimensionalen Array mit `x_n[:,0]` eine Dimension herausziehen.

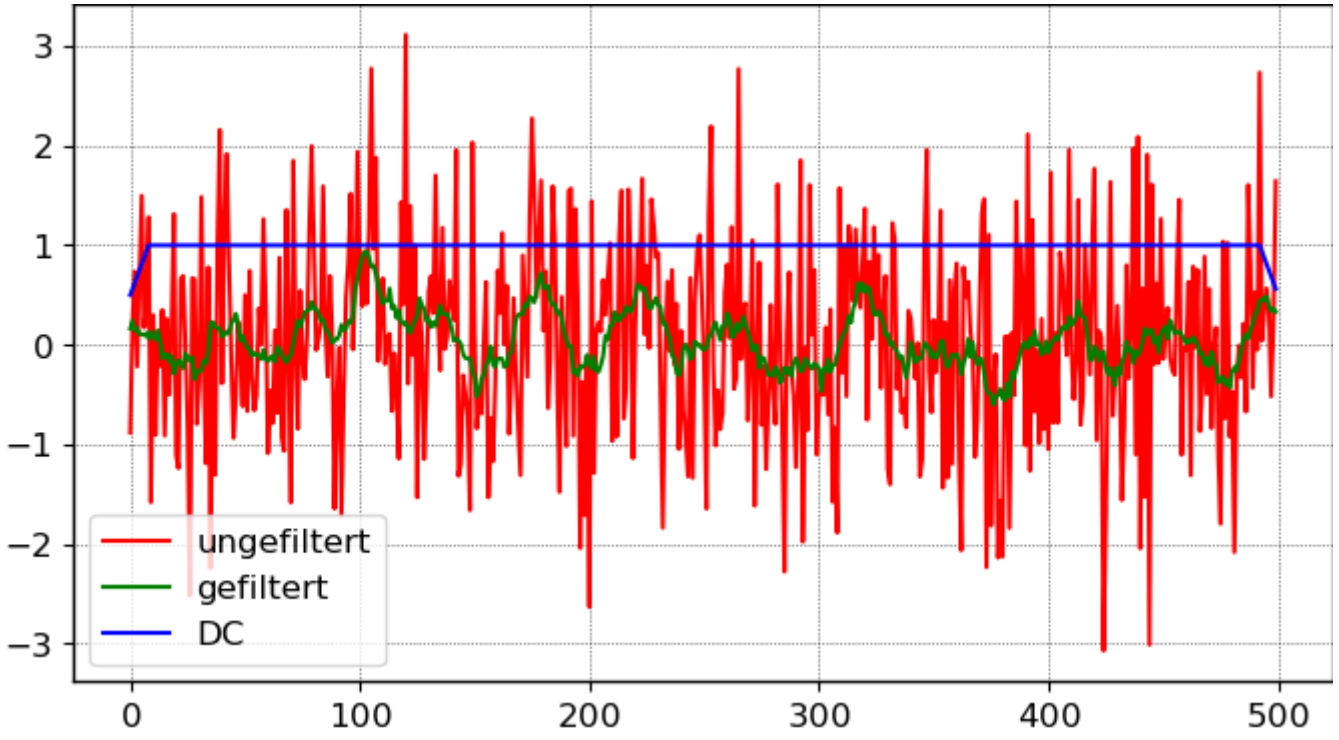
SIMULATION:

- **Definieren Sie jetzt die Impulsantwort des MA-Filters ohne zusätzliche Nullen mit `h = np.ones(L)` und falten Sie das Rauschen mit der Impulsantwort per `y=np.convolve(x_n, h)`. Stimmt die Länge des resultierenden Arrays `y` mit Ihrer Berechnung aus der Vorbereitung überein?**
- **Stellen Sie ungefiltertes und gefiltertes Rauschen im gleichen Plotfenster dar. Skalieren Sie die Zeitachse in Sekunden.**
- **Passen Sie das Filter so an, dass die DC-Verstärkung 1 ist und vergleichen Sie ungefiltertes und gefiltertes Rauschen.**
- **Welche Art von Filterung (HP, TP, ...) hat das Moving Average Filter? Erhöhen Sie die Länge des Rauschsignals und hören Sie es sich an (Sie müssen das Signal mit `y.T` transponieren - ich habe auch noch nicht verstanden warum).**

In [ ]:

```
#Filter erstellen
L = 16
x_n = np.squeeze(np.random.randn(500,1))
t = np.arange(len(x_n))
h = np.ones(L) # MA-filter mit Länge L - setzen Sie den passenden Zahlenwert ein
h = h / L

#Signal filtern
y = np.convolve(np.squeeze(x_n), h, mode = 'same')
y2 = np.convolve(np.ones(len(x_n)), h, mode = 'same')

display(Audio(data=y.T, rate=16000))

#Shape and print
print("shape x_n: ", np.shape(x_n), "shape y: ", np.shape(y))

#figure, axis = plt.subplots(3)
#figure.set_dpi(160)
#axis[0].plot(x_n)
#axis[0].set_title("Original")
#axis[1].plot(y)
#axis[1].plot(y2)
#axis[1].set_title("Filtered")
#axis[2].plot(y2)
#axis[2].set_title("Filtered Normalized")

#Plotten
plt.plot(t, x_n, color='r', label='ungefiltert')
plt.plot(t, y, color='g', label='gefiltert')
plt.plot(t, y2, color='b', label='DC')

plt.legend()
plt.tight_layout()
plt.show()
```


```
shape x_n:  (500,) shape y:  (500,)
```

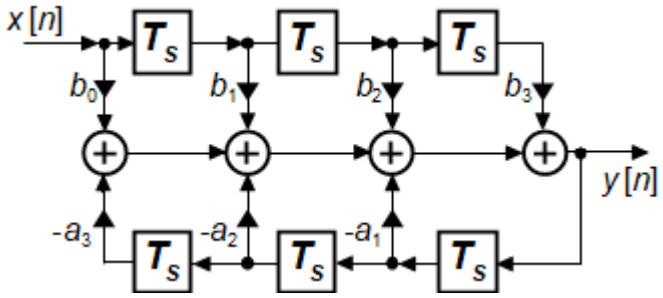


3. Transferfunktion von LTI-Systemen

Zeitdiskrete Systeme werden in Python und Matlab meistens über die Koeffizienten a_i des rekursiven Teils und die Koeffizienten b_i des nicht-rekursiven Teils angegeben:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}}{1 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}}$$

Aus den Koeffizienten erhält man nämlich sofort den Signalflussgraph in der sog. "Direktform":


x [n]
T_S
T_S
T_S
b_0
b_1
b_2
b_3
y [n]
-a_3
-a_2
-a_1
T_S
T_S
T_S

Sie können mit der Software pyfda die Eigenschaften des Systems visualisieren und eigene Filter entwerfen. Im Tab "b,a" importieren Sie Koeffizienten aus einem CSV-File oder (nach Auswahl von "Clipboard" in den Einstellungen) direkt aus der Zwischenablage. Die Koeffizienten `b` des nicht-rekursiven Teils geben Sie hierfür getrennt durch Kommata an, optional in einer zweiten Zeile die Koeffizienten `a` des rekursiven Teils.

Alternativ arbeiten Sie mit dem Notebook LTF-Filter_properties.ipynb, das Ihnen ebenfalls die wichtigsten Eigensschaften darstellt. Aus diesem Notebook können Sie auch leicht Code zum Plotten verschiedener Filtereigenschaften kopieren.

3.1 Analyse des Moving Average Filters aus Abschnitt 2

VORBEREITUNG:

- **Geben Sie nicht-rekursiven Koeffizienten des Filters an. Vergleichen Sie dazu obige Abbildung der Direktform mit der Abbildung des Moving Average Filters.**

  > $b_n = [1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1]$

- **Wie lauten die rekursiven Koeffizienten?**

  > $a_0 = 1$ **Der Rest ist 0**

- **Begründen Sie mit Hilfe der Systemfunktion $H(z)$ ob das Filter linearphasig ist.**

  > **symmetrische Impulsantwort = linearphasig bei FIR Filtern**
  >
  > **P/N Diagramm. Falls Pole außerhalb des Ursprungs dann nicht linearphasig**

- **Welche Ordnung N hat es?**

  > $N = 15$

- **Welche Gruppenlaufzeit τ_g hat es?**

  > **Mitte des Filters =>** $\tau_g = \frac{N}{2} \cdot T_s = 15/2 \cdot T_s = 7.5 * T_s$

> **Mitte des Filters ->** $\tau_g = \frac{N}{2} \cdot T_S = 15/2 \cdot T_S = 7.5 * T_S$

- **Berechnen Sie die Frequenzen, bei denen der Betragsgang Null wird (siehe z.B. Folie LTF-54 oder Musterlösung zu Aufgabe 2.11).**
- **Wie müsste $h[n]$ aussehen, damit die Gruppenlaufzeit $\tau_g = 0$ wäre?**

SIMULATION:

- **Plotten Sie Betrags- und Phasenfrequenzgang, Gruppenlaufzeit und P/N-Diagramm.**
- **Lesen Sie $H(f = 0)$ aus der Simulation ab und vergleichen Sie mit Ihrem berechneten Wert.**

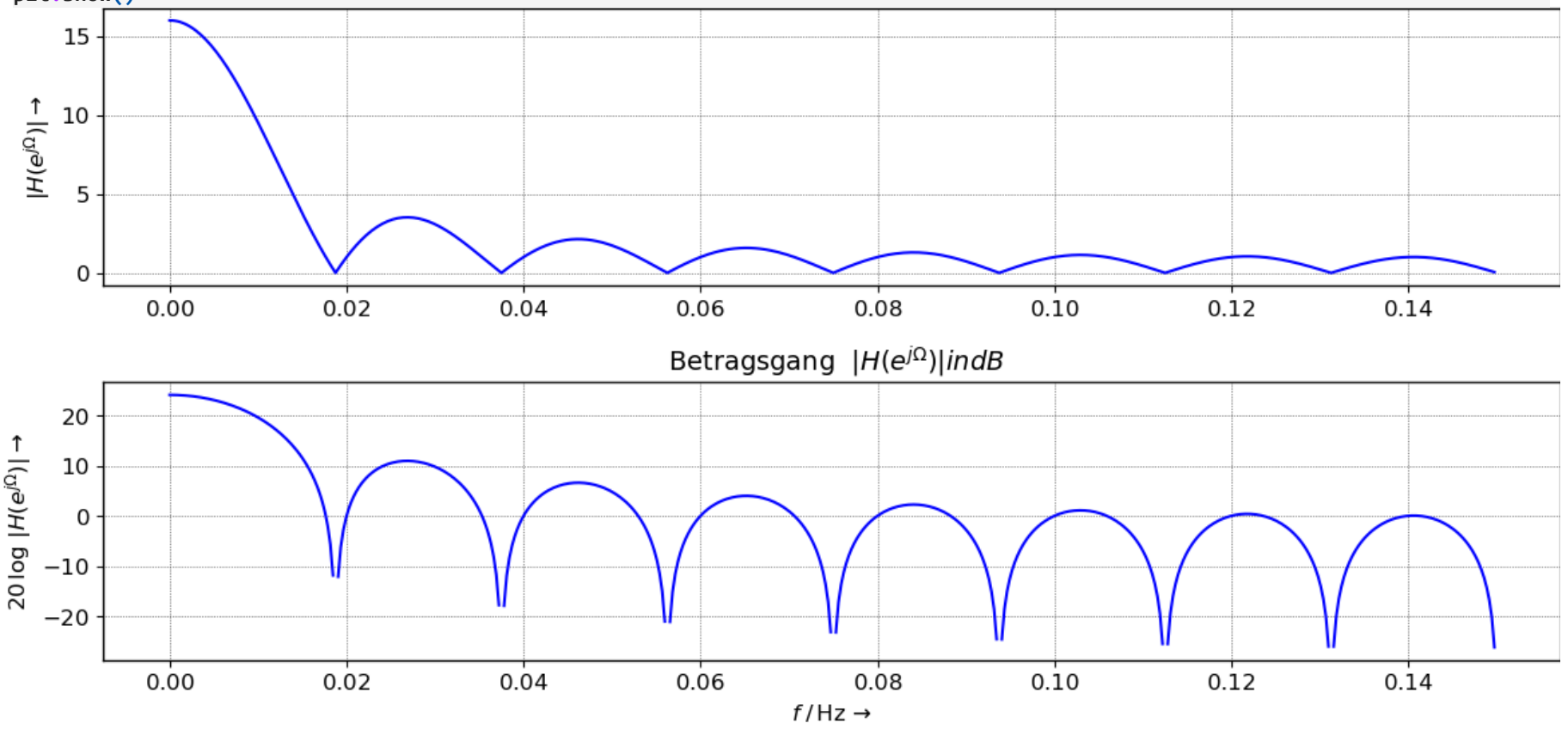
> **Simulation: Dämpfung vin H(f=0) = 0dB -> DC-Verstärkung von 1**
>
> **Berechnung: Für z = 0 -> H_z(z=0) = 0.06251 + 0.0625z^-1 + 0.0625z^-2 .... + 0.0625z^-15 = 1 -> Gleiche DC Verstärkung**

- **Stimmt die simulierte Gruppenlaufzeit mit der berechneten überein?**

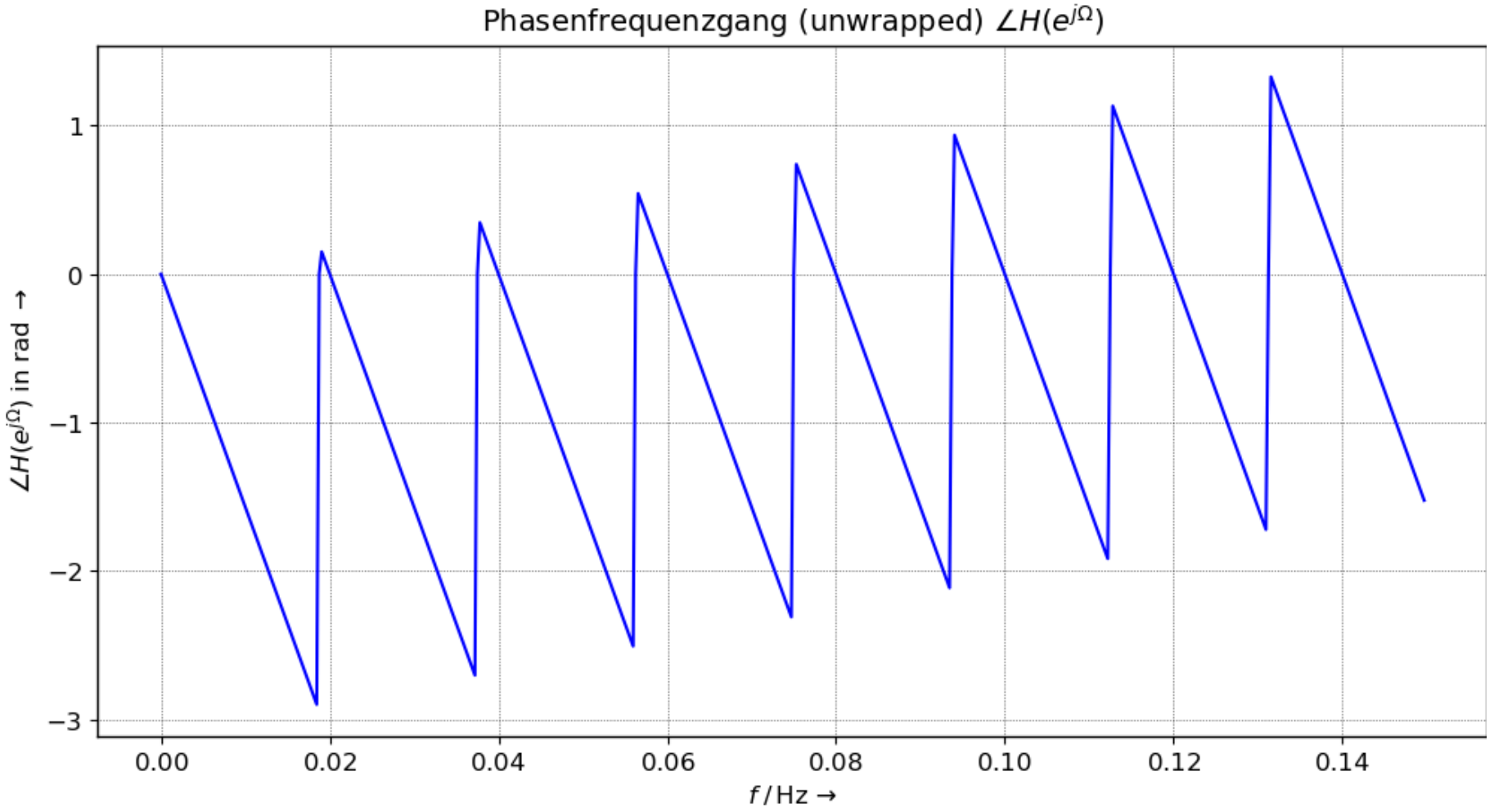
> **Gruppenlaufzeit liegt ca. 117ms**

In [ ]:
```
b = np.ones(16); a = 1
fig, [ax1,ax2] = plt.subplots(2,**size)
f_P = 0.2 # Eckfrequenz Passband bezogen auf f_S/2
f_S = 0.3 # Eckfrequenz Stopband bezogen auf f_S/2
Apass = 1 # max. Ripple im Passband in dB
Astop = 40 # min.Dämpfung im Stopband in dB
fil_type='ellip'
#b,a = sig.iirdesign(f_P, f_S, Apass, Astop, ftype=fil_type)
[W, H] = sig.freqz(b, a, whole=False);
f = f_S * W  / (2 * np.pi)
ax1.plot(f, (np.abs(H)), 'b')
ax1.set_ylabel(r'$|H(e^{j \Omega})| \rightarrow$')
ax1.set_title(r'Betragsfrequenzgang $|H(e^{j \Omega})|$')
#
ax2.set_title(r' Betragsgang $\ |H(e^{j \Omega})| in dB$')
ax2.plot(f, 20*np.log10(np.abs(H)), 'b')
ax2.set_ylabel(r'$20 \,\log \,|H(e^{j \Omega})| \rightarrow$')
```

```
ax2.set_xlabel(r'$f \; \mathrm{/\,Hz} \, \rightarrow$');
#
plt.tight_layout()
plt.show()
```

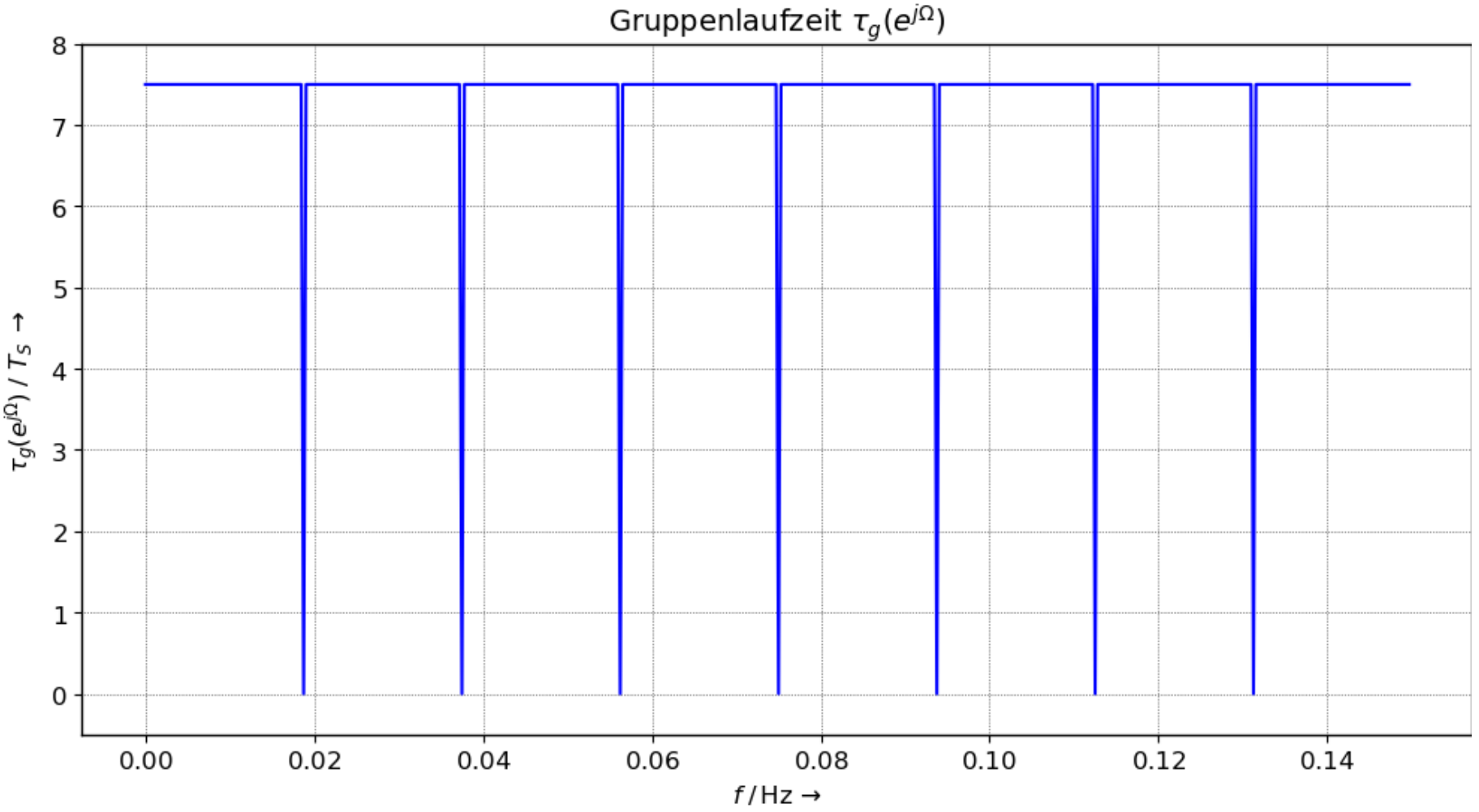


In [ ]:
```
fig, ax3 = plt.subplots(**size)
#[W, H] = sig.freqz(b, a, whole=False);
#f = f_S * W  / (2 * np.pi)
ax3.plot(f,np.unwrap(np.angle(H)), 'b')
ax3.set_ylabel(r'$\angle H(e^{j \Omega}) \mathrm{\;in\;rad\;} \rightarrow$')
ax3.set_xlabel(r'$f \; \mathrm{/\,Hz} \, \rightarrow$')
ax3.set_title(r'Phasenfrequenzgang (unwrapped) $\angle H(e^{j \Omega})$');
```

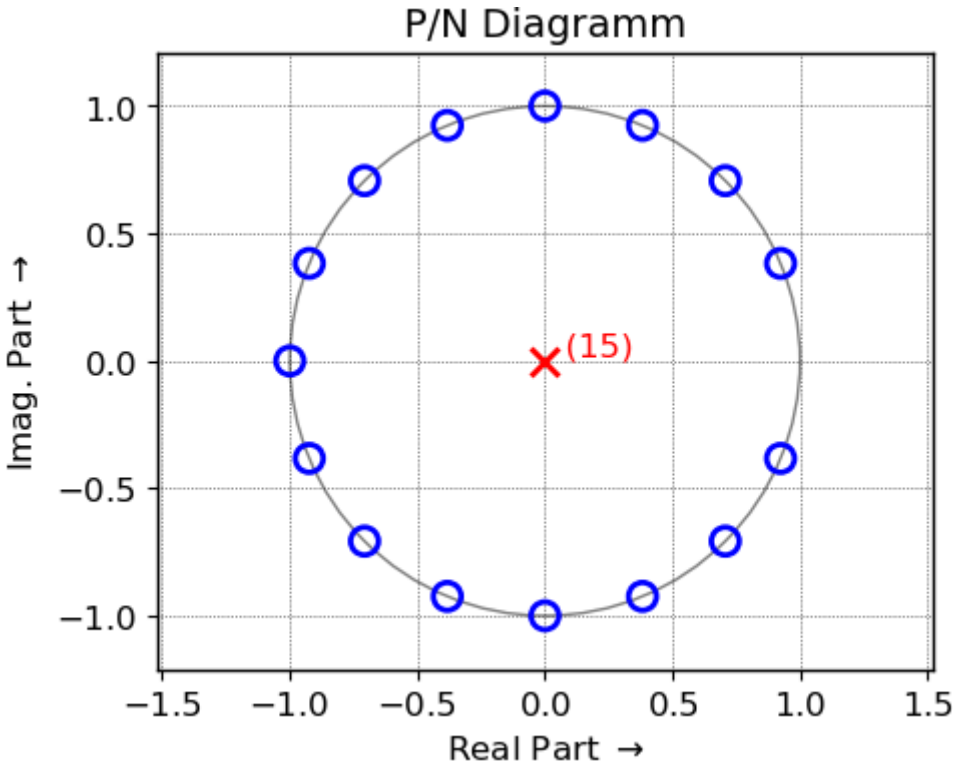


In [ ]:
```
fig, ax = plt.subplots(**size)
W, tau_g = dsp.group_delay(b, a)
f = f_S * W  / (2 * np.pi)
ax.plot(f, tau_g, 'b')
ax.set_ylabel(r'$\tau_g(e^{j \Omega}) \;/\; T_S \;\rightarrow$')
ax.set_xlabel(r'$f \; \mathrm{/\,Hz} \, \rightarrow$')
ax.set_ylim([min(tau_g)-0.5, max(tau_g)+0.5])
ax.set_title(r'Gruppenlaufzeit $\tau_g(e^{j \Omega})$');
```

```
FIR filter, using J.O. Smith's algorithm for group delay.
singularity -> setting to 0 at:
i = 0.39269908169872414
i = 0.7853981633974483
i = 1.1780972450961724
i = 1.5707963267948966
i = 1.9634954084936207
i = 2.356194490192345
i = 2.748893571891069
i = 3.141592653589793
i = 3.5342917352885173
i = 3.9269908169872414
i = 4.319689898685965
i = 4.71238898038469
i = 5.105088062083414
i = 5.497787143782138
i = 5.890486225480862
```



In [ ]:
```
fig3, ax3 = plt.subplots(figsize=(5,4))
ax3.set_xlabel(r'Real Part $\rightarrow$');
ax3.set_ylabel(r'Imag. Part $\rightarrow$')
ax3.set_title('P/N Diagramm')
dsp.zplane(b,a,plt_ax=ax3);
```



Copyright