← LAB ◊ ↑ TOC ↑

Name, Vorname:

LAB 2: Filterung und DFT als Näherung für spektrale Analyse

Inhalt

0. Allgemeines

1. FIR und IIR Filter

2. Schätzung der Fourierreihe durch FFT

3. Schätzung des Fourierspektrums von Sinustönen mit überlagertem Rauschen durch FFT

4. Fourierspektrum eines Rechteckpulses

0. Allgemeine Hinweise

- \<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>

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.

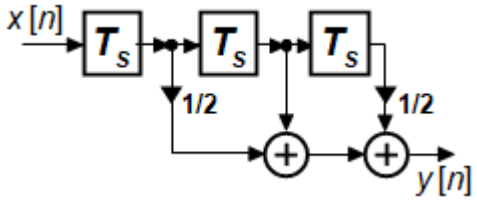
In [ ]:

```
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":(12,5)} # Plotgröße in Inch

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

from IPython.display import Audio, display
#-------------------------------------------------------------------------
def wav2np(filename):
    """ Read the wav-file and convert it to a one or two-dimensional numpy array,
        depending on the number of channels.
        Properties of the WAV-file are stored as function attributes (evil)
    """
    wf = wave.open(filename,'rb')
    wav2np.N_CH = wf.getnchannels() # number of channels
    wav2np.W    = wf.getsampwidth() # wordlength per sample in bytes
    wav2np.N_FR = wf.getnframes()   # number of frames
    wav2np.f_S  = wf.getframerate() # sample (frame) rate

    print("{0} channels with {1} frames of {2} bytes and f_S = {3} Hz.".format(wav2np.N_CH, wav2np.N_FR, wav2np.W, wav2np.f_S)

    if wav2np.W == 2:
        samples_in = np.frombuffer(wf.readframes(-1), dtype=np.int16) # read wav data as 16 bit integers, R and L samples are
    elif wav2np.W == 1:
        samples_in = np.frombuffer(wf.readframes(-1), dtype=np.int8) # read wav data as 8 bit integers, R and L samples are in
    else:
         raise TypeError("Unknown data format: {0} bytes".format(wav2np.W))

    samples = np.array([samples_in[idx::wav2np.N_CH] for idx in range(wav2np.N_CH)], dtype=np.int32) # deinterleave channels t
    return samples
```

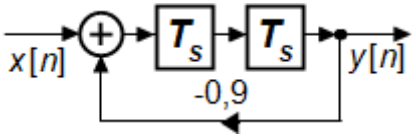
```
Python version: 3.10.9
Numpy: 1.23.5
Scipy: 1.10.0
Matplotlib: 3.7.0 module://matplotlib_inline.backend_inline
```

1. FIR und IIR Filter

In diesem Abschnitt analysieren Sie die beiden Filter aus der folgenden Abbildung:



Filter 1



Filter 2

VORBEREITUNG:

- Woran erkennen Sie, welches davon ein IIR- und welches ein FIR Filter ist?

  > Filter 1 hat keine Rückkopplung FIR
  >
  > Filter 2 hat Rückkopplung IIR

- Berechnen Sie die Differenzengleichung der beiden Filter.

  > Filter 1 $y[n] = 0.5 \cdot x[n-1] + x[n-2] + 0.5 \cdot x[n-3]$
  >
  > Filter 2 $y[n] = 0.9 \cdot y[n-2] + x[n-2]$

- Berechnen Sie die Systemantwort der beiden Filter. Können Sie die Impulsantwort beider Filter einfach aufschreiben?

  Systemfunktionen

  > Filter 1 $H(z) = \frac{Y(z)}{X(z)} = 0.5z^{-1} + z^{-2} + 0.5z^{-3} = \frac{0z^3+0.5z^2+z+0.5}{z^3} = 0.5 \cdot \frac{(z+1)^2}{z^3}$
  >
  > Filter 2 $H(z) = \frac{z^{-2}}{1+0.9z^{-2}} = \frac{1}{z^2+0.9}$

  Impulsantwort

  > Filter 1 $h[n] = \{0; 0.5; 1; 0.5\}$
  >
  > Filter 2 $h[n] = \delta[n-2] - 0.9h[n-2]$

- Berechnen Sie Pole und Nullstellen beider Systemfunktionen und skizzieren Sie (auf einem Blatt Papier) den P/N - Plan.

  > Berechnung über Mitternachtsformel FIR: Alle 3 Pole im Ursprung & 2 Nst. bei (-1)
  >
  > IIR: Keine Nst. , komplex konjugierte Pole bei $\pm 0.949i$

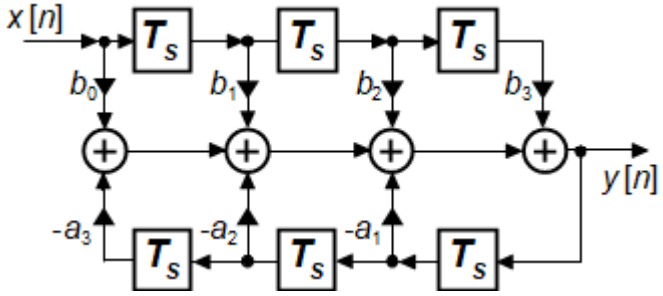
- Schätzen Sie aus dem P/N Plan ab, um welchen Filtertyp es sich handelt. Bei welcher Frequenz liegt das Maximum des Betragsgangs? Welchen Wert hat es?

Welchen Wert hat es?

> Tiefpass bei FIR
>
> Bandpass bei IIR

1.1 Simulation

Für die Simulation müssen wir zuerst die Filter als Koeffizientenvektoren `a` und `b` darstellen (siehe folgendes Bild und Abschnitt 3 von LAB 1).



Als "Codesteinbruch" können Sie wieder das Notebook 02_LTF/LTF-Filter_properties.ipynb verwenden.

Alternativ verwenden Sie pyfda: 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.

SIMULATION:

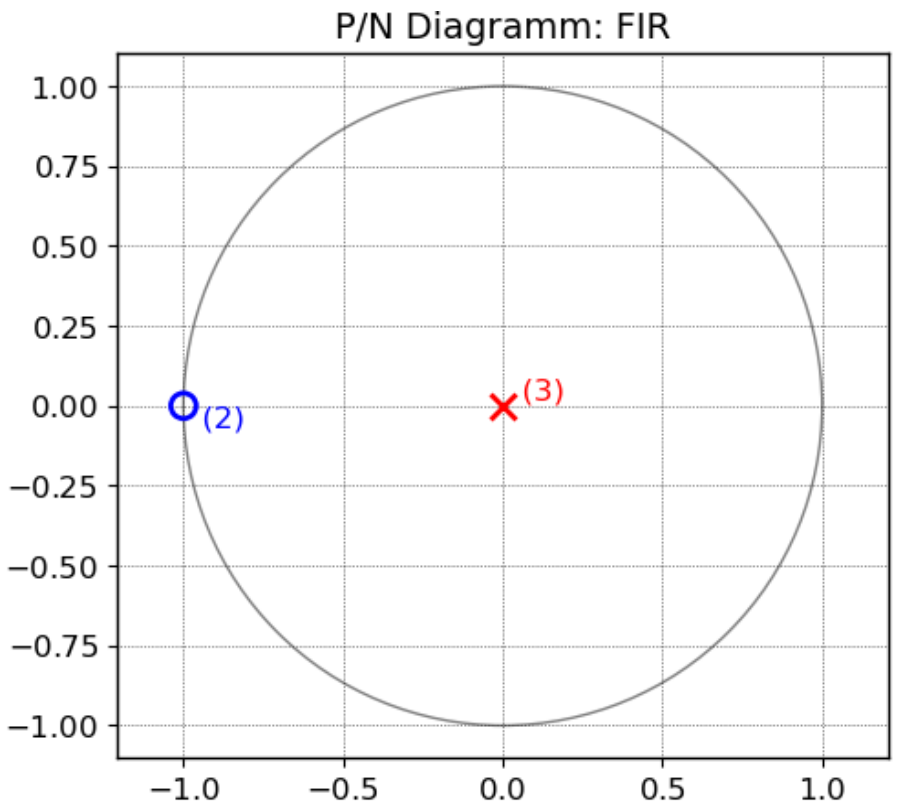
- Stellen Sie Filter 1 und Filter 2 (s.o.) als Koeffizientenvektoren `a` und `b` dar. Informationen dazu finden Sie im Abschnitt 3 von LAB 1 und in obiger Abbildung. Beachten Sie:
  - Die Reihenfolge ist $a = [1, a_1, a_2, \dots]$ und $b = [b_0, b_1, \dots]$. Warum ist immer $a_0 = 1$? Warum dreht sich das Vorzeichen der rekursiven Koeffizenten herum?
  - "Fehlende" Koeffizienten müssen als 0 eingetragen werden.
  - Die imaginäre Einheit ist `1j`, dementsprechend werden imaginäre Zahlen z.B. als `0.3j` repräsentiert.
- Testen Sie mit dem P/N Diagramm, ob Rechnung und Simulation zusammen passen.
- Lassen Sie sich die Impulsantwort anzeigen, für das FIR-Filter ist das relativ einfach ;-), beim IIR-Filter benötigen Sie `dsp.impz()`
- Plotten Sie Betragsfrequenzgang, Phasengang und Gruppenlaufzeit.

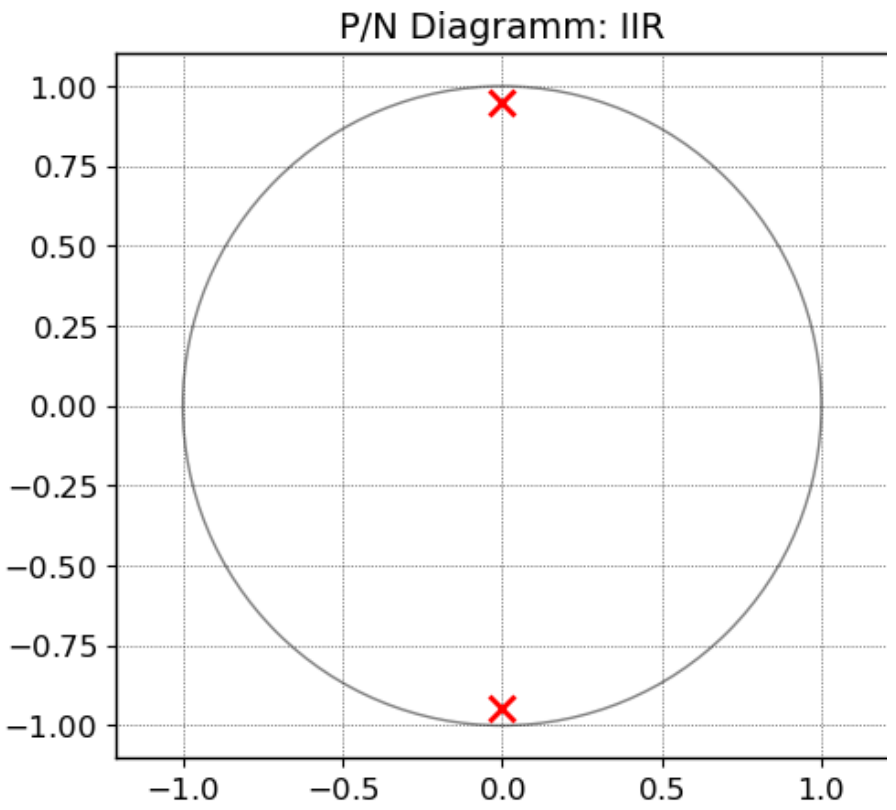
1.1.1 P/N Diagramm

In [ ]:

```
a_FIR = [1,0,0,0]
b_FIR = [0,0.5,1,0.5]
#'a_FIR = [2,0,0,0]
#b_FIR = [0,1,2,1]
a_IIR = [1,0,0.9]
b_IIR = [1]
#---------------
fig, (ax1,ax2) = plt.subplots(nrows=1, ncols=2,**size)
ax1.set_title('P/N Diagramm: FIR')
dsp.zplane(b_FIR,a_FIR, plt_ax=ax1);
print("Nullstellen FIR: {0}".format(np.roots(b_FIR)))

ax2.set_title('P/N Diagramm: IIR')
dsp.zplane(b_IIR,a_IIR, plt_ax=ax2);
print("Nullstellen IIR: {0}".format(np.roots(b_IIR)))
if type(a_IIR) in {list, np.ndarray} and len(a_IIR) > 1:
    print("Polstellen IIR:  {0}\n".format(np.roots(a_IIR)))
```

```
Nullstellen FIR: [-1. -1.]
Nullstellen IIR: []
Polstellen IIR:  [-0.+0.9486833j  0.-0.9486833j]
```



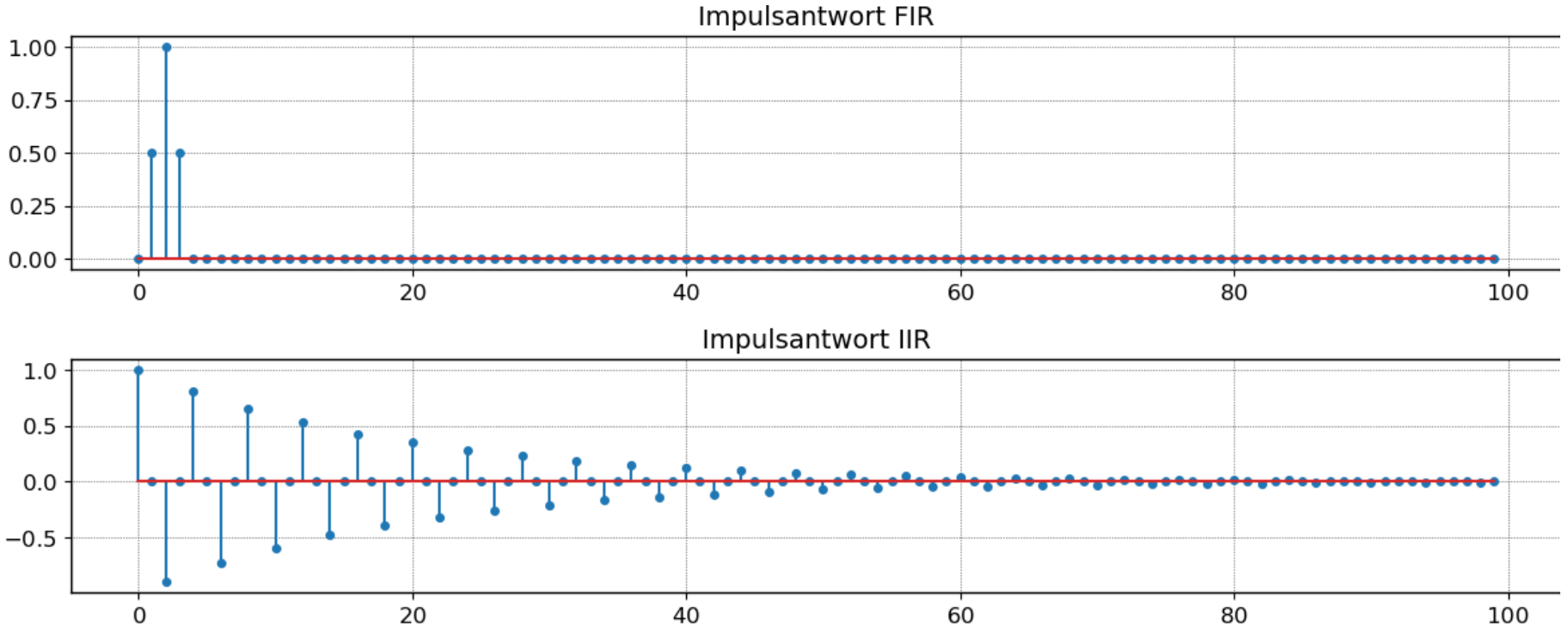


1.1.2 Impulsantwort

Der Befehl `h,t = dsp.impz(b,a)` berechnet die Impulsantwort, die Sie dann plotten können, am Besten als stem-Plot.

Warum ist beim IIR-Filter jeder zweite Impuls Null?

In [ ]:
```
fig, (ax1,ax2) = plt.subplots(nrows=2, ncols=1,**size)
ax1.set_title('Impulsantwort FIR')
h_FIR,t_FIR = dsp.impz(b_FIR,a_FIR);
ax1.stem(t_FIR, h_FIR, markerfmt='.')
ax2.set_title('Impulsantwort IIR')
h_IIR,t_IIR = dsp.impz(b_IIR,a_IIR);
ax2.stem(t_IIR, h_IIR, markerfmt='.')
fig.set_tight_layout(True)
```



1.1.3 Betrags- und Phasengang sowie Gruppenlaufzeit

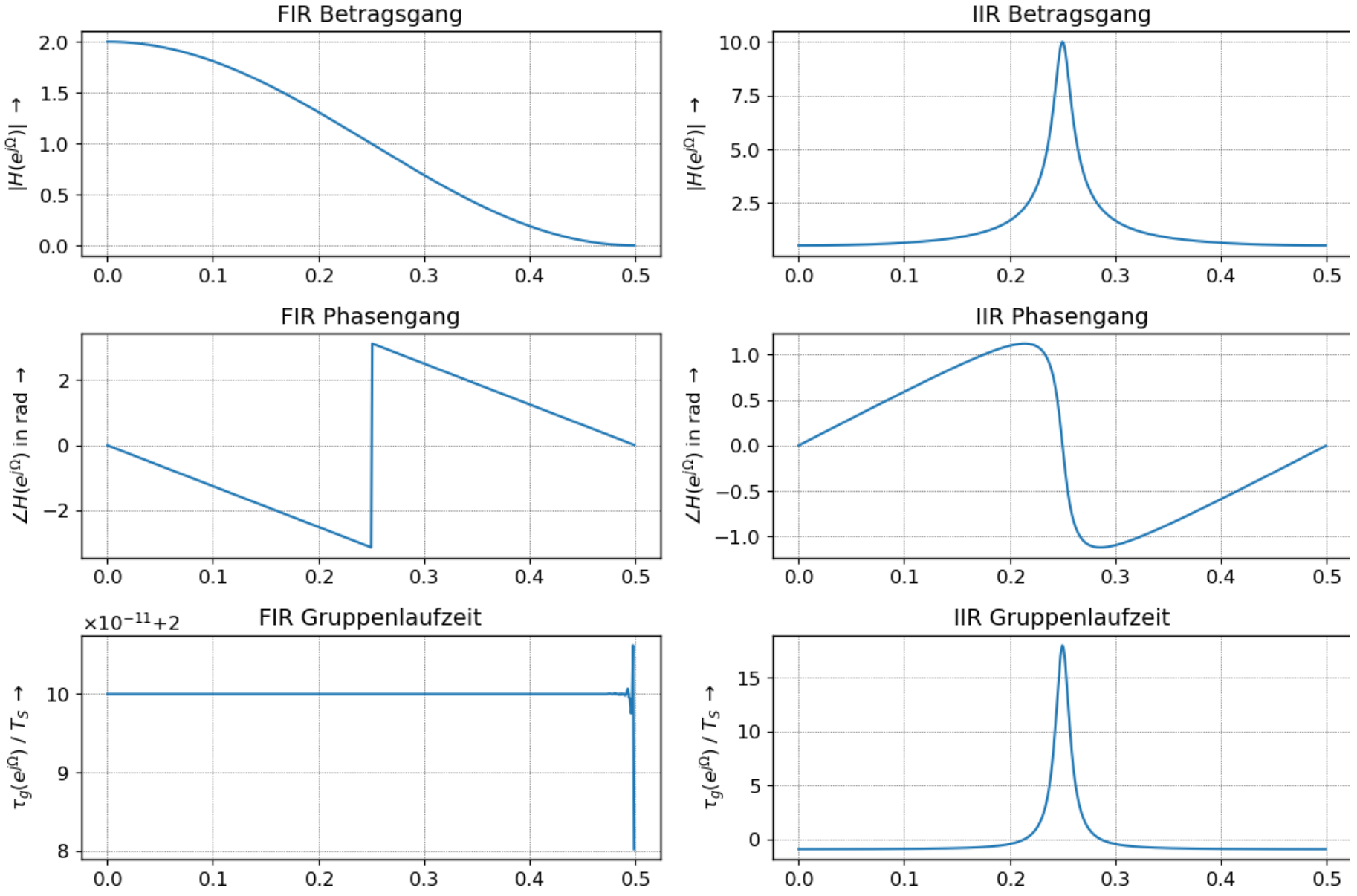
Aus dem komplexen Frequenzgang `omega, H = sig.freqz(b,a)` ermitteln Sie mit `np.abs()` und `np.angle()` Betrags- und Phasengang. Defaultmäßig werden 512 Frequenzpunkte zwischen 0 und $f_S/2$ bestimmt und als normierte Kreisfrequenz zurückgegeben.

Die Gruppenlaufzeit ermitteln Sie mit `w, H = sig.group_delay((b,a), omega)`. Mit `omega` übergeben Sie die Frequenzen, an denen die Gruppenlaufzeit berechnet werden soll (z.B. die, die Sie aus der Berechnung des komplexen Frequenzgangs erhalten haben).

Der Plot der Gruppenlaufzeit sieht u.U. etwas seltsam aus, mit `set_ylim([y_min, y_max])` passen Sie die Grenzen an.

In [ ]:

```
fig, ax = plt.subplots(nrows=3, ncols=2, figsize=(12,8))
w_FIR, H_FIR = sig.freqz(b_FIR, a_FIR)
_, tau_FIR = sig.group_delay((b_FIR, a_FIR), w_FIR)
w_IIR, H_IIR = sig.freqz(b_IIR, a_IIR)
_, tau_IIR = sig.group_delay((b_IIR, a_IIR), w_IIR)
ax[0][0].set_title("FIR Betragsgang")
ax[0][0].plot(w_FIR / (2*np.pi), np.abs(H_FIR))
ax[0][0].set_ylabel(r"$|H(e^{j \Omega})| \;\rightarrow$")
ax[1][0].set_title("FIR Phasengang")
#ax[1][0].plot(w_FIR / (2*np.pi), np.unwrap(np.angle(H_FIR)))
ax[1][0].plot(w_FIR / (2*np.pi), np.angle(H_FIR))
ax[1][0].set_ylabel(r'$\angle H(e^{j \Omega}) \mathrm{\;in\;rad\;} \rightarrow$')
ax[2][0].set_title("FIR Gruppenlaufzeit")
ax[2][0].plot(w_FIR / (2*np.pi), tau_FIR)
ax[2][0].set_ylabel(r'$\tau_g(e^{j \Omega}) \;/\; T_S \;\rightarrow$')
#ax[2][0].set_ylim([min(tau_FIR)-0.5, max(tau_FIR)+0.5])
#...
ax[0][1].set_title("IIR Betragsgang")
ax[0][1].plot(w_IIR / (2*np.pi), np.abs(H_IIR))
ax[0][1].set_ylabel(r"$|H(e^{j \Omega})| \;\rightarrow$")
ax[1][1].set_title("IIR Phasengang")
ax[1][1].plot(w_IIR / (2*np.pi), np.unwrap(np.angle(H_IIR)))
ax[1][1].set_ylabel(r'$\angle H(e^{j \Omega}) \mathrm{\;in\;rad\;} \rightarrow$')
ax[2][1].set_title("IIR Gruppenlaufzeit")
ax[2][1].plot(w_IIR / (2*np.pi), tau_IIR)
ax[2][1].set_ylabel(r'$\tau_g(e^{j \Omega}) \;/\; T_S \;\rightarrow$')
#ax[2][1].set_ylim([min(tau_IIR)-0.5, max(tau_IIR)+0.5])
#ax[2][1].set_xlabel(r"$F \;\rightarrow$")
fig.set_tight_layout(True)
```


FIR Betragsgang
IIR Betragsgang
FIR Phasengang
IIR Phasengang
FIR Gruppenlaufzeit
IIR Gruppenlaufzeit
$|H(e^{j\Omega})|$ →
$\angle H(e^{j\Omega})$ in rad →
$\tau_g(e^{j\Omega})$ / T_S →
×10^{-11}+2

1.2 Anhören

Im folgenden filtern wir Sprach- oder Rauschsignale mit unseren beiden Filtern und hören und schauen uns das Resultat an. Eine FIR-Filterung könnten Sie wieder mit `convolve(x,h)` durchführen (nur für eindimensionale Arrays). Warum funktioniert bei IIR-Filtern die `convolve` -Methode nicht?

Für IIR und FIR-Filter können Sie die Routine `y = sig.lfilter(b,a,x)` verwenden (funktioniert auch mit zweidimensionalen Arrays).

SIMULATION:

- Filtern Sie ein Rauschsignal oder einen WAV-File aus dem Unterordner `medien` mit dem FIR- und dem IIR-Filter. Rauschen erzeugen Sie wieder z.B. mit `x_n = np.random.randn(16000)` , WAV-Files wandeln Sie mit der Hilfsfunktion `wav2array(filename)` in ein ein- oder zweidimensionales Array um.
- Hören Sie sich die Wirkung der beiden Filter an mit der `Audio` Klasse aus dem `IPython.display` - Modul:

  `display(Audio((data=None, rate=None)` , `data` kann dabei ein ein- oder zweidimensionales numpy-Array oder Liste sein, ein Filename oder auch eine URL. Der Parameter `rate` definiert die Abtastrate.
- Schauen Sie sich einen Ausschnitt des ursprünglichen und des gefilterten Signals im gleichen Plot an und vergleichen Sie.

In [ ]:
```
x_n = np.random.randn(16000)
x_w = wav2np("../medien/87778__marcgascon7__vocals.wav")
#Convolve ?!?
print(np.shape(x_w))
y_FIR = sig.lfilter(b_FIR, a_FIR, x_w)
y_IIR = sig.lfilter(b_IIR, a_IIR, x_w)
print('\nOriginal')
display(Audio(data=x_w, rate=wav2np.f_S))
print('Rauschen FIR')
display(Audio(data=y_FIR, rate=wav2np.f_S))
print('Rauschen IIR')
display(Audio(data=y_IIR, rate=wav2np.f_S))
y_FIR
```

```
2 channels with 349952 frames of 2 bytes and f_S = 44100 Hz.
(2, 349952)
```

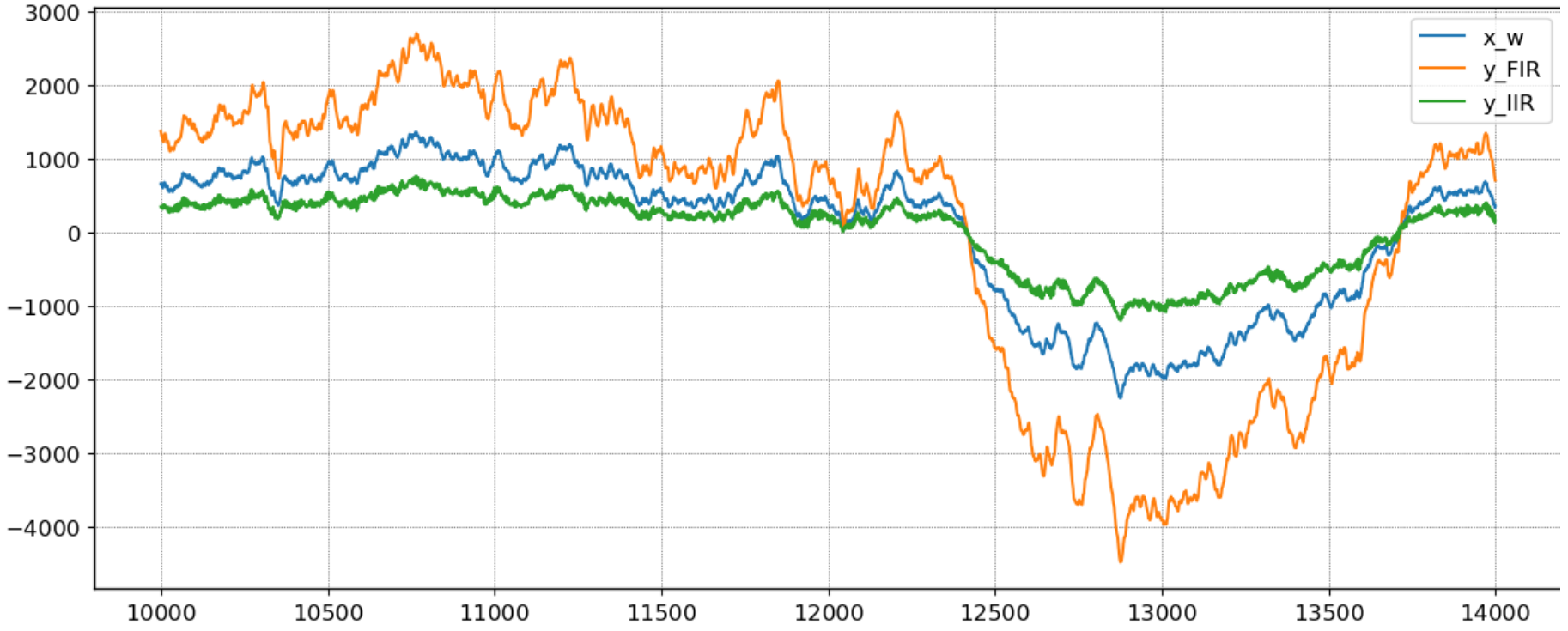
Original

Rauschen FIR

Rauschen IIR

Out[ ]:
```
array([[ 0.0000e+00, -5.0000e-01, -2.5000e+00, ..., -1.4790e+03,
        -1.5620e+03, -1.5810e+03],
       [ 0.0000e+00,  1.4000e+01,  3.9000e+01, ..., -1.1710e+03,
        -1.1850e+03, -1.1755e+03]])
```

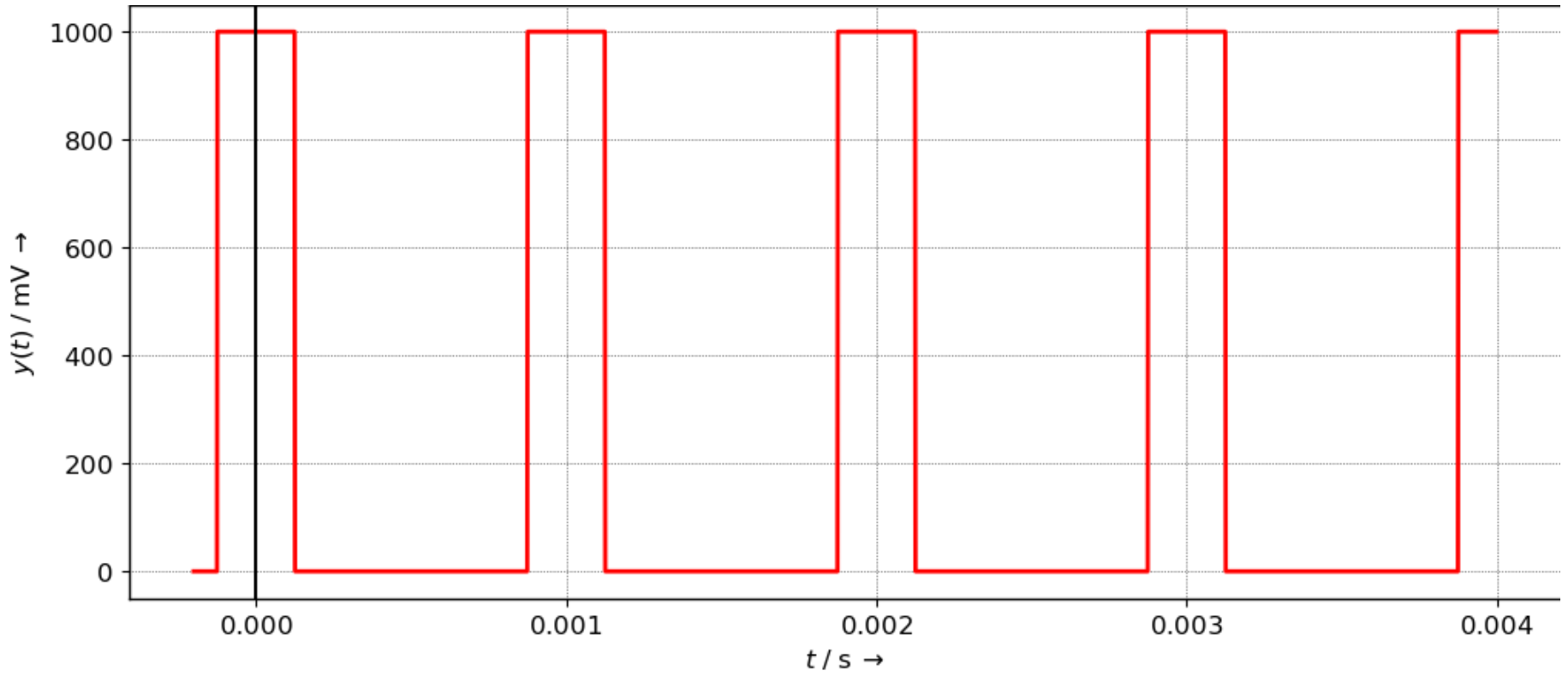
In [ ]:
```
#fig, (ax1, ax2, ax3) = plt.subplots(**size, nrows=3)
fig, ax1 = plt.subplots(**size)
l = 4000
start = 10000

t =np.arange(start, start+l, 1)

ax1.plot(t, x_w[0][t], label="x_w")
ax1.plot(t, y_FIR[0][t],label="y_FIR")
ax1.plot(t, y_IIR[0][start:start+l], label="y_IIR")
ax1.legend();
fig.set_tight_layout(True)
```



2. Fourierreihe einer Rechteckpulsfolge

In diesem Versuchsteil bestimmen wir die Fourierkoeffizienten einer rechteckförmigen zeitkontinuierlichen Pulsfolge mit $T_1 = 1$ ms. Der Duty Cycle $\alpha = 0.25$ und Amplitude $A = 1000$ mV, die Pulsbreite ist also $\Delta T = T_1/4$ (siehe nächster Plot).

In [ ]:
```
fig, ax = plt.subplots(**size)
t = np.arange(-0.2e-3, 4e-3, 1/640e3) # pseudo-analoger Zeitvektor in ms (mit 640 kHz so fein abgetastet, dass der Unterschied
y = 500*sig.square(t*1e3*2*np.pi+np.pi/4, duty=0.25) + 500
ax.plot(t, y, 'r', lw=2)
ax.axvline(0, ls='-',color='k')
ax.set_xlabel(r"$t \; / \;\mathrm{s} \;\rightarrow$")
ax.set_ylabel(r"$y(t)\; / \;\mathrm{mV} \;\rightarrow$");
```


1000
800
600
400
200
0
y(t) / mV →
0.000
0.001
0.002
0.003
0.004
t / s →

2.1 Berechnung der Koeffizienten

Ein mit T_1 periodisches Signal $y(t)$ lässt sich mit Hilfe der Fourierreihe für reellwertige Signale (Index $k \in \mathbb{N}_0$) darstellen:

$$y(t) = a_0 + 2\sum_{k=1}^{\infty} a_k \cos 2\pi k f_1 t + b_k \sin 2\pi k f_1 t \; k \in \mathbb{N}$$

Zunächst berechnen wir die Koeffizienten a_k, b_k für $k = 0, 1, \ldots, 11$ mit Hilfe der Formel für die Fourierreihe von reellwertigen Zeitsignalen. Da das Signal achsensymmetrisch ist, sind die imaginärwertigen Koeffizienten $b_k = 0$.

$$a_k = \frac{1}{T_1}\int_{-T_1/2}^{T_1/2} y(t)\cos(2k\pi f_1 t)\,\mathrm{d}t = \frac{A}{T_1}\int_{-\alpha T_1/2}^{\alpha T_1/2} \cos(2k\pi f_1 t)\,\mathrm{d}t \tag{1}$$

$$= \frac{A}{T_1 \cdot 2k\pi f_1}\sin(2k\pi f_1 t)\Big|_{-\alpha T_1/2}^{\alpha T_1/2} = \frac{A}{k\pi}\sin(k\alpha\pi) = A\alpha\,\mathrm{si}(k\alpha\pi) \tag{2}$$

SIMULATION:

Schreiben Sie ein Skript, das die ersten 11 Koeffizienten der Rechteckfunktion exakt berechnet. Welchen Vorteil hat es, die `sinc(x)` Funktion zu verwenden anstelle von `sin(x)/x`? Aber Achtung: In Numpy (und in Matlab) ist die sinc-Funktion definiert als $\mathrm{sinc}(x) = sin(\pi x)/\pi x$, das π im Argument müssen Sie daher weglassen. Drucken Sie die Ergebnisse übersichtlich in eine Tabelle aus (s.u.)

2.1.1 Formatierte Ausdrucke in Python

Einen formatierten Ausdruck erhalten Sie z.B. mit `print("\n{0:7.2f} | ".format(Y[i]), end="")` (der Teil in der geschweiften Klammer wird ersetzt durch `Y[i]` und formatiert mit insgesamt 7 Stellen, 2 Nachkommastellen, keinen Zeilenumbruch). Eingebettet in eine `for` Schleife bekommen Sie so schnell eine übersichtliche Tabelle.

- **"\n"** erzeugt einen Zeilenumbruch in einem String
- `print(a, end="" )` ersetzt den Zeilenumbruch am Ende des Printbefehls durch ein anderes Zeichen (oder nichts wie hier)
- Die **str.format()** Anweisung ersetzt Ausdrücke in geschweiften Klammern im String durch die Argumente in der .format() Anweisung, versuchen Sie es selbst mit `"{0} ich heiße und bin {1} Jahre alt".format("Yoda", 877)`.
- Der Ausdruck in der geschweiften Klammer kann durch Formatierungsanweisung ergänzt werden, **{0:7.2f}** formatiert das Argument 0 als

Float mit 2 Nachkommastellen und insgesamt mindestens 7 Stellen. **{0:>7}** lässt ebenfalls Platz für mindestens 7 Stellen oder Zeichen und richtet den Ausdruck rechtsbündig aus.

Auf https://www.python-kurs.eu/python3_formatierte_ausgabe.php finden Sie eine übersichtliche Grafik hierzu.

In [ ]:
```
k=np.arange(16)
alpha = 4/16; A = 1000
Y0 =  A*alpha*np.sinc(k*alpha)
print("i = | ", end="")
for i in k:
    #Y.append()
    print("{0:>7d} | ".format(i), end="")
print("\nY0 = | ", end="")
for i in k:
    print("{0:7.2f} | ".format(Y0[i]), end="")
```

```
i = |       0 |       1 |       2 |       3 |       4 |       5 |       6 |       7 |       8 |       9 |      10 |      11 |
12 |      13 |      14 |      15 |
Y0 = |  250.00 |  225.08 |  159.15 |   75.03 |    0.00 |  -45.02 |  -53.05 |  -32.15 |   -0.00 |   25.01 |   31.83 |   20.46
|    0.00 |  -17.31 |  -22.74 |  -15.01 |
```

2.2 Abschätzung des Spektrums durch FFT

Um das Spektrum des zeitkontinuierlichen Signals mit einer FFT abschätzen zu können, muss es zunächst abgetastet werden und zwar über eine ganzzahlige Anzahl Perioden T_1 (da es ja periodisch ist), wir starten zunächst mit einer Periode, $T_{mess1} = T_1$. Für eine effiziente Berechnung sollen $N_1 = 2^4 = 16$ Samples genommen werden.

VORBEREITUNG:

- Welche Abtastfrequenz f_{S1} ist dafür erforderlich?

  > $f_{S1} = \frac{1}{T_{Mess}} \cdot N_1 = \frac{1}{1ms} \cdot 16 = 16$ kHz

- Welchen Abstand T_{S1} haben die abgetasteten Punkte? In welchem Abstand liegen die Frequenzpunkte der FFT?

  > $T_{S1} = \frac{1}{f_{S1}} = 62.5\mu s$
  >
  > $\Delta f = \frac{f_{S1}}{N_1} = \frac{16kHz}{16} = 1000$
  >
  > **SIMULATION:**

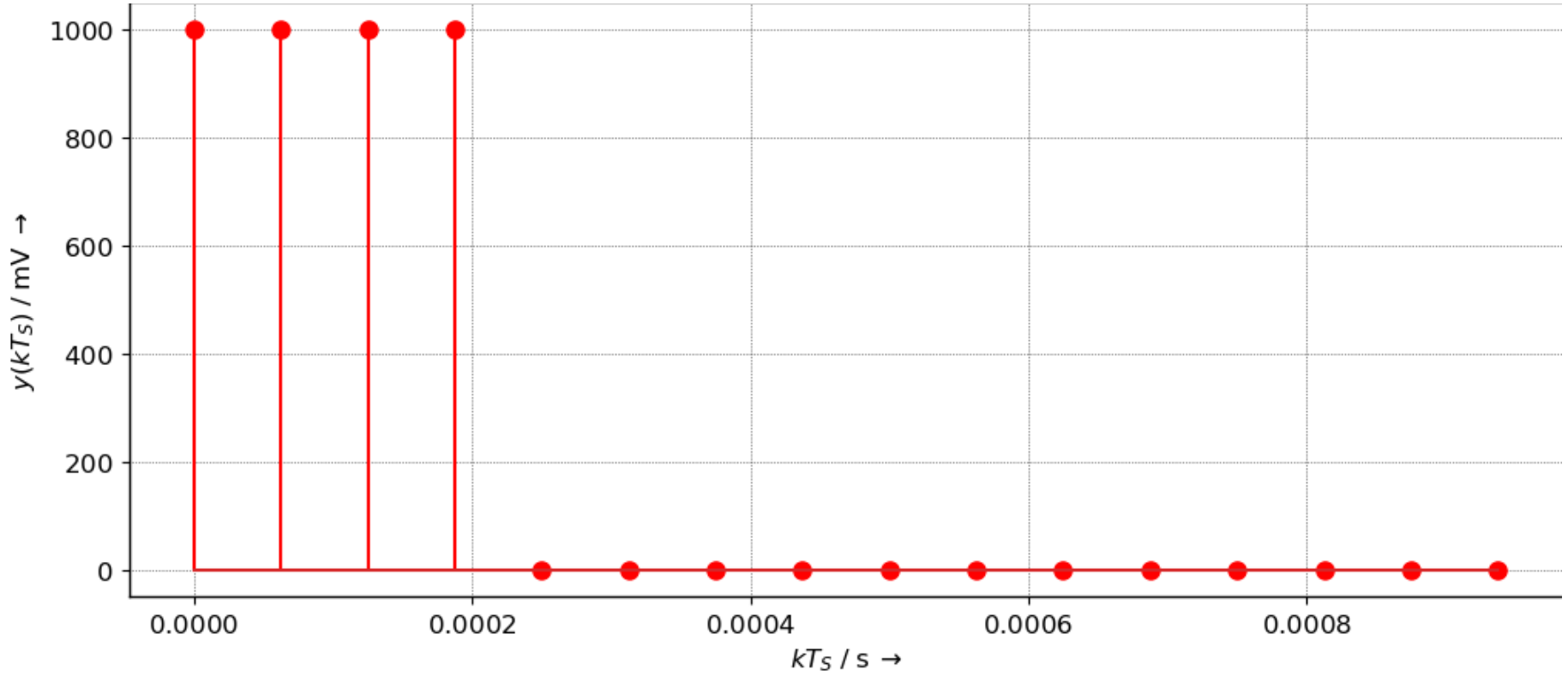
Zunächst einmal benötigen Sie einen passenden Zeitvektor z.B. mit `t1 = np.arange(N_1)/f_S1` oder durch Abtasten des "zeitkontinuierlichen" Zeitvektors `t` (s.u.). Das abgetastete Signal $y_1[n]$ erzeugen Sie:

- **Händisch** mit `y1 = np.ones(N_1); y1[4:N_1] = 0` oder durch `y1 = np.concatenate((np.ones(4), np.zeros(12)))`
- Durch **Abtasten** des "zeitkontinuierlichen" Signals ($f_S = 640$ kHz) mit `y1 = y[::40]`, mit diesem Befehl wird von y nur jedes vierzigste Element nach `y1` kopiert. Das funktioniert natürlich nur, wenn wie hier $f_{S1} = f_S/40$ ist. Wenn wie hier Start- und Endwert fehlen, werden alle Elemente von `y` vom ersten bis zum letzten berücksichtigt, Sie müssen ggf. noch Start- und Endwert anpassen, um den richtigen Ausschnitt finden.
- Durch **Berechnung aus dem Zeitvektor** mit Hilfe der `np.where()` Funktion wie im vorigen Lab.

Egal für welche Variante Sie sich entschieden haben, plotten Sie das abgetastete Signal mit dem folgenden Skript

Egal für welche Variante Sie sich entschieden haben, plotten Sie das abgetastete Signal mit dem folgenden Skript.

In [ ]:
```
fig, ax = plt.subplots(**size)
N1 = 16
f_S1 = 16e3
f_S = 64e3
t1 = np.arange(N1)/f_S1
y1 = 1000*np.ones(N1); y1[4:N1] = 0
ax.stem(t1, y1, 'r')
ax.set_xlabel(r"$k T_S \; / \;\mathrm{s} \;\rightarrow$")
ax.set_ylabel(r"$y(k T_S)\; / \;\mathrm{mV} \;\rightarrow$");
```



2.2.1 FFT, erster Ansatz: $f_{S1} = 16\ \mathrm{kHz}$, $N_1 = 16$

VORBEREITUNG:

- Welche höchste Frequenzkomponente ist im zeitkontinuierlichen Signal enthalten?

  > $\frac{f_s}{2} = \frac{640kHz}{2} = 320$ kHz

- Welche maximale Frequenzkomponente ist im abgetasteten Signal enthalten?

  > $\frac{f_{S1}}{2} = \frac{16kHz}{2} = 8$ kHz

SIMULATION:

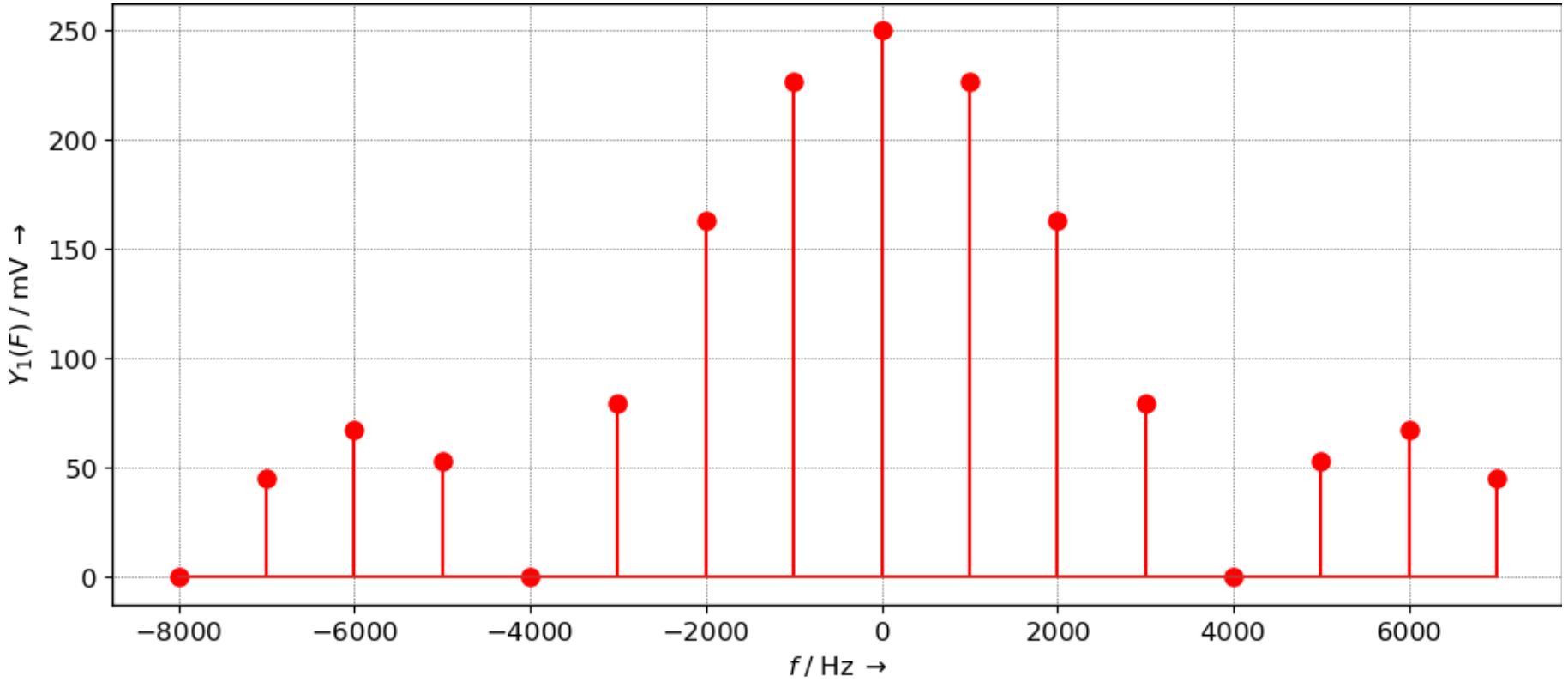
In numpy berechnen wir die FFT mit `Y = np.fft.fft(y, N)` (die FFT liefert *komplexe* Werte ...). Ohne den Parameter `N` wird die FFT über das gesamte Signal berechnet mit der Anzahl der Datenpunkte. Nutzen Sie das Notebook 03_DFT/DFT-Skalierung.ipynb zur korrekten Berechnung und Skalierung des Signals. Mit f = `np.fft.fftfreq(N, T_S)` erzeugen Sie eine passenden Frequenzvektor.

- Vervollständigen Sie das Skript in der nächsten Codezelle: Wie müssen Sie die Amplitude und die Frequenz skalieren, um physikalisch sinnvolle Werte zu erhalten?

In [ ]:

```
fig, ax = plt.subplots(**size)
N1 = 16
f_S1 = N1 * 1e3
t1 = np.linspace(0.0, N1 / f_S1, num = N1, endpoint=False)
#y1 = ...
#y1 = np.ones(N1); y1[4:N1] = 0
Y1 = np.fft.fft(y1, N1)
Y1 = np.abs(Y1)/N1
f1 = np.fft.fftfreq(N1, 1/f_S1)
ax.stem(f1, Y1, 'r')
ax.set_xlabel(r"$f \; / \;\mathrm{Hz} \;\rightarrow$")
ax.set_ylabel(r"$Y_1(F)\; / \;\mathrm{mV} \;\rightarrow$");
# ----------- Tabellenausgabe --------------
# Drucken Sie i, Y0 und Y1 aus

print("i = | ", end="")
for i in k:
    #Y.append()
    print("{0:>7d} | ".format(i), end="")
print("\nY0 = | ", end="")
for i in k:
    print("{0:7.2f} | ".format(Y0[i]), end="")
print("\nY1 = | ", end="")
for i in k:
    print("{0:7.2f} | ".format(Y1[i]), end="")
```

```
i = |       0 |       1 |       2 |       3 |       4 |       5 |       6 |       7 |       8 |       9 |      10 |      11 |
12 |      13 |      14 |      15 |
Y0 = |  250.00 |  225.08 |  159.15 |   75.03 |    0.00 |  -45.02 |  -53.05 |  -32.15 |   -0.00 |   25.01 |   31.83 |   20.46
|    0.00 |  -17.31 |  -22.74 |  -15.01 |
Y1 = |  250.00 |  226.53 |  163.32 |   79.55 |    0.00 |   53.15 |   67.65 |   45.06 |    0.00 |   45.06 |   67.65 |   53.15
|    0.00 |   79.55 |  163.32 |  226.53 |
```



Nach der SIMULATION:

- Vergleichen Sie die simulierten mit den berechneten Werten. Welche Koeffizienten weichen besonders stark von den erwarteten (berechneten) Werten ab? Was ist die Ursache dafür?
- Wie könnte man die Abweichungen reduzieren?

2.2.2 FFT mit Erhöhung der Abtastfrequenz auf $f_{S2} = 64\ \mathrm{kHz}$ und N_2 Samples

Die Abtastfrequenz wird jetzt vervierfacht auf den Wert $f_{S2} = 64\ \mathrm{kHz}$.

VORBEREITUNG:

- Wie lang muss das Messfenster T_{Mess2} der FFT sein für eine Frequenzauflösung $\Delta f = 1$ kHz? Wieviele Samples N_2 sind dafür erforderlich?

  > $T_{Mess2} = \frac{1}{\Delta f} = 1$ ms
  >
  > $N_2 = T_{Mess2} \cdot f_{S2} = 1ms \cdot 64kHz = 64$

- Welche maximale Frequenzkomponente ist im abgetasteten Signal enthalten?

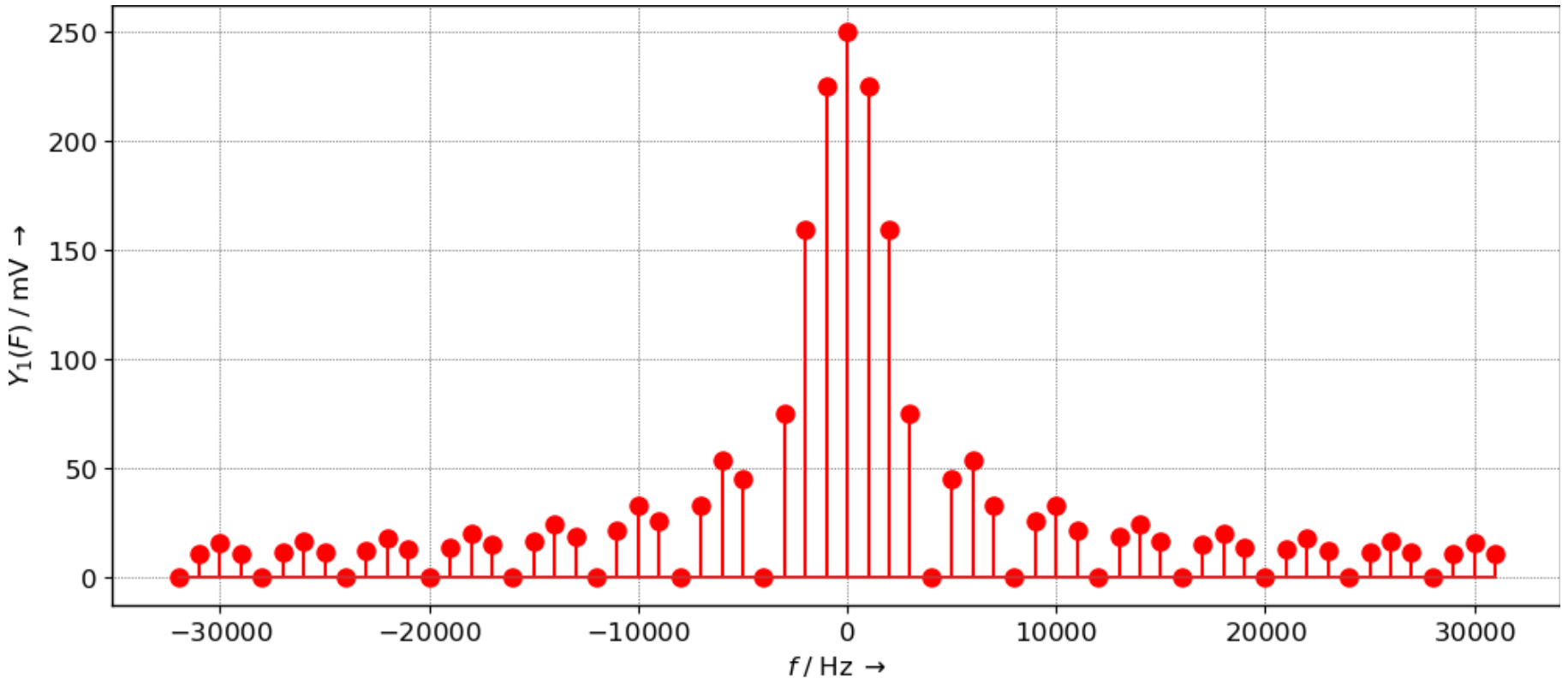
  > $f_{max} = \frac{f_{S2}}{2} = 32$ kHz
  >
  > **SIMULATION:**

- Kopieren Sie das Skript aus dem vorigen Unterpunkt und passsen Sie es an die ermittelten Parameter an.
- Vergleichen Sie erneut Rechnung mit Simulation

In [ ]:

```
fig, ax4 = plt.subplots(**size)
N2 = 64
f_S2 = N2 * 1e3
t2 = np.arange(N2)/f_S2
y2 = 1000* np.where(t2< 250e-6, 1, 0)
Y2 = np.fft.fft(y2)
Y2 = np.abs(Y2)/N2
f2 = np.fft.fftfreq(N2, 1/f_S2)
ax4.stem(f2, Y2, 'r')
ax4.set_xlabel(r"$f \; / \;\mathrm{Hz} \;\rightarrow$")
ax4.set_ylabel(r"$Y_1(F)\; / \;\mathrm{mV} \;\rightarrow$");
# ----------- Tabellenausgabe --------------
# Drucken Sie i, Y0 und Y1 aus

print("i = | ", end="")
for i in k:
    #Y.append()
    print("{0:>7d} | ".format(i), end="")
print("\nY0 = | ", end="")
for i in k:
    print("{0:7.2f} | ".format(Y0[i]), end="")
print("\nY1 = | ", end="")
for i in k:
    print("{0:7.2f} | ".format(Y1[i]), end="")
print("\nY2 = | ", end="")
for i in k:
    print("{0:7.2f} | ".format(Y2[i]), end="")
```

```
i = |       0 |       1 |       2 |       3 |       4 |       5 |       6 |       7 |       8 |       9 |      10 |      11 |
12 |      13 |      14 |      15 |
Y0 = |  250.00 |  225.08 |  159.15 |   75.03 |    0.00 |  -45.02 |  -53.05 |  -32.15 |   -0.00 |   25.01 |   31.83 |   20.46
|    0.00 |  -17.31 |  -22.74 |  -15.01 |
Y1 = |  250.00 |  226.53 |  163.32 |   79.55 |    0.00 |   53.15 |   67.65 |   45.06 |    0.00 |   45.06 |   67.65 |   53.15
|    0.00 |   79.55 |  163.32 |  226.53 |
Y2 = |  250.00 |  225.17 |  159.41 |   75.30 |    0.00 |   45.47 |   53.83 |   32.80 |    0.00 |   25.84 |   33.15 |   21.49
|    0.00 |   18.55 |   24.63 |   16.45 |
```


250
200
150
100
50
0
$Y_1(F)$ / mV →
−30000
−20000
−10000
0
10000
20000
30000
f / Hz →

2.2.3 FFT mit verlängertem Messfenster ($T_{Mess3} = 4\ \mathrm{ms}$) und $f_{S3} = 16\ \mathrm{kHz}$

Die Abtastfrequenz ist wieder wie zu Beginn $f_{S3} = 16\ \mathrm{kHz}$, jetzt soll getestet werden wie sich eine Verlängerung des Messfensters um den Faktor 4 auswirkt.

VORBEREITUNG:

- Wie viele Samples N_3 benötigen Sie jetzt? Wieviele Perioden passen in Ihr Messfenster?

  > $N_3 = T_{Mess3} \cdot f_{S3} = 64$
  >
  > L = $\frac{T_{Mess3}}{T_1} = 4$

- Welche Frequenzauflösung erzielt man mit diesem Setup? Welche Frequenzkomponente kann maximal dargestellt werden?

  > $\Delta f = \frac{1}{T_{Mess3}} = 250Hz$
  >
  > **SIMULATION:**

Da Ihr Messfenster jetzt mehrere Perioden des Signals umfasst, müssen Sie Ihr Signal erzeugen entweder

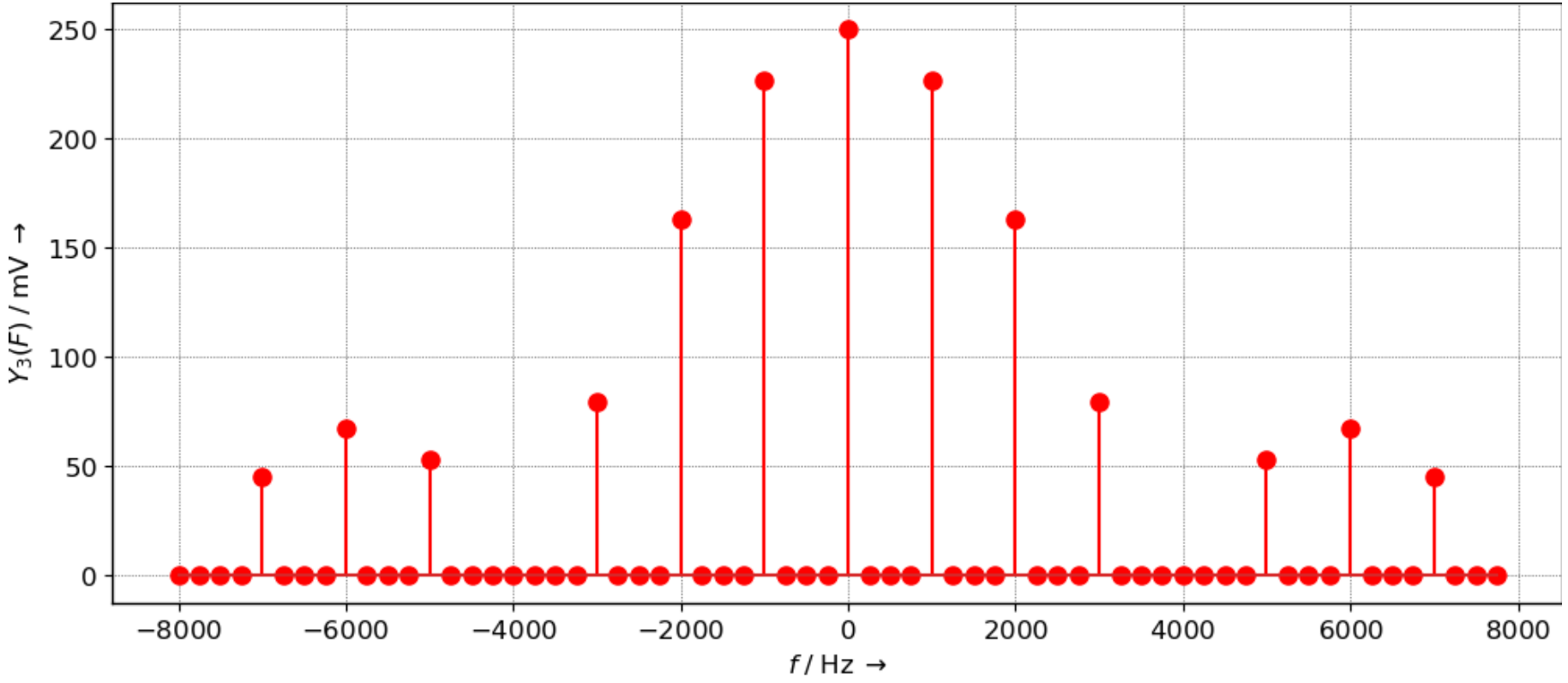
- **Händisch** durch mehrfaches Aneinanderhängen eines Pulses z.B. mit `y3 = np.tile(y1, 4)`
- Durch **Abtasten** des "zeitkontinuierlichen" Signals wie am Anfang von 2.2 beschrieben. Achten Sie wieder darauf, Anfang und Ende passend zu wählen.

Plotten Sie Ihr Zeitsignal, damit Sie sicher sind dass Ihr Signal so aussieht wie Sie sich das vorstellen.

- Vergleichen Sie erneut Rechnung mit Simulation
- Ähnelt das Spektrum mehr dem ursprünglichen Spektrum Y_1 oder dem Spektrum mit erhöhter Abtastrate Y_2?
- Welchen Vorteil hat die Verlängerung des Messfenster? Welchen Vorteil hat sie im hier vorliegenden Fall?

In [ ]:

```
fig, ax = plt.subplots(**size)
N3 = 64
f_S3 = 16 * 1e3
t3 = np.arange(N3)/f_S3
y3 = np.tile(y1, 4)
Y3 = np.abs(np.fft.fft(y3))/N3
f3 = np.fft.fftfreq(N3, 1./f_S3)
ax.stem(f3, Y3, 'r')
ax.set_xlabel(r"$f \; / \;\mathrm{Hz} \;\rightarrow$")
ax.set_ylabel(r"$Y_3(F)\; / \;\mathrm{mV} \;\rightarrow$");
```



2.2.4 FFT über nicht-ganzzahlige Anzahl von Perioden

Das Signal wird wieder mit $f_{S4} = 16\ \mathrm{kHz}$ abgetastet, allerdings ist das Messfenster jetzt nur noch $60\,T_S$ lang.

VORBEREITUNG:

- Welche Länge T_{Mess4} hat jetzt Ihr Messfenster und wieviele Signalperioden passen in das Messfenster?

  > $T_{Mess4} = \frac{60}{f_{S4}} = 3.75\text{ms}$
  >
  > $L = \frac{T_{Mess4}}{T1} = 3.75$, passt in das Messfesnter keine ganzzahlige Anzahl Signalperioden.

- Welche Frequenzauflösung Δf ergibt sich, in welchem Raster liegen die Frequenzbins f_k?

  > $\Delta f = \frac{1}{T_{Mess4}} = 266,67\ \text{Hz}$ $f_k = k \cdot \Delta f$
  >
  > **SIMULATION:**

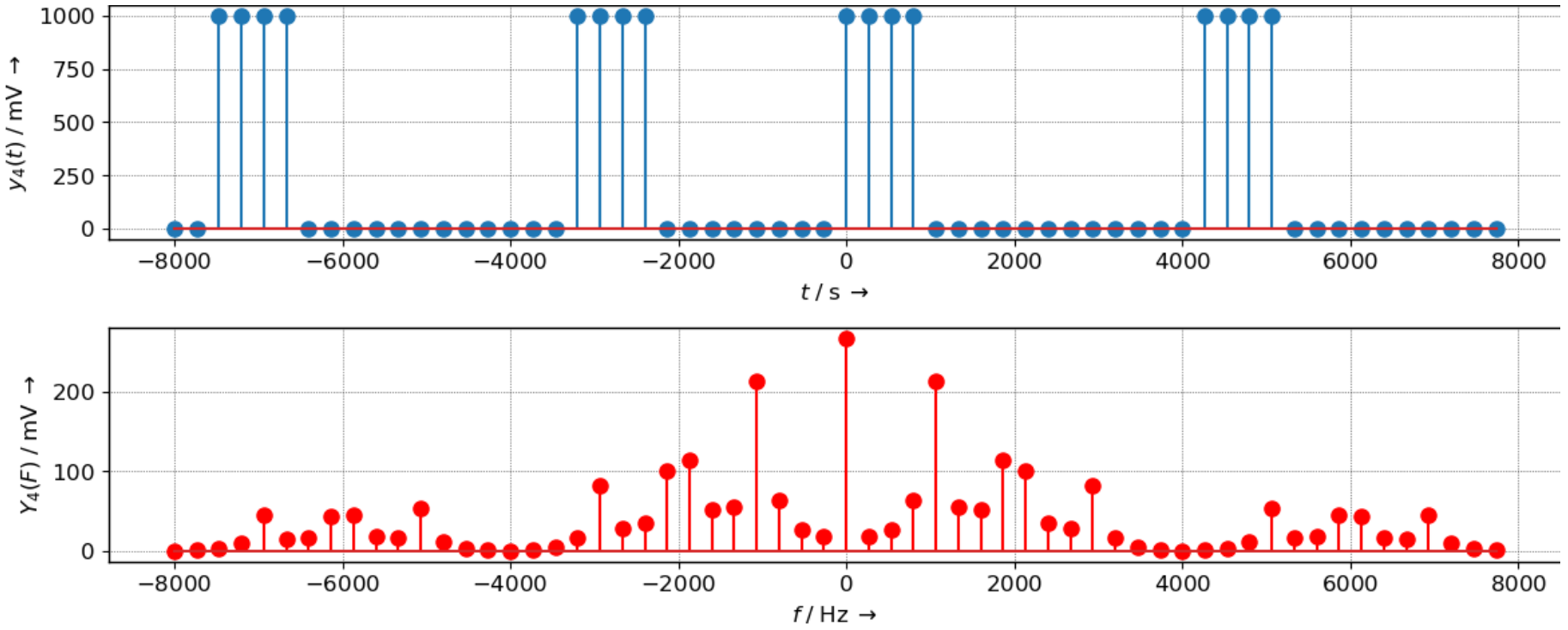
Signal und Zeitvektor erzeugen Sie am einfachsten durch Verkürzen der Signale aus dem letzten Abschnitt, also mit `y4 = y3[:60]` oder alternativ `y4 = y3[:-4]` (negative Indizes zählen vom Ende aus). Der Index, der das Ende markiert ("60" bzw. "-4") definiert dabei den ersten Wert, der *nicht* mit ausgegeben wird. Das ist z.B. bei `arange(0,10,1)` genauso und ist eine Folge des 0-based indexing (der erste Wert eines Arrays wird mit 0 angesprochen).

- Was hat sich im Vergleich zum Spektrum Y_3 des vorigen Abschnitts geändert?

In [ ]:

```
fig, ax = plt.subplots(2, **size)
N4 = 60
f_S4 = 16 * 1e3
t4 = t3[:N4]
y4 = y3[:-4]
Y4 = np.abs(np.fft.fft(y4))/N4
f4 = np.fft.fftfreq(N4, 1./f_S4)
ax[0].stem(f4, y4)
ax[0].set_xlabel(r"$t \; / \;\mathrm{s} \;\rightarrow$")
ax[0].set_ylabel(r"$y_4(t)\; / \;\mathrm{mV} \;\rightarrow$");
ax[1].stem(f4, Y4, "r")
ax[1].set_xlabel(r"$f \; / \;\mathrm{Hz} \;\rightarrow$")
ax[1].set_ylabel(r"$Y_4(F)\; / \;\mathrm{mV} \;\rightarrow$");
fig.set_tight_layout(True)
k = np.arange(12)
print("i  = | ", end="")
for i in k:
    print("{0:>7d} | ".format(i), end="")
print("\n|Y4| = | ", end="")
for i in k:
    print("{0:>7.2f} | ".format(Y4[i]), end="")
print("\n")
```

```
i  = |       0 |       1 |       2 |       3 |       4 |       5 |       6 |       7 |       8 |       9 |      10 |      11
|
|Y4| = |  266.67 |   18.52 |   26.52 |   62.62 |  212.93 |   55.77 |   51.29 |  113.09 |   99.64 |   34.91 |   28.87 |   81.2
8 |
```


1000
750
500
250
0
$y_4(t)$ / mV →
−8000
−6000
−4000
−2000
0
2000
4000
6000
8000
t / s →
200
100
0
$Y_4(F)$ / mV →
f / Hz →

3. Schätzung des Fourierspektrums von Sinustönen mit überlagertem Rauschen durch FFT

In diesem Versuchsteil versuchen wir mit Hilfe der FFT Amplituden und Frequenzen von zwei Sinustönen zu schätzen mit $f_1 = 990\ \mathrm{Hz}$ und $A_1 = 100\ \mathrm{mV}$ sowie $f_2 = 1010\ \mathrm{Hz}$ und $A_2 = 2\ \mathrm{mV}$, überlagert ist eine gleichverteilte Rauschstörung im Bereich $\pm 1\ \mathrm{mV}$.

Tipp: Diesen Versuchsteil können Sie auch mit pyfda durchführen, im Tab `y[n]` wählen Sie dazu ein Sinussignal mit überlagertem Rauschen aus und schauen sich das Spektrum im Untertab "Frequency" an. Enablen Sie "Stimulus X" und deaktivieren Sie "Response Y".

3.1 Abtastfrequenz frei wählbar

Die Anzahl der Datenpunkte N für die FFT und die Abtastfrequenz f_S können zunächst frei gewählt werden.

VORBEREITUNG:

- Welche Abtastfrequenz benötigen Sie mindestens?

  > Die Abtastfrequenz muss größer als $2f_2 = 2020\ \mathrm{Hz}$ sein.

- Wählen Sie die kleinstmöglichen Werte für N und f_S so, dass kein Frequenzfehler und kein Leckeffekt auftritt. Tipp: Bestimmen Sie mit $T_{min} = L_1 T_1 = L_2 T_2$ zunächst das kürzeste Zeitfenster, in dem sowohl f_1 als auch f_2 periodisch sind. Danach können Sie mit $T_{Mess} = \frac{N}{f_S} = M T_{min}$ eine Bedingung für N und T_S ableiten

  > mit $T_{min} = \frac{L_1}{990Hz} = \frac{L_2}{1010Hz}$ bekommt man $\frac{L_1}{L_2} = \frac{99}{101}$ also $L_1 = 99, L_2 = 101$ und $T_{min} = 100ms$ Daraus erhält man $T_{Mess} = \frac{N}{f_S} = M\frac{99}{990Hz}$ und damit $f_S = 10Hz$ und $N = M = 1$.
  >
  > Diese Kombination ist zwar periodisch, verletzt aber natürlich das Nyquist-Kriterium. Durch Erhöhen von Abtastfrequenz und N um den gleichen Faktor (damit sich $T_{Mess} = T_{min}$ nicht ändert) erhählt man schließlich $f_s = 2030Hz$ und $N = 203$.

- Wie groß ist das Verhältnis der Amplituden und wie groß das Verhältnis der Leistungen beider Sinustöne in dB?

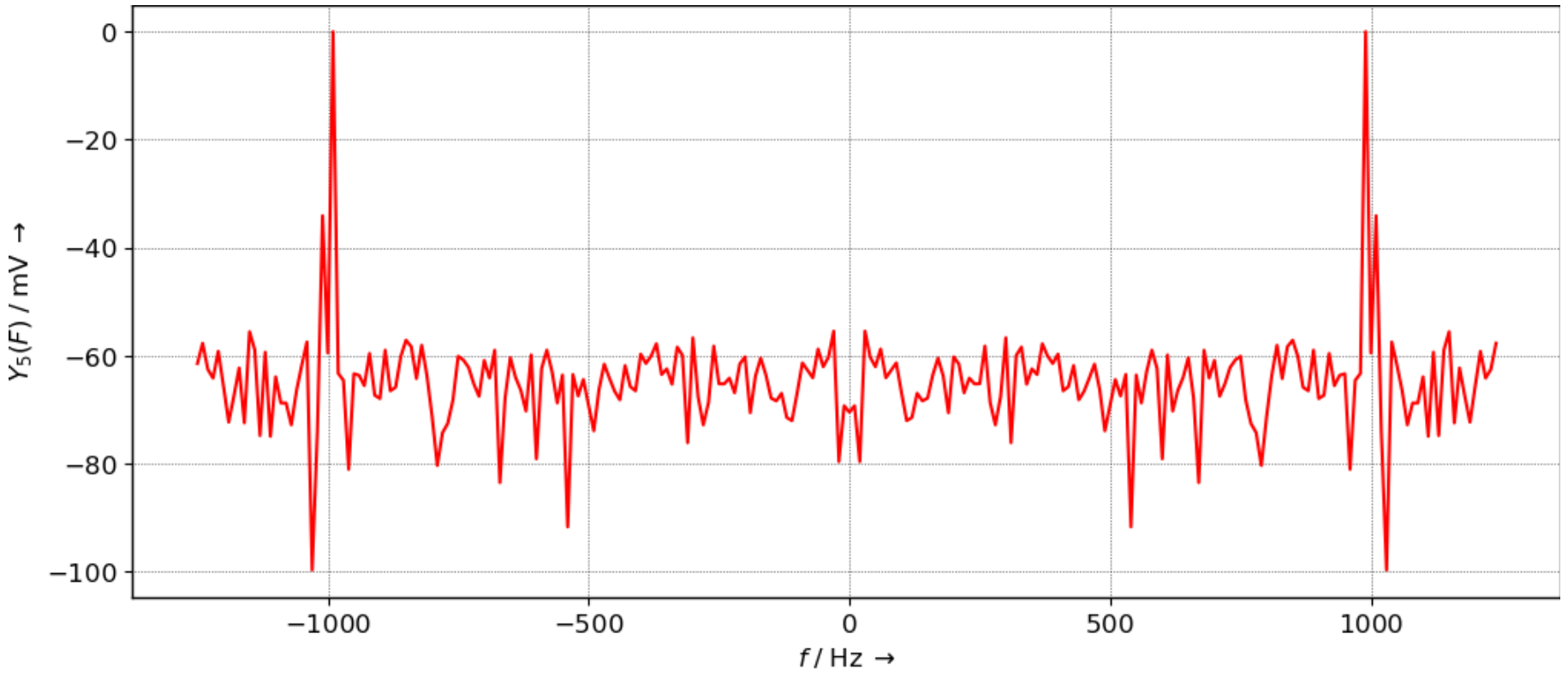
  > A_1

> $\frac{A_1}{A_2} = 50$ damit $20 \log_{10} 50 = 34dB$.
>
> $\frac{P_1}{P_2} = \frac{A_1^2}{A_2^2} = 2500$ damit $10 \log_{10} 2500 = 34dB$.

SIMULATION:

- Wählen Sie $f_S = 2500\ \mathrm{Hz}$ und $N = 250$ für ein leichter ablesbares Spektrum.
- Stellen Sie das Spektrum als (Amplituden-)Betragsspektrum im logarithmischen Maßstab dar, die Werte sollen relativ zum Maximum dargestellt werden. Zur Verbesserung der Darstellung können Sie mit `np.fft.fftshift` positive und negative Achse von Frequenzvektor und Spektrum vertauschen.

In [ ]:
```
N5 =  250
f_S5 = 2500
n = np.arange(N5)
t = n/f_S5
noi = 2 * np.random.rand(N5) - 1
y5 = 100 * np.sin(2 * np.pi * 990 * t) + 2 * np.sin(2 * np.pi * 1010 * t) + noi
Y5 = np.abs(np.fft.fft(y5))/N5
Y5 /= np.max(Y5)
Y5 = np.fft.fftshift(20 * np.log10(Y5))
f5 = np.fft.fftshift(np.fft.fftfreq(N5, 1./f_S5))
fig, ax = plt.subplots(**size)
ax.plot(f5, Y5, 'r')
ax.set_xlabel(r"$f \; / \;\mathrm{Hz} \;\rightarrow$")
ax.set_ylabel(r"$Y_5(F)\; / \;\mathrm{mV} \;\rightarrow$");
```


0
−20
−40
−60
−80
−100
$Y_5(F)$ / mV →
−1000
−500
0
500
1000
f / Hz →

3.2 Feste Abtastfrequenz $f_S = 4\,\mathrm{kHz}$

Die Abtastfrequenz ist jetzt fest eingestellt auf $f_S = 4\,\mathrm{kHz}$, die Anzahl der Datenpunkte N für die FFT soll eine Zweierpotenz sein, $N = 2^L$, für eine möglichst effiziente Berechnung der FFT.

Auch diesen Unterpunkt können Sie mit pyfda bearbeiten.

VORBEREITUNG:

- Gibt es bei der gewählten Abtastfrequenz einen Wert für N, bei dem kein Frequenzfehler und kein Leckeffekt auftritt (mit und ohne Beschränkung auf Zweierpotenzen)? Gibt es für $N = 256$ eine Abtastfrequenz, bei der kein Leckeffekt auftritt?

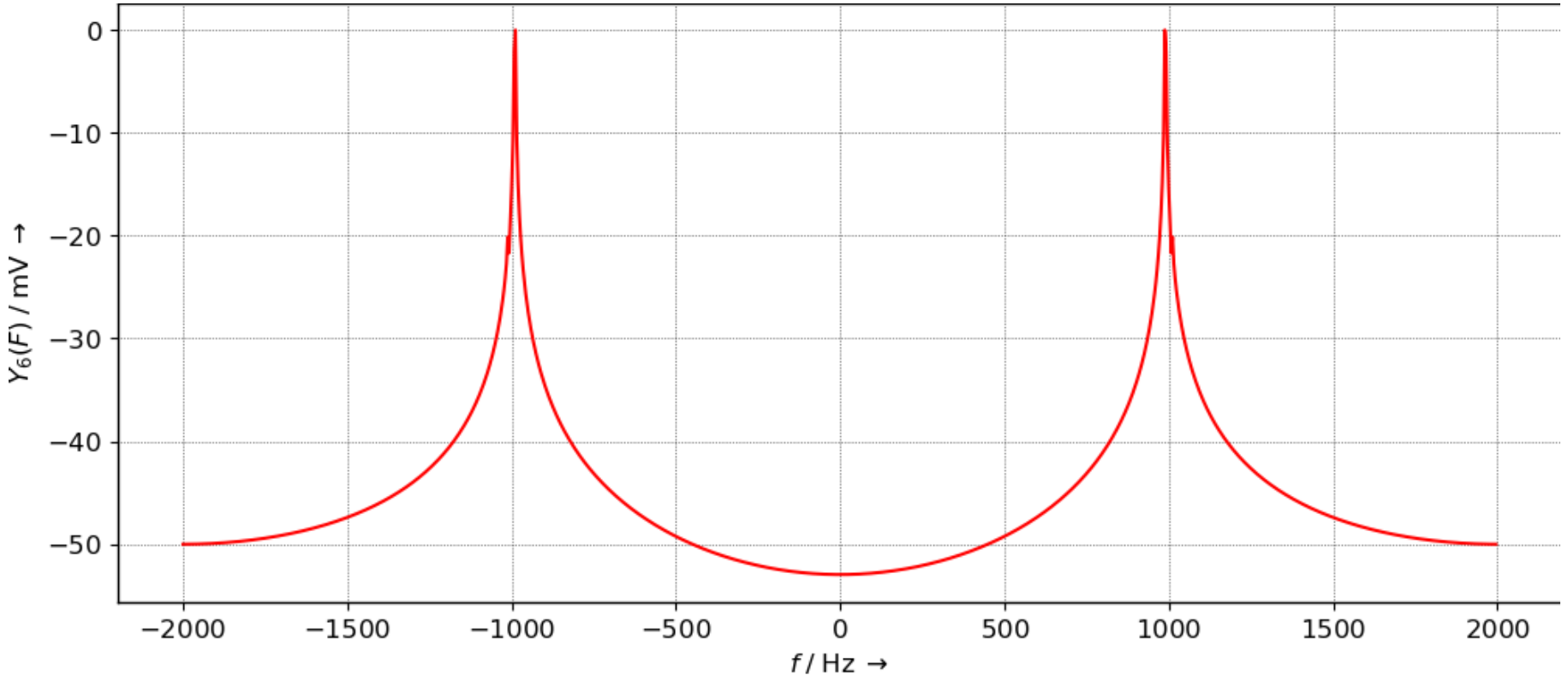
  > Wie weiter oben ermittelt ist auch $f_S = 2500$ und $N = 250$ eine Kombination, bei der beider Töne kohärent abgetastet werden.
  >
  > Analog dazu benötigt man für $N = 256$ die Abtastfrequenz $f_S = 2560Hz$.
  >
  > **SIMULATION:**

- Der Leckeffekt soll jetzt durch Verwendung "weicherer" Fenster reduziert werden. Fensterfunktionen finden Sie unter `scipy.signal.windows` (https://docs.scipy.org/doc/scipy/reference/signal.windows.html). Vor Berechnung der FFT müssen Sie dazu Ihr Signal mit der Fensterfunktion multiplizieren, also z.B. `y_win = y * sig.windows.hann(256, sym=False)` . Dabei wurde angenommen, dass auch Ihr Zeitsignal `y` eine Länge von $N = 256$ hat. Für die Spektralanalyse periodischer Signale (im Gegensatz zum Filterdesign und zur Analyse von impulsförmigen Signalen) muss `sym = False` gewählt werden.

- Testen Sie den Einfluss der Fensterlänge $N = 256$, $N = 512$ und $N = 1024$ ... und den Effekt eines rechteckförmigen (`boxcar` bzw. gar keine Fensterung) und eines Hann-Fensters. Wie genau werden die Amplituden und Frequenzen geschätzt? Schauen Sie sich auch die anderen Fensterfunktionen an.

In [ ]:

```
N6 =  1024
f_S6 = 4000
n = np.arange(N6)
t = n/f_S6
noi = 2 * np.random.rand(N6) - 1
y6 = 100 * np.sin(2 * np.pi * 990 * t) + 2 * np.sin(2 * np.pi * 1010 * t) #+ noi
y6_win = y6 * sig.windows.boxcar(N6, sym=True)
Y6 = np.abs(np.fft.fft(y6_win))/N6
Y6 /= np.max(Y6)
Y6 = np.fft.fftshift(20 * np.log10(Y6))
f6 = np.fft.fftshift(np.fft.fftfreq(N6, 1./f_S6))
fig, ax = plt.subplots(**size)
ax.plot(f6, Y6, 'r')
ax.set_xlabel(r"$f \; / \;\mathrm{Hz} \;\rightarrow$")
ax.set_ylabel(r"$Y_6(F)\; / \;\mathrm{mV} \;\rightarrow$");
```



4. Schätzung des Fourierspektrums eines rechteckförmigen Impulses durch FFT

In diesem Versuchsteil schätzen wir das Fourierintegral des rechteckförmigen Impulses $y(t)$ mit $\Delta T = 250\,\mu$s und $A = 1000\,\mathrm{mV}$ (s. nächster Plot) mit Hilfe der FFT.

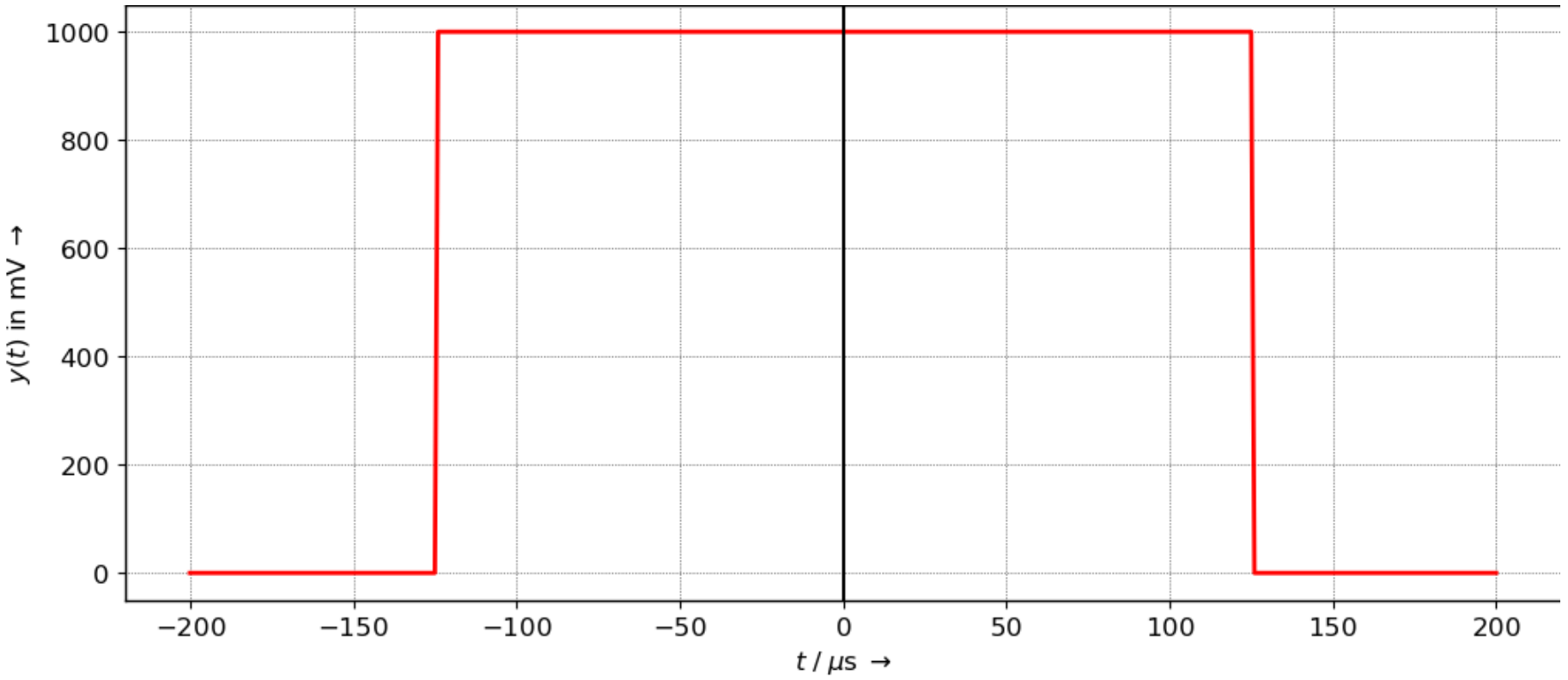
4.1 Berechnung der Fouriertransformierten

VORBEREITUNG:

- Berechnen Sie die Fouriertransformierte des Impulses und skizzieren Sie die Betragsfunktion (oder plotten Sie sie).
- Bei welcher Frequenz hat die Fouriertransformierte die erste Nullstelle?

> Die erste Nullstelle f_0 der Fouriertransformierten ist bei $\pi f_0 \Delta T_1 = \pi \;\Rightarrow\; f_0 = 1/\Delta T = 4\,\mathrm{kHz}$.

In [ ]:
```
fig, ax = plt.subplots(**size)
Delta_T = 250e-6
A = 1000
t = np.arange(-200e-6, 200e-6, 1e-6)
ax.plot(t*1e6, A*np.where((t >= -Delta_T/2) & (t < Delta_T/2), 1, 0), 'r', lw=2)
ax.axvline(0, ls='-',color='k')
ax.set_xlabel(r"$t \; / \;\mu\mathrm{s} \;\rightarrow$")
ax.set_ylabel(r"$y(t) \; \mathrm{in \;mV} \;\rightarrow$");
```


1000
800
600
400
200
0
y(t) in mV →
−200
−150
−100
−50
0
50
100
150
200
t / μs →

Das zeitkontinuierliche Signal $y(t)$ hat eine endliche zeitliche Ausdehnung und endliche Energie, daher kann sein Amplitudenspektrum mit dem Fourierintegral beschrieben werden:

$$Y\left(f\right) = \int_{-\infty}^{\infty} y(t) \mathrm{e}^{-\mathrm{j}2\pi ft} \mathrm{d}t$$

Der Impuls ist symmetrisch zur y-Achse was die Berechnung deutlich vereinfacht:

$$Y\left(f\right) = A \int_{-\Delta T/2}^{\Delta T/2} \mathrm{e}^{-\mathrm{j}2\pi ft} \mathrm{d}t = \frac{A}{-\mathrm{j}2\pi f} \mathrm{e}^{-\mathrm{j}2\pi ft} \Big|_{-\Delta T/2}^{\Delta T/2} = A \frac{\mathrm{e}^{\mathrm{j}\pi f \Delta T} - \mathrm{e}^{-\mathrm{j}\pi f \Delta T}}{2\pi \mathrm{j} f} = A \frac{\sin \pi f \Delta T}{\pi f} \tag{3}$$

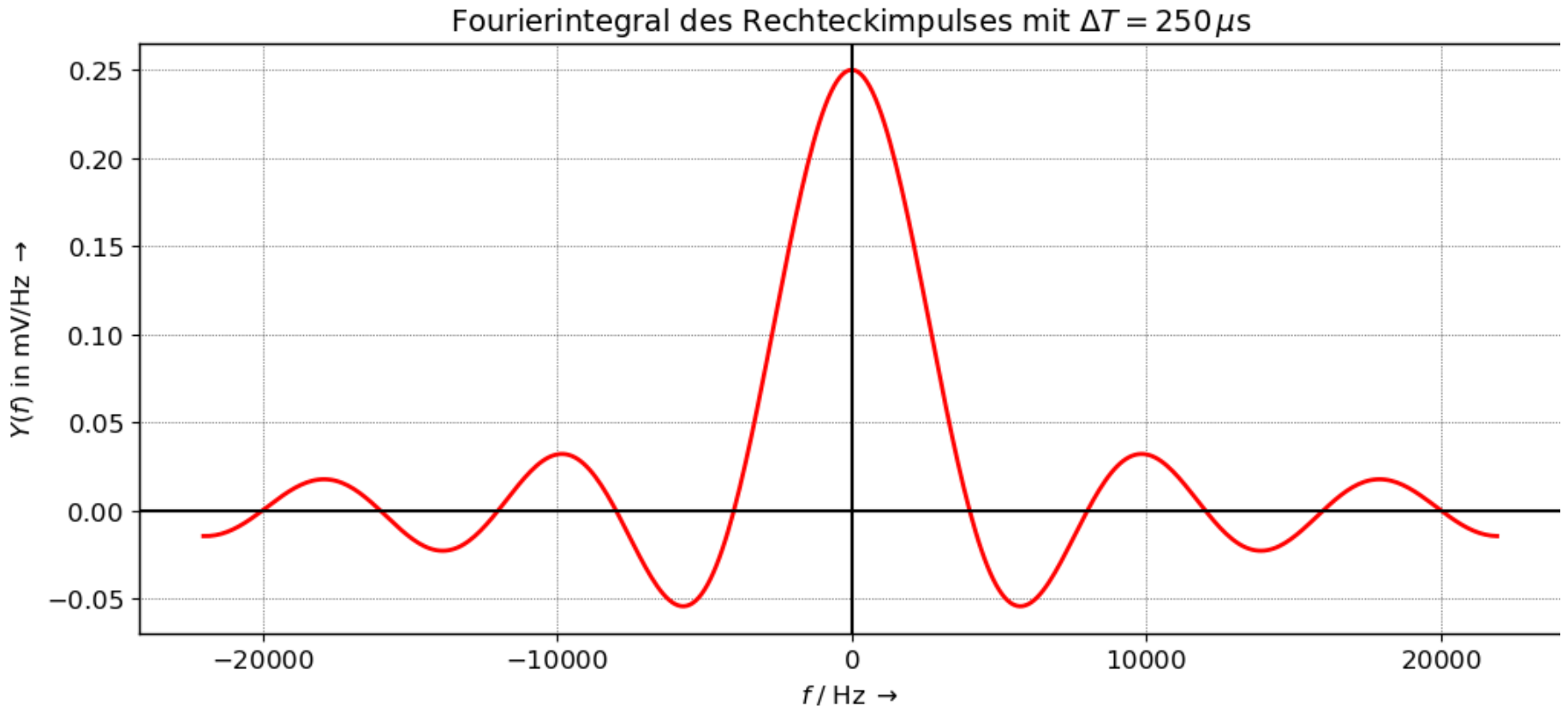
$$= A\Delta T \frac{\sin \pi f \Delta T}{\pi f \Delta T} = A\Delta T \mathrm{sinc} f\Delta T \text{ mit } \mathrm{sinc} f = \frac{\sin \pi f}{\pi f} \tag{4}$$

Das Ergebnis ist die Amplitudenspektrumsdichte in V/Hz.

Berechnet man die FFT anstatt des Fourierintegrals, überstreicht jeder Frequenzpunkt die Bandbreite $\Delta f = f_S/N$. Möchte man die FFT eines Energiesignals so skalieren dass man (in etwa) identische Zahlenwerte erhält wie bei der spektralen Amplitudendichte des ursprünglichen zeitkontinuierlichen Signals, so muss man mit $1/\Delta f = N/f_S$ multiplizieren. Da die FFT bereits mit $1/N$ skaliert ist, hebt sich der Faktor N auf. Man muss also nur durch f_S dividieren.

Für unendlich ausgedehnte Leistungssignale (periodische Signale, stationäre und nicht-stationäre Prozesse) konvergiert das Integral nicht, man nimmt stattdessen die Fourierreihe (periodische Signale) oder die spektrale Leistungsdichte, die über die Autokorrelationsfunktion berechnet wird (aber nicht hier und jetzt ...).

In [ ]:
```
fig, ax = plt.subplots(**size)
Delta_T = 250e-6
A = 1000
f = np.arange(-2.2e4, 2.2e4, 1e2)
ax.plot(f, A * Delta_T*np.sinc(f * Delta_T), 'r', lw=2)
ax.axvline(0, ls='-',color='k'); ax.axhline(0, ls='-',color='k')
ax.set_title(r"Fourierintegral des Rechteckimpulses mit $\Delta T = 250\, \mu \mathrm{s}$")
ax.set_xlabel(r"$f \; / \;\mathrm{Hz} \;\rightarrow$")
ax.set_ylabel(r"$Y(f) \; \mathrm{in \;mV/Hz} \;\rightarrow$");
```


Fourierintegral des Rechteckimpulses mit $\Delta T = 250\,\mu s$
0.25
0.20
0.15
0.10
0.05
0.00
−0.05
Y(f) in mV/Hz →
−20000
−10000
0
10000
20000
f / Hz →

4.2 Schätzung der Fouriertransformierten mit Hilfe der FFT

Im folgenden soll die Fouriertransformierte des Impulses mit Hilfe einer FFT abgeschätzt werden, dazu wird der Impuls mit $L = 16$ Samples abgetastet.

VORBEREITUNG:

- Welche Abtastfrequenz f_S ist dazu notwendig?

  > $f_S = \frac{L}{\Delta T} = 64KHz$

- Wie groß ist die Frequenzauflösung Δf? Wieviele Frequenzpunkte erhält man zwischen $f = 0$ und der ersten Nullstelle f_0?

  > $\Delta f = \frac{1}{T_{Mess}} = \frac{1}{\Delta T} = \frac{f_S}{L} = 4KHz$
  >
  > Wegen $\Delta f = \frac{1}{\Delta T} = f_0$ erhält man unabhängig von der Abtastfrequenz einen Frequenzpunkt bei $f = 0$ und bereits den nächsten bei $f = f_0$

- Kann man die graphische Darstellung des Amplitudenspektrums zwischen $f = 0$ und der ersten Nullstelle f_0 durch Änderung der Abtastfrequenz verbessern?

  > Nein

- Wie kann die graphische Darstellung des Betragsgangs verbessert werden ohne die Abtastfrequenz zu verändern?

  > Durch nur Verlängerung des Messfensters erhöht man die Auflösung $\Delta f = \frac{1}{T_{mess}}$. Da die Signallänge begrenzt ist auf ΔT, kann man das Fenster nur durch Anhängigen con Nullen verlängern("Zero Padding") die FFT wird dann berechnet über insgesamt N_{FFT} Datenpunkte, von denen nur L ungleich null sind.

- Wie müssen die Ergebnisse skaliert werden, um physikalisch korrekte Werte für die spektrale Amplitudendichte in V/Hz zu bekommen (so wie theoretisch berechnet)?

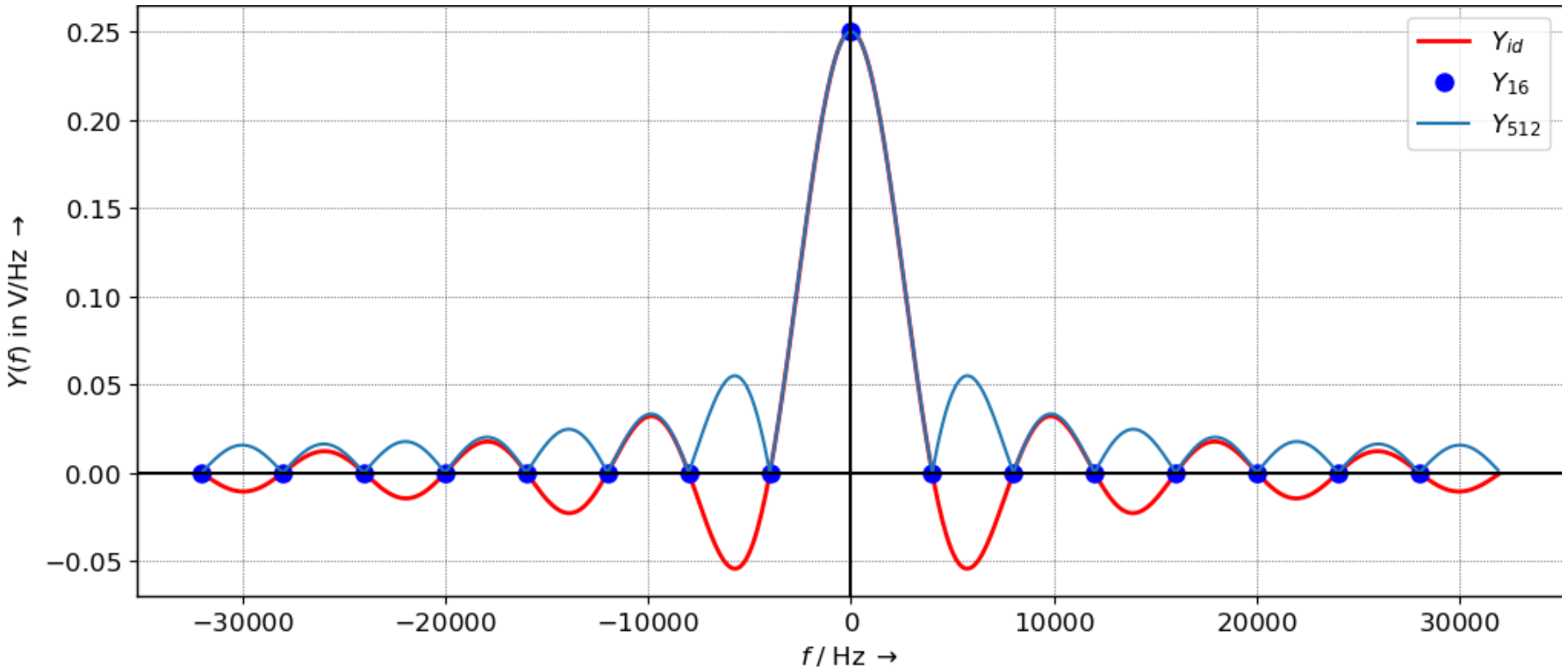
  > Tipp: eine Skalierung mit $\frac{1}{N_{FFT}}$ ist nur sinnvol für ein periodisches Signal mit einer Periode von N_{FFT} Samples. Der mit

> Zero-Padding verlängerte Puls hätte als periodischer Rechteckpuls einen Duty Cycle $\frac{L}{N_{FFT}}$ dessen Amplitudenspektrum mit zunehmendem N_{FFT} immer mehr abnimmt.
>
> Daher darf hier nur mit $\frac{1}{L}$ skalert werden.

SIMULATION:

- Berechnen Sie die FFT des Rechteckpulses mit $L = 16$ und mit $N_{FFT} = 2^9 = 512$ Punkten.
- Erstellen Sie einen Plot mit der FFT mit $L = 16$ Punkten, $N_{FFT} = 512$ Punkten und dem idealen Amplitudenspektrum des Rechteckimpulses.

In [ ]:
```
fig, ax = plt.subplots(**size)
f_S = 64e3
Delta_T = 250e-6
A = 1000
f = np.arange(-32e3, 32e3, 1e2)
f_16 = np.fft.fftshift(np.fft.fftfreq(16, d= 1/f_S))
f_512 = np.fft.fftshift(np.fft.fftfreq(512, d= 1/f_S))
y_id = A * Delta_T*np.sinc(f * Delta_T)
y_16 = A*np.ones(16)
Y_16 = np.fft.fftshift(np.abs(np.fft.fft(y_16))/f_S)
Y_512 = np.fft.fftshift(np.abs(np.fft.fft(y_16, 512))/f_S)
ax.plot(f, y_id, 'r', lw=2, label="$Y_{id}$")
ax.plot(f_16, Y_16, 'bo', label="$Y_{16}$")
ax.plot(f_512, Y_512, label="$Y_{512}$")
ax.legend()
ax.axvline(0, ls='-',color='k'); ax.axhline(0, ls='-',color='k')
ax.set_xlabel(r"$f \; / \;\mathrm{Hz} \;\rightarrow$")
ax.set_ylabel(r"$Y(f) \; \mathrm{in \;V/Hz} \;\rightarrow$");
```



Copyright