Denys Bast, Armin Galetzka
Institut für Theorie Elektromagnetischer Felder

Projektseminar Beschleunigertechnik

TECHNISCHE UNIVERSITÄT DARMSTADT

Inhaltsverzeichnis

1 Abstract

Das breitbandige Bestimmen der Übertragungsfunktion eines unbekannten Systems ist mit Methoden wie dem Sinus Sweep aufwendig und zeitintensiv. Pseudorauschen ist eine Möglichkeit, schnell und effizient den Frequenzgang eines Systems in einem breiten Spektrum zu bestimmen. Im Rahmen dieses Seminars wurde ein Python Tool zur automatisierten Messung der Übertragungsfunktion eines Kavitätensystems implementiert und erfolgreich getestet. Als Pseudo-Rauschsignal wurde ein MLBS (*maximum length binary sequence*) Signal verwendet. Der Vergleich der neuen Methode mit dem Sinus Sweep ergab eine hohe Übereinstimmung der Übertragungsfunktion bei gleichzeitiger Reduktion der Messdauer von 30 Minuten auf 42 Sekunden.

2 Einführung der Problemstellung

Für das Großprojekt FAIR werden von der Organisationseinheit SIS - Abteilung Ring RF des GSI Helmholtzzentrums für Schwerionenforschung Barrier-Bucket Systeme entworfen. Diese Systeme dienen der longitudinalen Manipulation von Teilchenstrahlen. Dazu müssen am Gap der Kavität einzelne sinusförmige Spannungspulse erzeugt werden. Das dafür geplante System besteht aus Signalgenerator, Verstärker und Kavität. Für ausreichend kleine Eingangssignale verhält sich das System linear. Das Verhalten des Systems kann dann mathematisch durch den Frequenzgang $\underline{H}(\omega)$ beschrieben werden. Abbildung 2.1 zeigt die auszumessende Kavität mit den Spannungen $\underline{U}_{\text{in}}$, $\underline{U}_{\text{out},1}$ und $\underline{U}_{\text{out},2}$. Die Spannung $\underline{U}_{\text{in}}$ liegt an den Einkopplungsschleifen an. Die Ausgangsspannungen liegen am Gap an. Messtechnisch ist es einfacher das Ausgangssignal an zwei Stellen zu erfassen. Die gesamte Ausgangsspannung ist gegeben durch

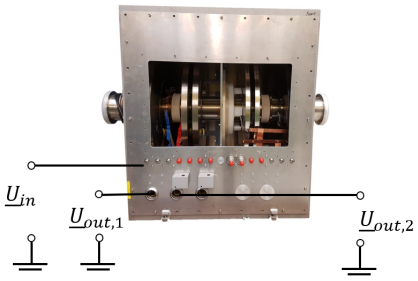


Abbildung 2.1: Kavität

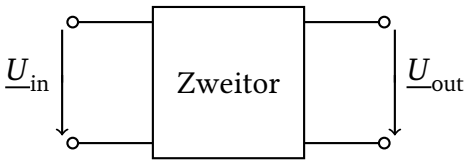


Abbildung 2.2: Beliebiges Zweitor

$$\underline{U}_{\text{out}} = \underline{U}_{\text{out},1} - \underline{U}_{\text{out},2}. \tag{1}$$

Eine abstraktere Betrachtung des Problems führt auf die Darstellung durch ein Zweitor, siehe Abbildung 2.2. Wenn der Frequenzgang bekannt ist, kann im Frequenzbereich (Fouriertransformation) zu einem Eingangssignal $\underline{U}_{\text{in}}$ das Ausgangssignal $\underline{U}_{\text{out}}$ mit

$$\underline{U}_{\text{out}}(\omega) = \underline{H}(\omega) \cdot \underline{U}_{\text{in}}(\omega) \tag{2}$$

bestimmt werden. Durch Umformen von Gleichung (2) auf

$$\underline{U}_{\text{in}}(\omega) = \underline{U}_{\text{out}}(\omega) \cdot \underline{H}(\omega)^{-1}, \tag{3}$$

lässt sich aus einem bekannten Ausgangssignal das Eingangsignal berrechnen. [1]

Ziel ist es, ein Einzelsinus-Signal am Ausgang zu erzeugen. Für eine Periodendauer von T_{BB} ist der Einzelsinus wie folgt definiert. [7]

$$U_{\sin}(t) = \begin{cases} -\widehat{U} \cdot \sin\left(\frac{2\pi}{T_{\mathrm{BB}}} t\right) & \text{für } -\frac{T_{\mathrm{BB}}}{2} < t < \frac{T_{\mathrm{BB}}}{2} \\ 0 & \text{sonst} \end{cases} \tag{4}$$

Abbildung 2.3 zeigt ein solches Einzelsinus Signal. Das Spektrum des Einzelsinus setzt sich aus zwei

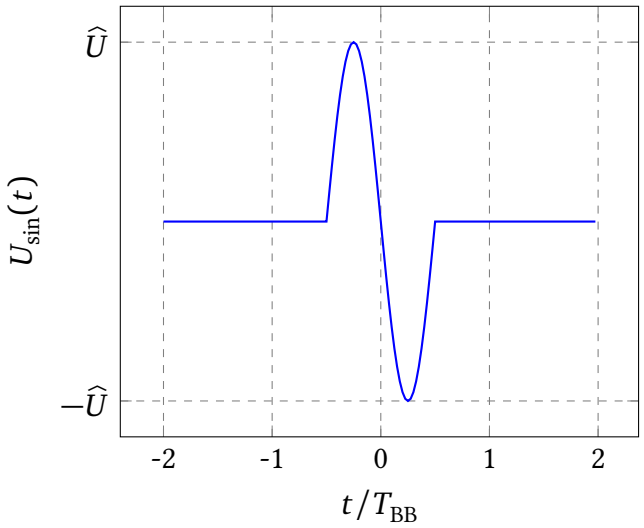


Abbildung 2.3: Einzelsinus

verschobenen si-Funktionen zusammen. Abbildung 2.4 zeigt die Fouriertransformierte des Einzelsinus der Amplitude $\widehat{U}$ und der Kreisfrequenz ω_{BB}. Ein idealer Einzelsinus beinhaltet also beliebig hohe Frequenzen. Da dies technisch nicht realisierbar ist, kommt es zu sogenannten Überschwingern vor und hinter dem Sinuspuls. Diese sollen im Bereich von $2,5\,\%$ der Amplitude $\widehat{U}$ liegen. [2, 7, 11, 9]

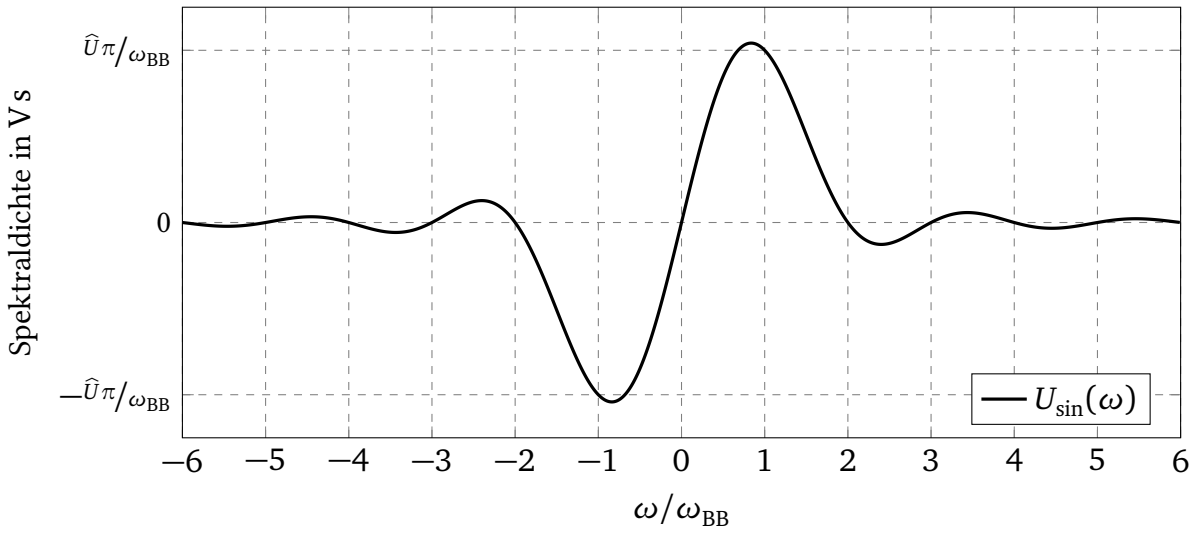


Abbildung 2.4: Frequenzspektrum eines Einzelsinus der Amplitude $\widehat{U}$ und der Kreisfrequenz ω_{BB} [7]

Die Ausgangspannung in Gleichung 3 ist damit gegeben durch $\underline{U}_{\text{out}}(\omega) = U_{\text{sin}}(\omega)$. Wird der Frequenzgang gemessen, kann mit Gleichung 3 das Eingangssignal berechnet werden, das angelegt werden muss, um einen Einzelsinus am Ausgang des Systems zu erhalten. [1, 7]

Es existiert ein intern entwickeltes Programm, das bei bekannter Übertragungsfunktion ein solches vorverzerrtes Signal erzeugt [6]. Bisher wird die Übertragungsfunktion mit Hilfe eines sinusförmigen Kleinsignal-Sweeps im Frequenzbereich von 10 kHz bis 80 MHz gemessen. Um sicherzustellen, dass sich das System im eingeschwungenen Zustand befindet, muss der Sweep ausreichend langsam durchgeführt werden. Dies führt zu einer langen und aufwändigen Prozedur der Messung. [1]

Ziel dieses Projektseminars ist es, ein Python-Tool zu entwickeln, mit dessen Hilfe die Messung des Frequenzganges des Systems automatisiert und zeiteffizient durchgeführt werden kann.

Weißes Rauschen regt gleichzeitig breite Frequenzbereiche an und eignet sich daher als Eingangssignal anstelle des Sweeps. So kann mit einem Messdurchgang die Übertragungsfunktion für alle interessanten Frequenzen bestimmt werden. Dadurch kann die Messung sehr viel schneller durchgeführt werden. [12]

3 Pseudorauschen

Pseudo-Rauschsignale sind digital erzeugte Folgen von Zufallszahlen, die das Verhalten und die Eigenschaften von weißem Rauschen nachbilden sollen. Im Folgenden wird erläutert, was Pseudo-Rauschsignale auszeichnet und was deren Eigenschaften sind. Anschließend wird dargelegt wie Pseudorauschen erzeugt werden kann. Im Anschluss wird die für dieses Projektseminar erstellte Implementierung in Python vorgestellt. [12]

3.1 Erzeugung von Pseudorauschen

Weißes Rauschen kann als Folge identisch verteilter, unabhängiger Zufallsvariablen betrachtet werden. Damit beschreibt es einen stochastischen Prozess, der mittelwertfrei und in sich unkorreliert ist. Außerdem besitzen dessen Variablen eine gleiche Verteilung. [12, 3]

Um digitale Rauschsignale als Eingangssignale verwenden zu können, müssen diese am Digitalrechner generiert werden. Dies geschieht über sogenannte Pseudo-Zufallszahlen, deren Eigenschaften bezüglich Verteilung, Mittelwert und Korrelation die von weißem Rauschen nachbilden. Im Folgenden soll T_{ns} die Taktzeit des Rauschens und T_{s} die Abtastzeit der Messung bezeichnen. Wird eine periodische

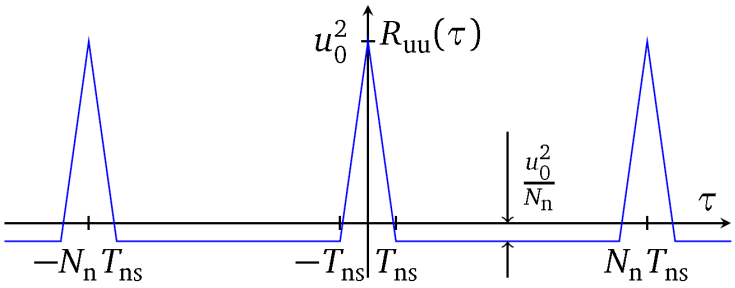


Abbildung 3.1: Autokorrelationsfunktion eines PBRS mit Taktzeit T_{ns} und Länge N_{N} [12]

Folge von binären Pseudo-Zufallszahlen erzeugt und hat diese eine Autokorrelationsfunktion nach Abbildung 3.1, wird dieses Signal als Pseudo-Binär-Zufallsprozess (*PBRS - pseudo random binary signal*) bezeichnet. Eine binäre Zufallszahl kann lediglich zwei Werte annehmen. Es werden die Zahlen 0 und 1 zufällig gezogen, wobei beide die gleiche Wahrscheinlichkeitsverteilung besitzen. Abbildung 3.2 zeigt die diskrete Verteilung der Zufallsvariablen. Die erzeugten Zufallszahlen werden am PC berechnet und sind reproduzierbar, erfüllen aber wichtige Kriterien echter Zufallszahlen. [4]

Eine Möglichkeit ein solches Signal zu erzeugen, ist die MLBS (*maximum length binary sequence*). Dabei wird ein Schieberegister verwendet, um die Folge mit den gewünschten Eigenschaften zu erzeugen. Das Schieberegister wird mit zufälligen Zahlenwerten aus $\{0, 1\}$ initialisiert, wobei mindestens

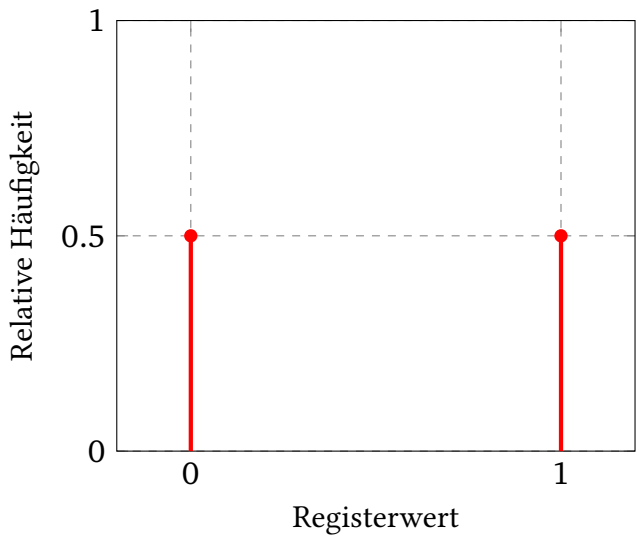


Abbildung 3.2: Diskrete Verteilung der Zufallsvariablen

Bits	N_n	Rückführung
6	63	$0 \oplus 5$
7	127	$0 \oplus 6$
8	255	$0 \oplus 1 \oplus 6 \oplus 7$
9	511	$3 \oplus 8$
10	1023	$2 \oplus 9$
11	2047	$1 \oplus 10$

Tabelle 3.1: Rückgeführte Bit zum Schieben des Registers

ein Bit den Zustand 1 haben muss. Daher ergibt sich für ein Schieberegister mit b Bit eine Folge der Länge

$$N_\mathrm{n} = 2^b - 1. \tag{5}$$

Abbildung 3.3 zeigt ein solches Schieberegister für $b = 7$ Bit. Das Schieberegister gibt mit jedem Takt

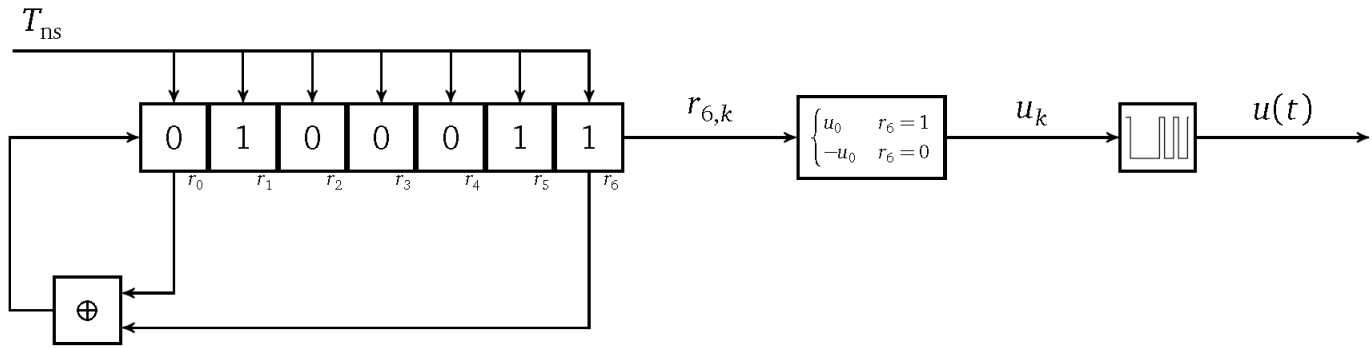


Abbildung 3.3: Schieberegister zur Erzeugung eines MLBS der Länge $N_N = 127$ [12]

ein Bit 0 oder 1 aus, wobei jedes einer Spannung u_0 oder $-u_0$ zugeordnet wird. Das erzeugte Signal hat also nur zwei diskrete Spannungsniveaus. Es müssen mindestens zwei Bit zurückgeführt werden, um ein MLBS erzeugen zu können. Eines der beiden Bit muss das letzte sein. Welches Bit noch zurückgeführt wird, hängt von der Länge des Schieberegisters ab. Bei 6 und 7 Bit wird jeweils das erste und das letzte Bit zurückgeführt. Die Kombination der zurückgeführten Bits ist in Tabelle 3.1 für verschiedene Längen b des Schieberegisters gezeigt.

Für ein solches Signal lässt sich die Fourier-Reihe bestimmen, deren absolute Koeffizienten gegeben sind durch

$$|u_\ell| \leq \begin{cases} \frac{u_0}{N_\mathrm{n}} & \ell = 0, \\ u_0 \frac{\sqrt{N_\mathrm{n}+1}}{N_\mathrm{n}} \left| \frac{\sin\left(\frac{\ell\pi}{N_\mathrm{n}}\right)}{\frac{\ell\pi}{N_\mathrm{n}}} \right| & \ell > 0. \end{cases} \tag{6}$$

Die Dämpfung der Koeffizienten ist durch die einhüllende si-Funktion beschrieben. Für $\ell = N_\mathrm{n}$ gilt für Gleichung (6)

$$|u_{N_\mathrm{n}}| = u_0 \frac{\sqrt{N_\mathrm{n}+1}}{N_\mathrm{n}} \left| \frac{\sin(\pi)}{\pi} \right| = 0. \tag{7}$$

Der Fourier Koeffizient verschwindets bei $\ell = N_\mathrm{n}$. Durch die Dämpfung ist die Anregung ab einem gewissen Frequenzbereich nicht mehr hoch genug. Abbildung 3.4 zeigt die Fourier Koeffizienten über ℓ. Die Wahl der Taktzeit T_ns des erzeugten Rauschsignals, also die Zeit zwischen zwei Pulsen, be-

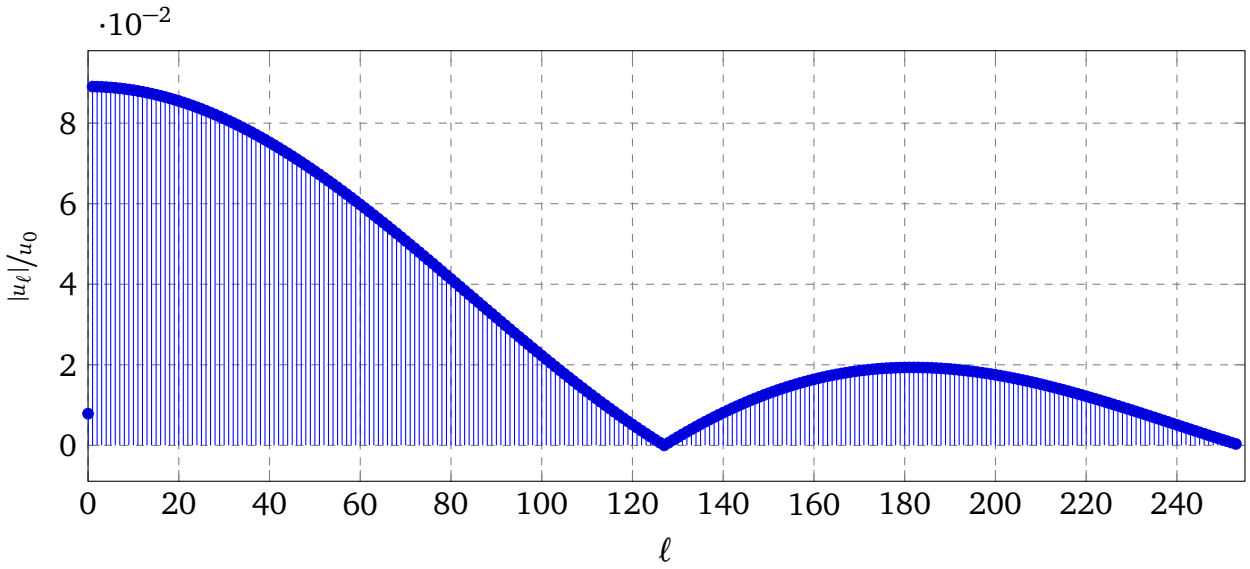


Abbildung 3.4: Fourier Koeffizienten, bei $\ell = N_\mathrm{n}$ wird der Koeffizient das erste mal 0

stimmt die höchste Frequenz f_max die gut angeregt wird. Einen Anhaltswert dafür liefert die Formel $T_\mathrm{ns} = 0,4/f_\mathrm{max}$. Die Frequenz f_max ist dann mit ungefähr $-3\,\mathrm{dB}$ gedämpft. Der verwendete Funktionsgenerator erwartet eine Sampling Rate. Diese ist gegeben durch $f_\mathrm{ns} = 1/T_\mathrm{ns} = 2.5 f_\mathrm{max}$.

Zusammen mit der Gesamtlänge N_n ergibt sich die Periodendauer des Signals zu $T_\mathrm{p} = N_\mathrm{n} T_\mathrm{ns}$ beziehungsweise die Frequenz zu $f_\mathrm{p} = 1/T_\mathrm{p}$. Wie in Kapitel 2 beschrieben, soll eine Übertragungsfunktion bestimmt werden. Dazu wird das Signal in den Frequenzbereich transformiert. Für den Fall von diskreten Daten im Zeitbereich, findet die Diskrete Fourier-Transformation Anwendung (DFT). Die Auflösung der DFT ist durch die Frequenz f_p bestimmt. Das Signal enthält nur vielfache der Frequenz f_p, folglich werden nur vielfache dieser Frequenz berechnet. Ist die Auflösung für die Anwendung nicht ausreichend fein, muss die Anzahl an Bits erhöht werden. Zusammenfassend lässt sich sagen:

- f_{ns} ist nach der Frequenz auszulegen, die noch gut angeregt werden soll und gegeben durch

$$f_{\mathrm{ns}} = 2.5 f_{\mathrm{max}}. \tag{8}$$

- Die Anzahl der Bits b ist nach der notwendigen Frequenzauflösung f_{p} auszulegen und ist bestimmt durch

$$b = \left\lceil \ln\left(2.5\frac{f_{\mathrm{max}}}{f_{\mathrm{p}}} + 1\right)\ln(2)^{-1} \right\rceil \in \mathbb{N}, \tag{9}$$

  mit der Aufrundungsfunktion $n = \lceil x \rceil$ definiert als $x \leq n < x + 1$ mit $n \in \mathbb{N}$.

Bei der Messung des MLBS Signals ist darauf zu achten, dass immer vollständige Perioden erfasst werden. Außerdem muss für die Berechnung der Fourierkoeffizienten aus einer Messung mit einer entsprechend hohen Frequenz abgetastet werden, sodass diese ausreichend genau berechnet werden können. Für $T_{\mathrm{s}} = T_{\mathrm{ns}}$, wobei T_{s} die Abtastzeit der Messung ist, liegt der Fehler der berechneten Koeffizienten in der Größenordnung der Referenzwerte. Es handelt sich trotzdem um eine sehr gute Näherung für weißes Rauschen. Bei $T_{\mathrm{s}} = 0,1 \cdot T_{\mathrm{ns}}$ befindet sich der Fehler im Bereich bis zu 10 % und bei $T_{\mathrm{s}} = 0,01 \cdot T_{\mathrm{ns}}$ bei 1 %. [12, 10, 14, 5]

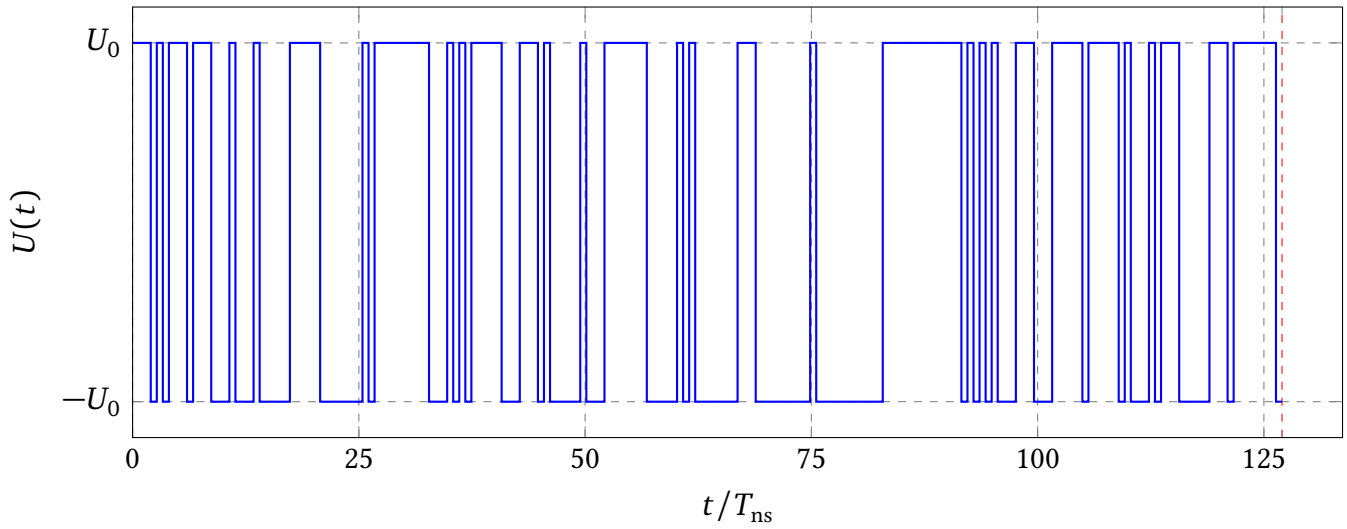


Abbildung 3.5: MLBS Signal für $n = 7$ bits mit dem Startregister $[1\,1\,0\,1\,0\,1\,1]$. Die rote Linie markiert das Ende der Periode bei $N = 127$.

3.2 Implementierung in Python

Da in der Abteilung Ring RF bereits mit Python gearbeitet wird, soll die hier entwickelte Software zur Bestimmung der Übertragungsfunktion ebenfalls als Python Code implementiert werden. Abbildung 3.7 zeigt den Code zur Erzeugung eines MLBS Signals. Dieser implementiert ein Schieberegister mit

wahlweise 6, 7, 8, 9 bzw. 10 Bit. Abbildung 3.5 zeigt das erzeugte Pseudo-Rauschsignal im Zeitbereich für das Startregister [1101011]. Es wird ein Signal mit 7 Bit und somit einer Länge von 127 erstellt. Zur Validierung des erzeugten Signals wird die Autokorrelation bestimmt. Abbildung 3.6 zeigt diese für eine Periode. Es ist zu erkennen, dass diese mit den theoretischen Erwartungen (Abbildung 3.1) übereinstimmt.

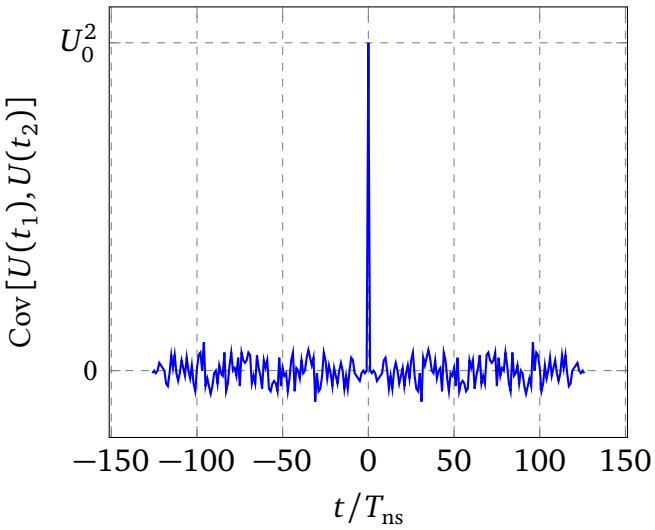


Abbildung 3.6: Autokorrelation einer Periode des Signals. Nur bei t=0, also dem unverschobenen Signal, ist eine Korrelation feststellbar.

```
import numpy as np
from scipy import stats
def get(bits=10):
    # ------- create random start register -------
    seedRandom = np.random.randint(2**31)
    np.random.seed(seed=579946590)
    xk = np.arange(2)
    pk = (0.5,0.5)
    custm = stats.rv_discrete(name='custm', values=(xk, pk))
    # ------- create Signal -------
    register = np.zeros(bits)
    while (np.sum(register)==0):
        register = np.array(custm.rvs(size=bits))
    N = 2**bits-1
    output = np.zeros(N)

    for i in range(0,N):
        output[i] = register[bits-1]
        if bits==6 or bits==7:
            r = np.logical_xor(register[0],register[bits-1])
        if bits==8:
            r = np.logical_xor(np.logical_xor(np.logical_xor(register[0],\
                               register[1]),register[6]),register[7])
        if bits==9:
            r = np.logical_xor(register[3],register[8])
        if bits==10:
            r = np.logical_xor(register[2],register[9])
        if bits==11:
            r = np.logical_xor(register[1],register[10])
        if r:
            r=1
        else:
            r=0
        register = np.append(r, register[0:bits-1])

    output = output - 0.5
    return (output, seedRandom)
```

Abbildung 3.7: MLBS Implementierung in Python

4 Algorithmus

Im folgenden Kapitel wird der erstellte Algorithmus und die Implementierung für die automatische Ermittlung der Übertragungsfunktion vorgestellt.

Die Implementierung des Pseudorauschens und die Auswertung der Übertragungsfunktion ist abhängig von den verwendeten Komponenten.

4.1 Software und Ansteuerung

Das Signal wird mit einem zweikanaligen Funktionsgenerator *Keysight 33600A Series Waveform Generator*, im Folgenden als AWG (*arbitary wave generator*) bezeichnet, erzeugt. Zum Messen wird ein digitales Oszilloskop vom Typ *Tektronix TDS 5054 Digital Phosphor Oscilloscope* verwendet. Das zu messende Zweitor wird mit $50\,\Omega$ Kabeln an AWG und Oszilloskop angeschlossen. Der AWG wird mittels einer USB-Verbindung angesteuert, das Oszilloskop über einen Switch per LAN-Verbindung. Beim Oszilloskop muss zusätzlich ein VXI-11-Server gestartet werden. Beide Geräte werden über das VISA Protokoll von National Instruments [13] programmiert und ausgewertet. Zur Implementierung des Algorithmus wird die open source Software Python 3.6 gewählt.[15] Abbildung 4.1 zeigt den schematischen Messaufbau, Abbildung 4.2 den realisierten. Die Pfeilrichtung zeigt die Richtung des Informations- und Datenflusses. AWG, PC und Oszilloskop tauschen jeweils Daten aus. Das AWG überträgt Daten an das Zweitor, das Daten an das Oszilloskop überträgt.

4.2 Implementierung des Algorithmus

Der Algorithmus kann in folgende vier Teile gegliedert werden:

- Erstellen des MLBS Signals
- Signalgenerator AWG programmieren und Signal auf Testgruppe ausgeben
- Oszilloskop programmieren und AWG Signal sowie Signal nach Testgruppe messen
- Übertragungsfunktion aus Messdaten erstellen

4.2.1 Erstellen des MLBS Signal

Das MLBS Signal kann mit der Implementierung für $b \in \{6, 7, 8, 9, 10\}$ erstellt werden. Damit das Startregister zufällig ist, die Ergebnisse aber reproduzierbar sind, wird in Python ein entsprechender

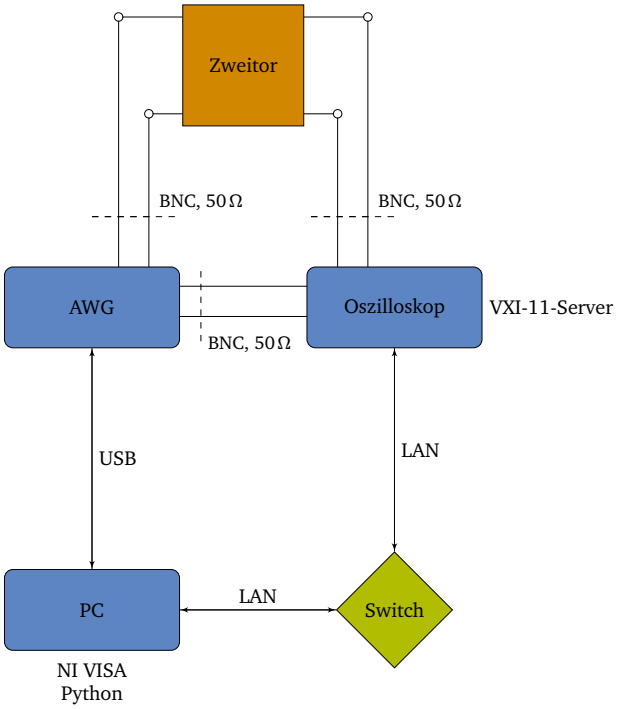


Abbildung 4.1: schematische Darstellung des Messaufbaus

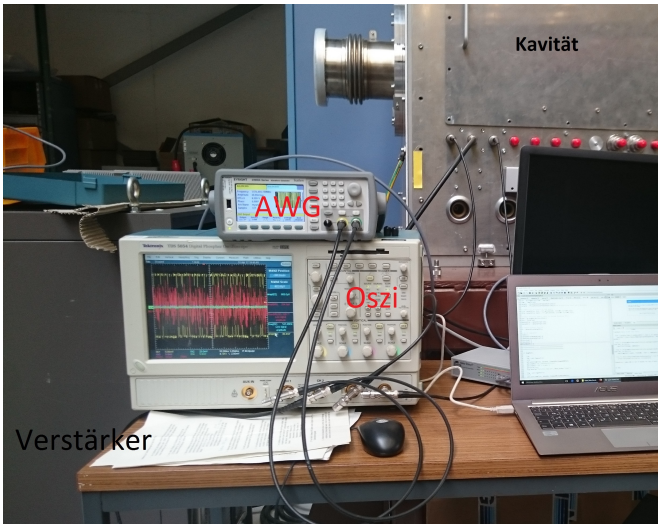


Abbildung 4.2: Messaufbau

seed key gesetzt. Der Zufallsgenerator für das Startregister wird mit diesem seed key initialisiert und erzeugt so für einen gleichen seed key die gleichen Zahlen für das Startregister.

4.2.2 Signalgenerator AWG programmieren

Das erstellte MLBS Signal wird für beide Kanäle des AWGs programmiert. Erst an dieser Stelle wird die Samplerate des Signals und damit auch die maximale Frequenz, die zuverlässig ausgemessen werden kann, festgelegt. Wie in Kapitel 3 erwähnt, beträgt die notwendige Frequenz $f_{\mathrm{ns}} = 1/T_{\mathrm{ns}} = 2,5 \cdot f_{\mathrm{max}}$. Die zwei verwendeten Kanäle werden zudem synchronisiert, damit die Messdaten in der Auswertung miteinander phasenfrei verrechnet werden können.

4.2.3 Oszilloskop programmieren

Die horizontale Achse des Oszilloskops wird so eingestellt, dass ungefähr $1,5$ Perioden des Signals sichtbar sind. Dies ist notwendig, da nur bestimmte Zeitskalierungen am Oszilloskop möglich sind und es sichergestellt werden muss, dass mindestens eine Periode sichtbar ist. In der Auswertung wird dann wieder auf eine Periode heruntergerechnet. Um das Ausmessen der Zeitsignale zum gleichen Zeitpunkt

garantieren zu können, wird das Oszilloskop in Stop geschaltet. Dann werden die Daten beider Kanäle übertragen. Für das Oszilloskop gilt die Beschränkung:

$$RecordLength = SampleRate \cdot HorizontalTimeScale \cdot 10 \tag{10}$$

Dabei ist *RecordLength* der benötigte Speicherplatz im Oszilloskop, *SampleRate* die Sample Rate des Oszilloskops und *HorizontalTimeScale* die Auflösung der horizontalen Achse des Oszilloskops. *HorizontalTimeScale* ist durch die Anforderung der Darstellung mindestens einer Periode festgelegt. Um Das Signal mit einem Fehler von ca. 1 % messen zu können, muss die Sample Rate des Oszilloskops $f_{\text{Oszi}} = 100 f_{\text{AWG}}$ sein (vgl. Kapitel 3). Für eine zu untersuchende Frequenz $f_{\text{max}} = 80\,\text{MHz}$ und einer Länge des MLBS Registers von $b = 9$ Bit werden die Parameter im Folgenden exemplarisch berechnet.

Für die Sample Rate des AWG ergibt sich

$$f_{\text{ns}} = 1/T_{\text{ns}} = 2,5 \cdot f_{\text{max}} = 2,5 \cdot 80\,\text{MHz} = 200\,\text{MHz} = f_{\text{AWG}} \tag{11}$$

und damit eine nötige Sample Rate im Oszilloskop von

$$f_{\text{Oszi}} = 100 \cdot f_{\text{AWG}} = 20\,\text{GSample/s}. \tag{12}$$

Um 1,5 Perioden des Signals darstellen zu können muss

$$HorizontalTimeScale = \frac{1,5 \cdot T_{\text{p}}}{10} = \frac{1,5 \cdot T_{\text{ns}} \cdot 2^b - 1}{10} = 383\,\text{ns} \tag{13}$$

sein, wobei T_{p} die Periodenlänge des Signals ist. Damit ergibt sich

$$RecordLength = 20\,\text{GSample/s} \cdot 383\,\text{ns} \cdot 10 = 76.600\,\text{Sample}. \tag{14}$$

Da das Oszilloskop nur feste Eingabewerte für diese Parameter akzeptiert, wird jeweils der nächst größere mögliche Wert gewählt. Damit ergeben sich folgende Werte:

- $HorizontalTimeScale = 400\,\text{ns}$
- $SampleRate = 25\,\text{GSample/s}$
- $RecordLength = 100.000\,\text{Sample}$

Diese Parameter werden automatisch von der Implementierung bestimmt.

4.2.4 Übertragungsfunktion aus Messdaten erstellen

Die Übertragungsfunktion wird am PC im Python Tool berechnet. Dazu werden zunächst die Messdaten des Eingangssignals und des Ausgangssignals über das VISA Protokoll vom Oszilloskop an den PC übertragen. Dort werden die Signale in Form der Messdaten auf eine Periode beschnitten und dann eine FFT (*Fast Fourier Transformation*) durchgeführt. Die Übertragungsfunktion bestimmt sich dann im Frequenzbereich nach $\underline{H}(\omega) = \underline{U}_{\text{out}}/\underline{U}_{\text{in}}$.

5 Ausmessen des Tiefpasses

Um die Funktionsweise des Pseudorauschens sowie die Implementierung des automatischen Ausmessens auf Richtigkeit zu überprüfen, wird zuerst ein einfaches Beispiel eingeführt. Abbildung 5.1 und 5.2 zeigen den für das Testen des Signals verwendeten Tiefpass sowie dessen Ersatzschaltbild.

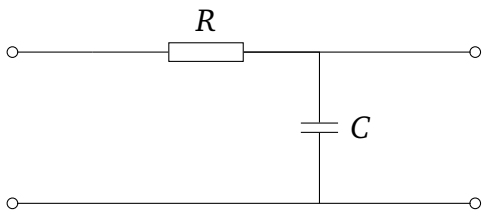


Abbildung 5.1: Ersatzschaltbild eines RC-Tiefpasses erster Ordnung

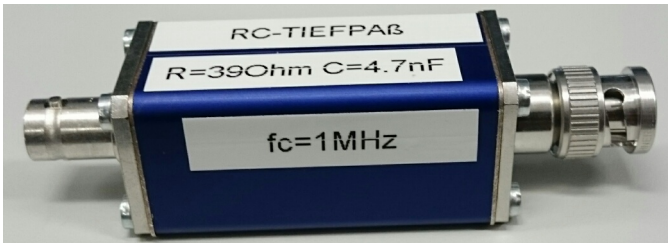


Abbildung 5.2: RC-Tiefpass erster Ordnung

5.1 Analytische Betrachtung des Tiefpasses

Für die Übertragungsfunktion des Tiefpasses gilt

$$\underline{H}(\omega) = \frac{\underline{U_{\text{out}}}(\omega)}{\underline{U_{\text{in}}}(\omega)} = \frac{1}{1 + j\omega CR} = \frac{1}{\sqrt{1 + \omega^2 C^2 R^2}} \mathrm{e}^{-j\phi}, \tag{15}$$

mit dem Phasenwinkel $\phi = -\arctan(\omega CR)$. Der gegebene Tiefpass wird durch die diskreten Bauelemente

$$R = 39\,\Omega \tag{16}$$

$$C = 4.7\,\text{nF} \tag{17}$$

bestimmt. Die Grenzfrequenz f_c ist bestimmt durch die Frequenz, bei der der Betrag der Übertragungsfunktion $H(\omega)$ um 3 dB vom Maximum abgefallen ist. Das Maximum der Übertragungsfunktion ist 1 für $\omega = 0$, somit gilt

$$|H(\omega)|_{f=f_\text{c}} = \frac{1}{\sqrt{2}}. \tag{18}$$

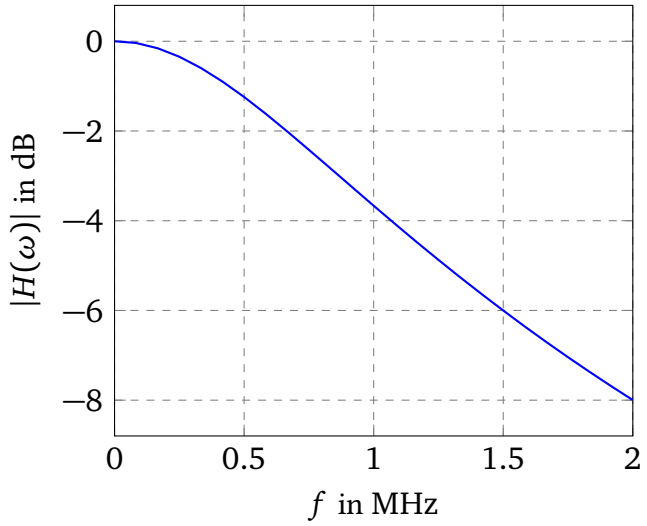


Abbildung 5.3: Übertragungsfunktion $|H(\omega)| = \left|\frac{U_{\text{out}}}{U_{\text{in}}}\right|$

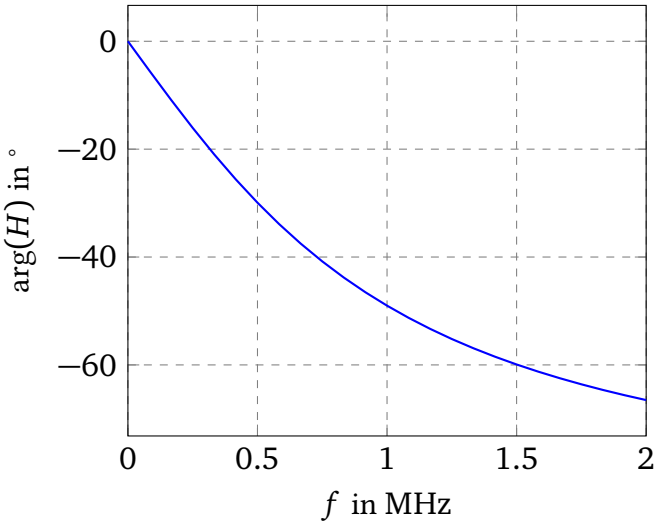


Abbildung 5.4: Phase $\phi = -\arctan(\omega CR)$ der Übertragungsfunktion H

Auflösen der Gleichung nach f_c liefert

$$f_c = \frac{1}{2\pi RC} \approx 0.868\,\text{MHz}. \tag{19}$$

Abbildung 5.3 zeigt den bekannten Verlauf der Übertragungsfunktion $|H(\omega)|$, Abbildung 5.4 entsprechend die Phase von $H(\omega)$. Der Tiefpass soll im Frequenzbereich von 0 bis 2 MHz untersucht werden.

5.2 Ausmessen des Tiefpasses

Der Tiefpass wurde mit dem Pseudorauschen ausgemessen. Kanal 1 des AWGs wurde mit dem Oszilloskop verbunden. Kanel 2 des AWGs wurde mit dem Tiefpass verbunden und dieser anschließend mit dem Oszilloskop. Um einen reflektionsfreihen Abschluss zu erhalten, wurden nur 50 Ω Leitungen verwendet und die Eingangsimpedanz des Oszilloskops ebenfalls auf 50 Ω festgelegt. Um die Übertragungsfunktion sicher bei 2 MHz auswerten zu können, wurden Frequenzen bis 5 MHz angeregt (siehe Kapitel 3). Abbildung 5.5 zeigt eine Periode des Pseudorauschens am Eingang des Tiefpasses, Abbildung 5.6 zeigt die Spannung am Ausgang des Tiefpasses.

Das Signal wird in den Frequenzbereich transformiert. Das transformierte Eingangssignal ist in Abbildung 5.7 im Bereich von 0 bis 8 MHz dargestellt. Bei $f = 5\,\text{MHz}$ wird der Fourier Koeffizient das erste Mal Null. Die starke Dämpfung ab ungefähr 2 MHz zeigt, dass eine Anregung von $f_{\text{AWG}} = 2.5 f_{\text{max}}$ notwendig ist.

Die ausgemessene Übertragungsfunktion zeigt die blaue Kurve in Abbildung 5.8. Die Grenzfrequenz lässt sich zu $f_c \approx 1.1\,\text{MHz}$ bestimmen. In grün dargestellt ist außerdem die Übertragungsfunktion, die

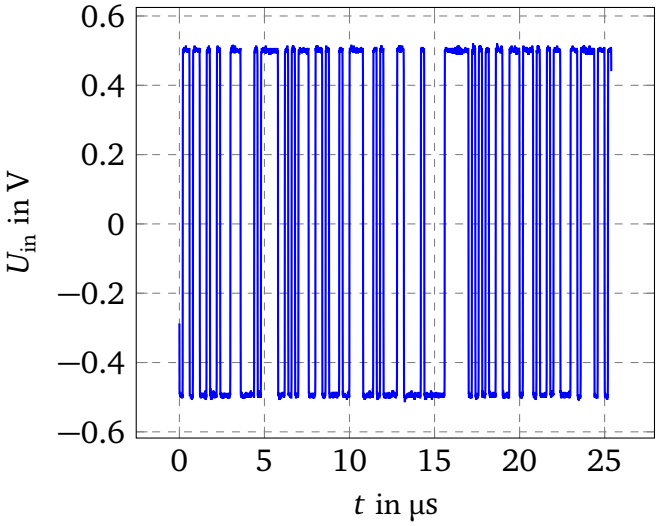


Abbildung 5.5: Eine Periode des Eingangssignals U_{in} am Tiefpass im Zeitbereich

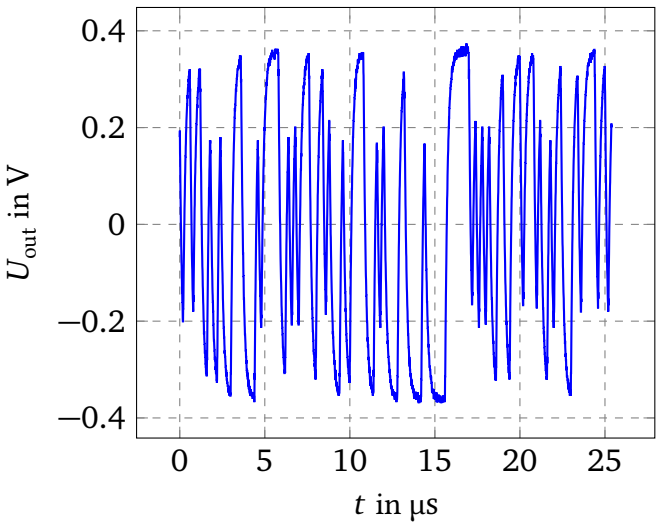


Abbildung 5.6: Eine Periode des Signals am Ausgang des Tiefpasses

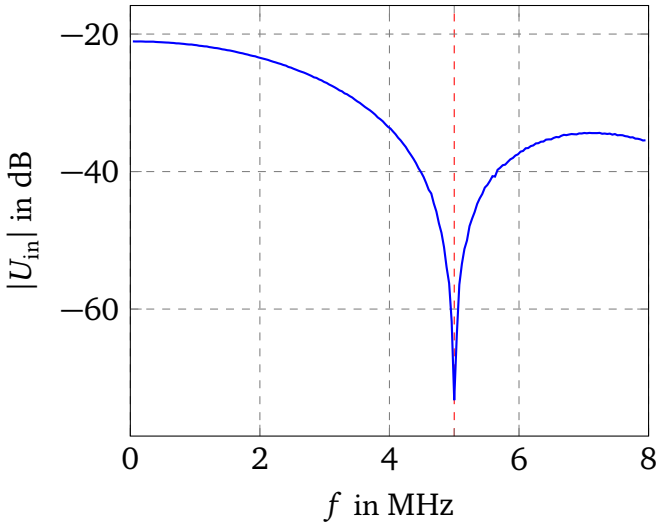


Abbildung 5.7: Eingangsspannung U_{in} im Frequenzbereich. Bei $f = 5\,\text{MHz} = f_{\text{max}}$ wird der Fourier Koeffizient das erste mal Null

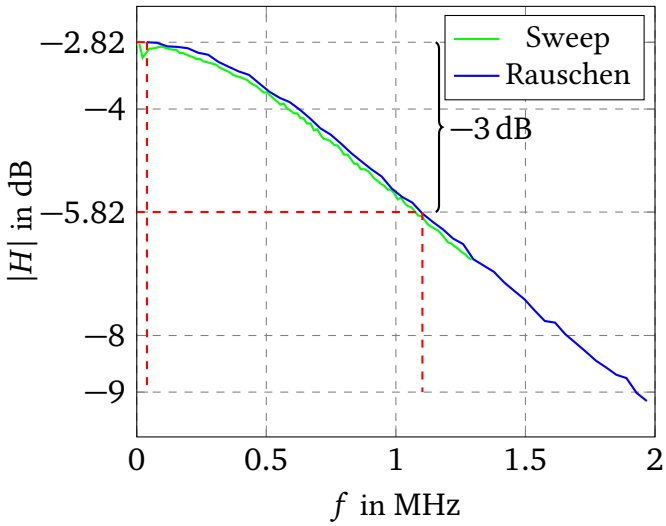


Abbildung 5.8: Übertragungsfunktion $H = \left|\frac{U_{\text{out}}}{U_{\text{in}}}\right|$ mit der −3 dB Grenzfrequenz bei $f \approx 1.1\,\text{MHz}$

sich ergibt, wenn der Tiefpass mittels Sweep ausgemessen wird. Beide Ergebnisse stimmen sehr gut überein. Das Bestimmen der Übertragungsfunktion mittels Pseudo-Rauschsignal ist also möglich.

Es wird angenommen, dass der Unterschied zwischen der gemessenen und berechneten Grenzfrequenz ist damit zu begründen, dass es sich um einen selbst gebauten Tiefpass handelt. Es wurden die genannten Komponenten verbaut, aber nicht berücksichtigt, dass kapazitive und induktive Effekte zum Gehäuse auftreten können. Daher stimmt die analytisch berechnete Grenzfrequenz nicht mit der gemessenen überein.

6 Bestimmung der Übertragungsfunktion des Gesamtsystems

Nachdem mit Hilfe des Tiefpasses die Funktionsfähigkeit der entwickelten Implementierung gezeigt wurde, wird diese im Folgenden auf das Kavitätensystem angewendet. Die gemessene Übertragungsfunktion wird dann mit Messergebnissen verglichen, die durch einen Sinus-Sweep erstellt wurden.

6.1 Messaufbau

Abbildung 6.1 zeigt den Messaufbau. Es werden die in Kapitel 4 vorgestellten Geräte (AWG und Oszilloskop) für die Messung verwendet. Die Kommunikation zwischen PC und AWG bzw. Oszilloskop bleibt unverändert.

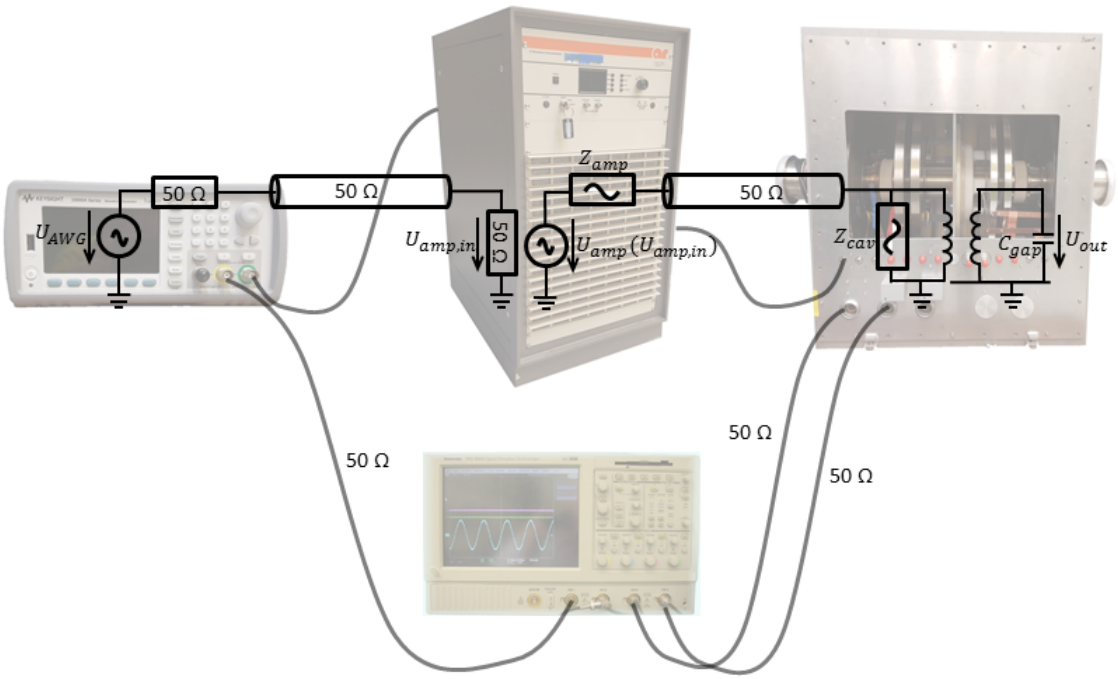
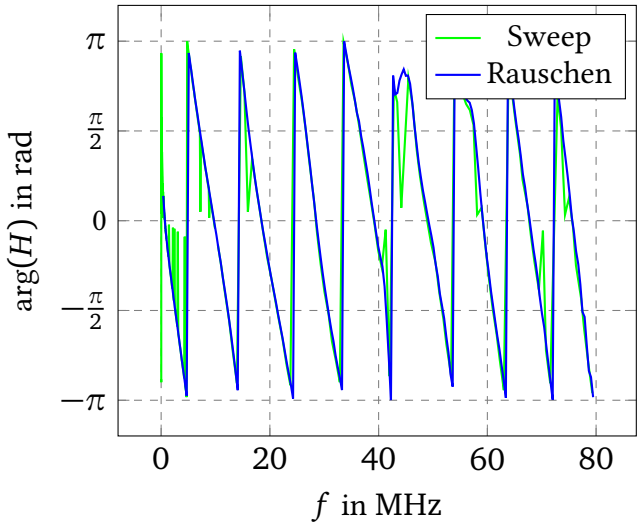


Abbildung 6.1: Messaufbau vgl. [8]

Die hohen verwendeten Frequenzen erfordern ein reflektionsfreies abschließen der Netzwerke. Um ein unangepasstes Netzwerk zu vermeiden, wird das Pseudorauschen sowohl an das auszumessende System als auch direkt an das Oszillkop angelegt. Das Ausgangssignal des Systems wird im Oszilloskop mathematisch bestimmt. Am dritten und vierten Kanal des Oszilloskops wird die Spannung je einer



h!]

Abbildung 6.2: Phase der Übertragungsfunktion des Kavitätensystems

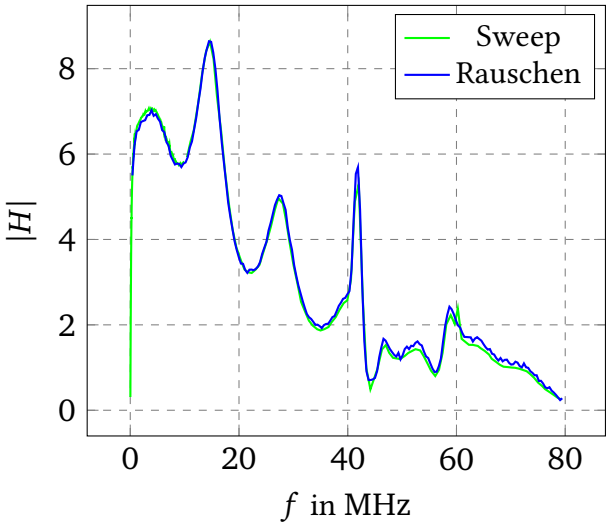


Abbildung 6.3: Übertragungsfunktion des Kavitätensystems

Gaphälfte gemessen und dann deren Differenz im Oszilloskop gebildet. Alle Oszilloskop-Eingänge sind hochohmig abgeschlossen. Die verwendeten Koaxialleitungen sind mit 50 Ω behaftet. Die Koaxialkabel sind am hochohmig abgeschlossenen Oszilloskop mit 50 Ω BNC Abschlusswiderständen versehen, um das Netzwerk reflektionsfrei abzuschließen.

6.2 Messung der Übertragungsfunktion

Zur Messung der Übertragungsfunktion muss die Python Routine *runme.py* gestartet werden. Dabei können verschiedene Parameter vorgegeben werden:

- maximale zu untersuchende Frequenz
- Spitze-Spitze Spannung des AWG Signals
- Anzahl der Bits des MLBS Signals

Das Programm speichert automatisch Eingangs- und Ausgangssignal sowie Übertragungsfunktion als .pdf- und .csv-Datei. Eine detaillierte Anleitung zur Durchführung der Messung ist im Anhang 8 zu finden.

6.3 Vergleich Übertragungsfunktion

Abbildung 6.2 und Abbildung 6.3 zeigen Phase und Amplitude der Übertragungsfunktionen, gemessen mit beiden Messmethoden. Es wird ein MLBS Signal mit 10 Bit verwendet und die Übertragungsfunktion bis 80 MHz untersucht. Die grüne Kurve wurde mittels Pseudo-Rauschsignal bestimmt, die blaue mit einem Sweep gemessen. Die Übertragungsfunktionen stimmen sehr gut überein. Dabei ist zu erkennen, dass sich die Übereinstimmung im oberen Frequenzbereich verschlechtert. Dies kann daran

liegen, dass das Pseudo-Rauschsignal Frequenzen in höheren Bereichen weniger stark anregt (vgl. Kapitel 3). Bei $f_{\max} = 80\,\text{MHz}$ wird die ANregung um ca. 3 dB gedämpft. Die mit dem Sweep gemessene Phase weist Ausreißer durch Messfehler auf. Diese treten bei einer Messung mittels Rauschen nicht auf.

Die Aufnahme der Übertragungsfunktion dauert bei der Verwendung des Sweeps $3 \cdot 500\,\text{s}$. Wird diese mit Hilfe von Rauschen bestimmt, benötigt eine Messung nur 42 s.

6.4 Vergleich Registerlänge

Für die Erstellung des MLBS Signals können verschiedene Registerlängen gewählt werden. Abbildung 6.4 und Abbildung 6.5 zeigen Phase und Amplitude der Übertragungsfunktion des Kavitätensystems. Dabei wurde die Länge des MLBS Registers zwischen 7 Bit und 10 Bit variiert.

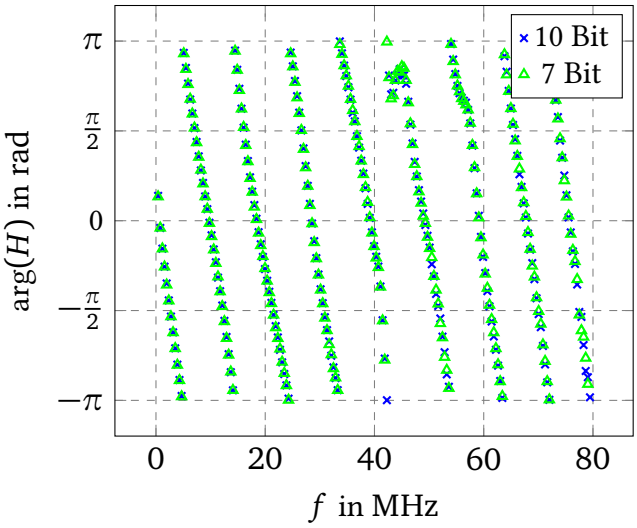


Abbildung 6.4: Vergleich Phase der Übertragungsfunktion des Kavitätensystems

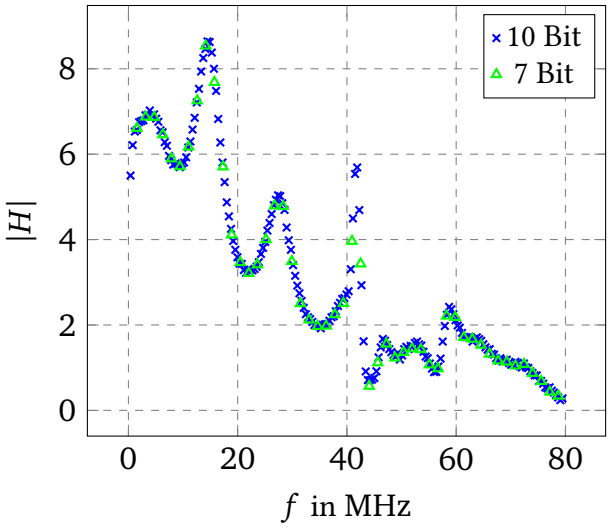


Abbildung 6.5: Vergleich Übertragungsfunktion des Kavitätensystems

Es ist zu erkennen, dass die Auflösung der Übertragungsfunktion bei 10 Bit besser ist als bei 7 Bit. Vor allem im Bereich von lokalen Maxima fehlen bei 7 Bit Frequenzpunkte, sodass diese Maxima nicht dargestellt werden. Abbildung 6.2 und Abbildung 6.3 haben gezeigt, dass die Auflösung bei 10 Bit alle Maxima und Minima ausreichend auflöst. Eine weitere Erhöhung der Registerlänge ist nicht möglich, da das zur Verfügung stehende AWG bei mehr als 10 Bit an die Grenze seines Speichers gelangt.

6.5 Untersuchung der Hochlaufzeit

Als erste Anwendung des neuen Verfahrens wird untersucht, wie und ob sich die Übertragungsfunktion mit der Zeit beim Einschalten des Systems verändert. Besonders dem Verstärker als aktives Bauelement wird eine Hochlaufzeit unterstellt. Dazu wurde die Übertragungsfunktion automatisch zu verschiede-

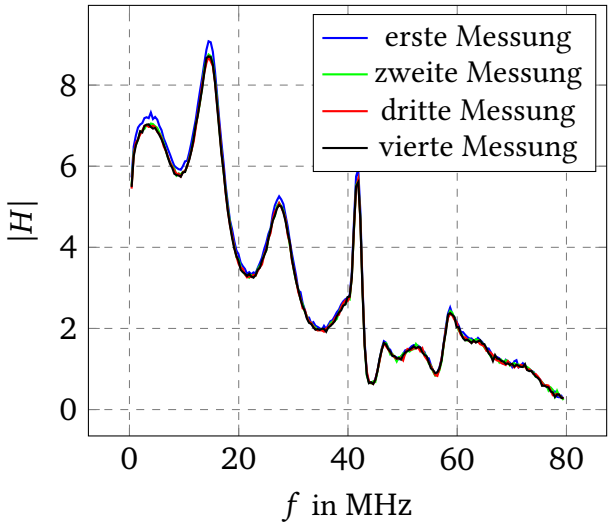


Abbildung 6.6: Übertragungsfunktion gemessen im Abstand von 4 Minuten

nen Zeitpunkten gemessen. Die erste Messung wurde direkt nach dem Einschalten der Verstärkers aufgenommen. Hier wurde auch das AWG mit dem MLBS Signal programmiert. Die weiteren Messungen finden in einem Abstand von ungefähr 35 s statt, wobei lediglich die Messdaten des Oszilloskops abgerufen werden. Abbildung 6.6 zeigt die Übertragungsfunktion des Systems zu verschiedenen Zeitpunkten mit einem Abstand von ungefähr 4 min. Zwischen der ersten und den weiteren Messungen ist ein Unterschied vor allem in den Extremstellen erkennbar. Danach verändert sich die Übertragungsfunktion nicht mehr. Eine Analyse der Messergbenisse zeigt, dass nach ungefähr 4 min das System eingeschwungen ist und sich nicht mehr maßgeblich verändert.

7 Zusammenfassung und Ausblick

Im Rahmen des vorliegenden Projektseminars wurde ein Python Code zur Ermittlung der Übertragungsfunktion eines Kavitätensystems mittels Pseudorauschen implementiert.

Nach einer Einführung über Pseudo-Rauschsignale und deren Generierung am Digitalrechner wurde eine Implementierung in Python vorgestellt. Diese verwendet eine *maximum length binary sequence* (MLBS) zur Erstellung des Rauschens.

Anschließend wurde die erstellte Implementierung zur Verifizierung der Funktionalität an einem Tiefpass getestet. Die gemessene Übertragungsfunktion des Tiefpasses stimmt mit dem bekannten Verlauf dieses Bauelements überein.

Nach der Verifizierung wurde das Python Tool auf das zu untersuchende Kavitätensystem angewendet und so dessen Übertragungsfunktion gemessen. Die Ergebnisse stimmen mit vorliegenden Messergebnissen eines Sinus Sweeps bis überein. Speziell für diese Anwendung ist noch zu überprüfen, ob die durch den Sinus Sweep ermittelte Übertragungsfunktion oder die durch das Pseudo Rauschen ermittelte besser geeignet ist, um einen Einzelsinus am Ausgang der Kavität zu erzeugen. Wegen der guten Übereinstimmung der Messergebnisse ist aber zu erwarten, dass die erzeugten Einzelsinus-Signale sehr ähnlich sind.

Außerdem wurde der Einfluss der Registerlänge des MLBS und die Veränderung der Übertragungsfunktion nach dem Einschalten des Systems untersucht. Durch ein längeres MLBS Register verbessert sich die Auflösung der Übertragungsfunktion. Länger als 10 Bit konnte dieses aber nicht gewählt werden, da der Speicher des Signalgenerators nicht ausreicht. Nach dem Einschalten des Systems verändert sich die Übertragungsfunktion noch für ca. 4 Minuten. Danach ist keine Änderung mehr zu verzeichnen.

Eine zeitliche Optimierung des Python Codes wurde nur bedingt durchgeführt. Das Senden und Auslesen der Daten ist im Programmcode immer mit einer Pause behaftet. Diese Zeiten lassen sich noch optimieren.

Das Oszilloskop kann für die Parameter $RecordLength$,$SampleRate$ sowie $HorizontalTimeScale$ nur diskrete Werte annehmen. Diese sollten im Code hinterlegt und so benutzt werden, dass die errechneten Parameter mindestens erfüllt sind. Für die $RecordLength$ wurde das bereits erledigt.

Die Entwicklung einer graphischen Oberfläche würde die Bedienbarkeit, das Einstellen der Parameter sowie die Beurteilung der Ergebnisse verbessern. Zudem muss der Programmcode noch dahingehend verändert werden, dass er auf Fehleingaben des Benutzers reagieren kann. Dazu gehören Eingaben, die außerhalb der Spezifikation der Geräte liegen, Schutzmaßnahmen für nachgeschaltete Geräte und nicht programmierte Parameter.

Um den Einfluss von Zufallsgrößen gegeben durch Messunsicherheiten, Messumfeld etc. zu minimieren, sollte eine Mittelwertbildung mehrerer Übertragungsfunktionen erfolgen.

8 Appendix

8.1 Bedienungsanleitung

Im Folgenden werden wichtige Bedienelemente des Programmcods, sowie der Messgeräte erklärt.

8.1.1 Eigener Laptop

- Installiere VISA [13]
- Füge der Windows-Firewall eine Regel hinzu:

  Start → Systemsteuerung → Windows-Firewall ↴

  ↳ links “erweiterte Einstellungen” → Eingehende Regeln ↴

  ↳ rechts “neue Regel” → “benutzerdefiniert” →weiter→ alle Programme →weiter ↴

  ↳ bei Ports keine Einstellungen vornehmen→weiter ↴

  ↳ für welche Remote-IP-Adressen gilt diese Regel: diese IP-Adressen: 169.254.225.181 hinzufügen↴

  ↳ weiter→ “Verbindung zulassen” →weiter→Domäne. Privat und Öffentlich anhaken→weiter↴

  ↳ Fertig stellen.
- Verbinde Notebook und Oszi mit zwei LAN-Kabeln und einem Switch oder Hub dazwischen
- Verbinde Notebook und AWG mit einem USB Kabel

8.1.2 Oszilloskop

- Oszi einschalten
- Minimiere mit File → Minimize die Mess-Software auf dem Oszi, so dass du die Windows-Oberfläche siehst (Win 2000)
- Gegebenenfalls IP-Adresse des Oszis auf 169.254.225.181 mit Subnetzmaske 255.255.0.0 kontrollieren
- Rechtsklick auf das Symbol mit dem roten Kreis rechts unten neben der Uhrzeit → Start VXI-11 Server
- Messsoftware wieder maximieren

8.1.3 Arbeiten mit dem Python Code

Um die Übertragungsfunkrion mit dem Python Code zu bestimmen, ist lediglich eine Funktion aufzurufen. Die Parameter dieser Funktion werden im Folgenden erklärt.

```
def compute(fmax, Vpp, bits=9, writeAWG=True, showPlots=True,
            createCSV=True, formatOutput=1):
```

Zwingend notwendig als Eingabe sind nur die ersten beiden Parameter fmax und Vpp. Die restlichen Parameter sind optional.

- fmax: Beschreibt die Frequenz die noch ausreichend gut angeregt werden soll
- Vpp: Spitze-Spitze Spannung am Ausgang des AWGs
- bit: Anzahl an Bit die zur Erzeugung des MLBS Signals verwendet werden soll. Mehr Bit bedeutet ein längeres Signal, womit die Auflösung der DFT steigt. Es kann zwischen 6, 7, 8, 9 und 10 Bit gewählt werden. Default: bit=9
- writeAWG: Falls True wird das AWG mit einem neuen, zufällig generiertem MLBS Signal beschrieben. Default: writeAWG=True
- showPlots: Falls True werden Plots der Ein- und Ausgangsdaten, sowie der Übertragungsfunktion angezeigt und als .pdf gespeichert. Default: showPlots=True
- createCSV: Falls True werden .csv Dateien der Ein- und Ausgangsdaten, sowie der Übertragungsfunktion erzeugt. Default: createCSV=True
- formatOutput: Bestimmt das Format der ausgegebenen Plots und .csv Dateien. Für formatOutput=0 werden alle Ausgangsdaten in dB gespeichert. Für formatOutput=1 werden alle Ausgangsdaten linear gespeichert. Für formatOutput=2 wird sowohl linear als auch in dB gespeichert. Default: formatOutput=1

Als nächstes wird der Aufruf der Funktion `compute` erläutert. Dazu öffnen wir die Datei “runme.py”. Wir betrachten zwei Beispiele.

1. Dieses Beispiel dient als Minimalbeispiel. Die Frequenz die noch ausreichend gut angeregt werden soll, liegt hier bei 80 MHz. Die Spitze-Spitze Spannung bei 40 mV. Der Rest der Parameter enstpricht den Default Werten.

   ```
   import getH
   getH.compute(80e6,40e-3)
   ```

2. Dieses Beispiel dient als Maximalbeispiel. Frequenz und Spannung sind identisch zu Beispiel 1. Die Anzahl der verwendeten Bits liegt bei 10. Die DFT wird also hochauflösender. Es werden keine Plots angezeigt oder gespeichert. Die .csv Dateien werden sowohl linear als auch in dB gespeichert.

```
import getH
getH.compute(80e6,40e-3, 10, True, False, True, 2)
```

8.2 Python Code

8.2.1 getH.py

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Tue Feb 21 13:10:28 2017

@author: Armin Galetzka, Denys Bast
Measures and calculates the transfer function of a two port.
Requirements:
    - AWG with two output ports
    - Channel 1 at oscilloscope is output signal of AWG
    - Channnel 3 and 4 are connected to the output of the two port and
      the quantity of interest is calculated by CH3-CH4


Input:  fmax          ------  max frequency of interest
        Vpp           ------  Output peac-peac voltage of AWG
        showPlots     ------  If True plots are shown and saved as .pdf
        createCSV     ------  If True a CSV file for each quantity of interest
                              is created
        formatOutput  ------  0=dB, 1=linear, 2=both
"""

def compute(fmax, Vpp, bits=10, writeAWG=True, showPlots=True, createCSV=True, \
            formatOutput=1):

    import visa
    import MLBS
    import time
    import matplotlib.pyplot as plt
    import numpy as np
    import FFT
    import csv
    import os


    # Create folder for results
    directory = time.strftime("%d.%m.%Y_%H_%M_%S")
    if not os.path.exists(directory):
        os.makedirs(directory)
    if not os.path.exists(directory + "/Plots"):
        os.makedirs(directory + "/Plots")
    if not os.path.exists(directory + "/csv"):
        os.makedirs(directory + "/csv")
    # Parameter
    awg_volt = Vpp
    samplerateAWG = 2.5*fmax
    samplerateOszi = 100*samplerateAWG
    fPlot = fmax
    possibleRecordLength = [500,2500,5000,10e3,25e3,50e3,100e3,250e3,500e3]
    possibleRecordLength = np.array(possibleRecordLength)
    linewidthPlot = 1

    font = {'family' : 'normal',
            'weight' : 'normal',
            'size'   : 12}

    plt.rc('font', **font)

    # Connect to Instruments
    rm = visa.ResourceManager()
    rs = rm.list_resources()
```

```
awg_id = rs[0]
AWG = rm.open_resource(awg_id)
# am Desktop PC
# dso_ip = rs[1]
# am Gruppenlaptop BTNBG006
dso_ip = 'TCPIP::169.254.225.181::gpib0,1::INSTR'
DSO = visa.ResourceManager().get_instrument(dso_ip)

[signal, seed] = MLBS.get(bits)
Tns = 0.4/fmax
periodTime = signal.size*Tns
horizontalScalePerDiv = 1.5*periodTime/10 #At least one period needs to be
                                          #shown on the DSO

##################################################
################## Write to AWG ##################
##################################################
if writeAWG:
    AWG.write("*RST")
    AWG.write("SOURce1:FUNCtion:ARBitrary:FILTer OFF")
    AWG.write("SOURce2:FUNCtion:ARBitrary:FILTer OFF")
    #time.sleep(5)
    AWG.write("DATA:VOLatile:CLEar")
    #time.sleep(5)
    myrange=max(abs(max(signal)),abs(min(signal)))
     #Data Conversion from V to DAC levels
    data_conv = np.round(signal*32766/myrange);
    data_conv = ",".join(str(e) for e in data_conv)
    AWG.write("SOURce1:DATA:ARBitrary:DAC myarb ," + data_conv)
    AWG.write("SOURce1:FUNCtion:ARBitrary 'myarb'")
    time.sleep(10)
    AWG.write("SOURce1:FUNCtion ARB") #USER
    AWG.write("DISPlay:FOCus CH1")
    AWG.write("DISPlay:UNIT:ARBRate FREQuency")
    AWG.write("SOURce1:FUNCtion:ARBitrary:SRATe " + str(samplerateAWG))
    AWG.write("SOURce2:DATA:ARBitrary:DAC myarb ," + data_conv)
    AWG.write("SOURce2:FUNCtion:ARBitrary 'myarb'")
    time.sleep(10)
    AWG.write("SOURce2:FUNCtion ARB") #USER
    AWG.write("DISPlay:FOCus CH2")
    AWG.write("DISPlay:UNIT:ARBRate FREQuency")
    AWG.write("SOURce2:FUNCtion:ARBitrary:SRATe " + str(samplerateAWG))
    AWG.write("FUNC:ARB:SYNC")
    AWG.write("SOURce1:VOLTage " + str(awg_volt))
    AWG.write("SOURce2:VOLTage " + str(awg_volt))
    time.sleep(5)
    AWG.write("OUTPut1 ON")
    AWG.write("OUTPut2 ON")
    AWG.write("DISPlay:FOCus CH1")

##################################################
################## Write to DSO ##################
##################################################

DSO.write("*RST") #Restores the state of the instrument from a copy of
                  #the settings stored in memory
DSO.write("ACQUIRE:STATE OFF") #This command stops acquisitions
DSO.write("SELECT:CH1 ON") #Turns the channel 1 waveform display on, and
                           #selects channel 1.
DSO.write("MATH3:DEFIne \"CH3-CH4\"") #Defines MATH function
DSO.write("SELECT:MATH3 ON") #Turns MATH3 display on
```

```
DSO.write("MATH1:DEFIne \"CH1\"") #Defines MATH function. CH1 is copied
                                  #to MATH1, because output format of
                                  #MATH1 is easier to handle
DSO.write("SELECT:MATH1 ON") #Turns MATH1 display on
DSO.write("TRIGger:A:EDGE:SOUrce CH1") #This command sets or queries the
                                       #source for the A edge trigger.
DSO.write("TRIGger:A:EDGE:SLOpe FALL") #This command sets or queries the
                                       #slope for the A edge trigger.

DSO.write("HORizontal:MAIn:SCAle " + str(horizontalScalePerDiv)) #Sets the
#time per division for the time base
# Here 1,5 periods are on screen. Necessary since Osci has only discrete
# values for horizontal scale and it needs to be ensured that at least
# one full period is in the screen
horizontalScalePerDiv = DSO.query("HORizontal:MAIn:SCAle?")
horizontalScalePerDiv = [float(s) for s
                         in horizontalScalePerDiv.split(',')]
horizontalScalePerDiv = horizontalScalePerDiv[0]
recordLength = horizontalScalePerDiv*10*samplerateOszi
ind = np.argmin(np.abs(recordLength - possibleRecordLength))
if possibleRecordLength[ind] < recordLength and \
(ind+1)<possibleRecordLength.size:
    recordLength = possibleRecordLength[ind+1]
else:
    recordLength = possibleRecordLength[ind]
DSO.write("HORIZONTAL:RECOrdlength " + str(recordLength)) #1e5
DSO.write("CH1:SCAle " + str(awg_volt/6)) #Sets the vertical scale
DSO.write("MATH1:SCAle " + str(awg_volt/6)) #Sets the vertical scale
DSO.write("CH2:SCAle 20.0E-3") #Sets the vertical scale
DSO.write("CH3:SCAle 50.0E-3") #Sets the vertical scale
DSO.write("CH4:SCAle 50.0E-3") #Sets the vertical scale
DSO.write("MATH3:SCAle 200.0E-3") #Sets the vertical scale
DSO.write("CH1:POSition 0") #Sets the horizontal scale
DSO.write("MATH3:POSition 0") #Sets the horizontal scale
DSO.write("MATH1:POSition 0") #Sets the horizontal scale
DSO.write("CH1:TERmination 1.0E+6") #Sets the termination of the channel
DSO.write("CH2:TERmination 1.0E+6") #Sets the termination of the channel
DSO.write("CH3:TERmination 1.0E+6") #Sets the termination of the channel
DSO.write("CH4:TERmination 1.0E+6") #Sets the termination of the channel
DSO.write("CH1:COUPling DC")  #Sets the coupling of channel 1 to AC
# Coupling to AC since the input signal has no DC component.
# No DC expected at the output. Use AC coupling to reduce influence
# from outside.
DSO.write("DATa:SOUrce MATH1")   #This command sets the location of
                                 #waveform data that is transferred from the
                                 #instrument by the CURVe? Query
DSO.write("DATa:ENCdg ASCIi") #This command sets the format of outgoing
                              #waveform data to ASCII
DSO.write("ACQUIRE:MODE SAMPLE") #This command sets the acquisition mode
                                 #of the instrument to sample mode
DSO.write("ACQUIRE:STOPAFTER SEQUENCE") #Specifies that the next
                                        #acquisition will be a
                                        #single-sequence acquisition.
DSO.write("HORizontal:MAIn:SAMPLERate " + str(samplerateOszi)) # Sets the
                                        # sample rate of the device.
                                        # Here: 10 times maximum expected
                                        # frequency to reduce aliasing
DSO.write("ACQUIRE:STATE ON") #This command starts acquisitions
DSO.write("DATa:STARt 1") #This command sets the starting data point
                #for waveform transfer. This command allows for the
                #transfer of partial waveforms to and from the instrument.
```

```
DSO.write("DATa:STOP " + DSO.query("HORIZONTAL:RECOrdlength?")) #Sets the
   #last data point that will be transferred when using the CURVe? query
time.sleep(5)
dataUin = DSO.query("CURVe?")
DSO.write("DATa:SOUrce MATH3")   #This command sets the location of
                                 #waveform data that is transferred from the
                                 #instrument by the CURVe? Query
DSO.write("DATa:ENCdg ASCIi") #This command sets the format of outgoing
                              #waveform data to ASCII
DSO.write("DATa:STARt 1") #This command sets the starting data point
                #for waveform transfer. This command allows for the
                #transfer of partial waveforms to and from the instrument.
DSO.write("DATa:STOP " + DSO.query("HORIZONTAL:RECOrdlength?")) #Sets the
     #Last data point that will be transferred when using the CURVe? query
time.sleep(5)
dataUout = DSO.query("CURVe?")

recordLength = DSO.query("HORIZONTAL:RECOrdlength?")
horizontalScalePerDiv = DSO.query("HORizontal:MAIn:SCAle?")
YScalePerDivUin = DSO.query("MATH1:SCAle?")
YScalePerDivUout = DSO.query("MATH3:SCAle?")

##################################################
############## Compute transfer function #########
##################################################

# Change format of data from DSO
dataUin = [float(s) for s in dataUin.split(',')]

dataUout = [float(s) for s in dataUout.split(',')]
dataUin = np.array(dataUin)
dataUout = np.array(dataUout)
recordLength = [float(s) for s in recordLength.split(',')]
recordLength = recordLength[0]
horizontalScalePerDiv = [float(s) for s
                         in horizontalScalePerDiv.split(',')]
horizontalScalePerDiv = horizontalScalePerDiv[0]
YScalePerDivUin = [float(s) for s in YScalePerDivUin.split(',')]
YScalePerDivUin = YScalePerDivUin[0]
YScalePerDivUout = [float(s) for s in YScalePerDivUout.split(',')]
YScalePerDivUout = YScalePerDivUout[0]

# Get time vector
dt = 10*horizontalScalePerDiv/recordLength
time = np.arange(0,10*horizontalScalePerDiv,dt)

# Reduce time vector and signal to one period
tmpTime = periodTime - time[0]
ind = np.argmin(abs(time - tmpTime)) #find next index
time = time[0:ind]
dataUin = dataUin[0:ind]
dataUout = dataUout[0:ind]

# Compute FFT of signals in time domain
[frq, UinAmpl, PhaseUin, Uin] = FFT.get(dataUin, 1/(time[-1]-time[-2]));
[frq, UoutAmpl, PhaseUout, Uout] = FFT.get(dataUout, \
                                           1/(time[-1]-time[-2]));

# Reduce frequency domain signal to maximum frequency fPlot
ind = np.argmin(abs(frq-fPlot))
frq = frq[0:ind]
```

```
UinAmpl = UinAmpl[0:ind]
UoutAmpl = UoutAmpl[0:ind]
Uin=Uin[0:ind]
Uout=Uout[0:ind]
PhaseUout=PhaseUout[0:ind]
PhaseUin=PhaseUin[0:ind]

# Compute transer function
H = UoutAmpl/UinAmpl
PhaseH=np.angle(Uout/Uin)
#PhaseH = PhaseUout-PhaseUin
# No DC component!
H=H[1:]
UinAmpl=UinAmpl[1:]
UoutAmpl=UoutAmpl[1:]
frq=frq[1:]
Uin=Uin[1:]
Uout=Uout[1:]
PhaseH=PhaseH[1:]
##################################################
############## Plots ##############################
##################################################

if showPlots:

    f=0
    fig = plt.figure(f+1)
    f+=1
    plt.plot(time*1e6, dataUin, linewidth=linewidthPlot)
    plt.ylabel(r'$U_{\mathrm{in}}(t)$')
    plt.xlabel(r'$t$ in $\mu$s')
    plt.grid(True)
    fig.savefig(directory + "/Plots/Uin_time.pdf", bbox_inches='tight')
    plt.show()

    fig = plt.figure(f+1)
    f+=1
    plt.plot(time*1e6, dataUout, linewidth=linewidthPlot)
    plt.ylabel(r'$U_{\mathrm{out}}(t)$')
    plt.xlabel(r'$t$ in $\mu$s')
    plt.grid(True)
    fig.savefig(directory + "/Plots/Uout_time.pdf", bbox_inches='tight')
    plt.show()

    fig = plt.figure(f+1)
    f+=1
    plt.plot(frq/1e6, PhaseH, linewidth=linewidthPlot)
    plt.ylabel(r'arg($H(\omega)$)')
    plt.xlabel(r'$f$ in MHz')
    plt.grid(True)
    fig.savefig(directory + "/Plots/PhaseH.pdf", bbox_inches='tight')
    plt.show()

    if (formatOutput==0) or (formatOutput==2):
        fig = plt.figure(f+1)
        f+=1
        plt.plot(frq/1e6, 20*np.log10(UinAmpl), linewidth=linewidthPlot)
        plt.grid(True)
        plt.ylabel(r'$|U_{\mathrm{in}}(f)|$ in dB')
        plt.xlabel(r'$f$ in MHz')
        fig.savefig(directory + "/Plots/UinAmpl_frq_dB.pdf",\
```

```
                    bbox_inches='tight')
        plt.show()

        fig = plt.figure(f+1)
        f+=1
        plt.plot(frq/1e6, 20*np.log10(UoutAmpl), linewidth=linewidthPlot)
        plt.grid(True)
        plt.ylabel(r'$|U_{\mathrm{out}}(f)|$ in dB')
        plt.xlabel(r'$f$ in MHz')
        fig.savefig(directory + "/Plots/UoutAmpl_frq_dB.pdf",\
                    bbox_inches='tight')
        plt.show()

        fig = plt.figure(f+1)
        f+=1
        plt.plot(frq/1e6, 20*np.log10(H), linewidth=linewidthPlot)
        plt.grid(True)
        plt.ylabel(r'$|H(f)|$ in dB')
        plt.xlabel(r'$f$ in MHz')
        fig.savefig(directory + "/Plots/H_dB.pdf", bbox_inches='tight')
        plt.show()

    if (formatOutput==1 or formatOutput==2):
        fig = plt.figure(f+1)
        f+=1
        plt.plot(frq/1e6, UinAmpl, linewidth=linewidthPlot)
        plt.grid(True)
        plt.ylabel(r'$|U_{\mathrm{in}}(f)|$')
        plt.xlabel(r'$f$ in MHz')
        fig.savefig(directory + "/Plots/UinAmpl_frq_linear.pdf",\
                    bbox_inches='tight')
        plt.show()

        fig = plt.figure(f+1)
        f+=1
        plt.plot(frq/1e6, UoutAmpl, linewidth=linewidthPlot)
        plt.grid(True)
        plt.ylabel(r'$|U_{\mathrm{out}}(f)|$')
        plt.xlabel(r'$f$ in MHz')
        fig.savefig(directory + "/Plots/UoutAmpl_frq_linear.pdf",\
                    bbox_inches='tight')
        plt.show()

        fig = plt.figure(f+1)
        f+=1
        plt.plot(frq/1e6, H, linewidth=linewidthPlot)
        plt.grid(True)
        plt.ylabel(r'$|H(f)|$')
        plt.xlabel(r'$f$ in MHz')
        fig.savefig(directory + "/Plots/H_linear.pdf", bbox_inches='tight')
        plt.show()
##################################################
############## Create CSV ########################
##################################################

if createCSV:

    with open(directory + '/csv/UinTime.csv', 'w', newline="") as csvfile:
        writer = csv.writer(csvfile, delimiter=';',
                            quotechar='|', quoting=csv.QUOTE_MINIMAL)
        for i in range(0,dataUin.size):
```

```
            writer.writerow([str(time[i]), str(dataUin[i])])

    with open(directory + '/csv/UoutTime.csv', 'w', newline="") as csvfile:
        writer = csv.writer(csvfile, delimiter=';',
                            quotechar='|', quoting=csv.QUOTE_MINIMAL)
        for i in range(0,dataUout.size):
            writer.writerow([str(time[i]), str(dataUout[i])])

    with open(directory + '/csv/PhaseH.csv', 'w', newline="") as csvfile:
        writer = csv.writer(csvfile, delimiter=';',
                            quotechar='|', quoting=csv.QUOTE_MINIMAL)
        for i in range(0,PhaseH.size):
            writer.writerow([str(frq[i]), str(PhaseH[i])])

    if (formatOutput==1) or (formatOutput==2):

        with open(directory + '/csv/UinAmplFrq_linear.csv', 'w',\
                  newline="") as csvfile:
            writer = csv.writer(csvfile, delimiter=';',
                                quotechar='|', quoting=csv.QUOTE_MINIMAL)
            for i in range(0,UinAmpl.size):
                writer.writerow([str(frq[i]), str(UinAmpl[i])])

        with open(directory + '/csv/UoutAmplFrq_linear.csv', 'w',\
                  newline="") as csvfile:
            writer = csv.writer(csvfile, delimiter=';',
                                quotechar='|', quoting=csv.QUOTE_MINIMAL)
            for i in range(0,UoutAmpl.size):
                writer.writerow([str(frq[i]), str(UoutAmpl[i])])

        with open(directory + '/csv/HAmpl_linear.csv', 'w',\
                  newline="") as csvfile:
            writer = csv.writer(csvfile, delimiter=';',
                                quotechar='|', quoting=csv.QUOTE_MINIMAL)
            for i in range(0,H.size):
                writer.writerow([str(frq[i]), str(H[i])])

    if (formatOutput==0 or formatOutput==2):

        with open(directory + '/csv/UinAmplFrq_dB.csv', 'w',\
                  newline="") as csvfile:
            writer = csv.writer(csvfile, delimiter=';',
                                quotechar='|', quoting=csv.QUOTE_MINIMAL)
            for i in range(0,UinAmpl.size):
                writer.writerow([str(frq[i]),
                                 str(20*np.log10(UinAmpl[i]))])

        with open(directory + '/csv/UoutAmplFrq_dB.csv', 'w',\
                  newline="") as csvfile:
            writer = csv.writer(csvfile, delimiter=';',
                                quotechar='|', quoting=csv.QUOTE_MINIMAL)
            for i in range(0,UoutAmpl.size):
                writer.writerow([str(frq[i]),
                                 str(20*np.log10(UoutAmpl[i]))])

        with open(directory + '/csv/HAmpl_dB.csv', 'w',\
                  newline="") as csvfile:
            writer = csv.writer(csvfile, delimiter=';',
                                quotechar='|', quoting=csv.QUOTE_MINIMAL)
            for i in range(0,H.size):
                writer.writerow([str(frq[i]), str(20*np.log10(H[i]))])
```

```
with open(directory + '/csv/UinFrq.csv', 'w', newline="") as csvfile:
    writer = csv.writer(csvfile, delimiter=';',
                        quotechar='|', quoting=csv.QUOTE_MINIMAL)
    for i in range(0,Uin.size):
        writer.writerow([str(frq[i]), str(Uin[i])])

with open(directory + '/csv/UoutFrq.csv', 'w', newline="") as csvfile:
    writer = csv.writer(csvfile, delimiter=';',
                        quotechar='|', quoting=csv.QUOTE_MINIMAL)
    for i in range(0,Uout.size):
        writer.writerow([str(frq[i]), str(Uout[i])])
```

8.2.2 MLBS.py

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Fri Feb 17 16:41:07 2017

@author: Armin Galetzka, Denys Bast

Creates and returns the MLBS signal for a specified number of bits.
Standard: bits=9
Input:  bits          ------  Number of bits for register length
Output: output        ------  MLBS Signal
        seedRandom    ------  Used seed for random number generator
"""

import numpy as np
from scipy import stats

def get(bits=10):
    # ------ create random start register ------
    seedRandom = np.random.randint(2**31)
    np.random.seed(seed=579946590)
    xk = np.arange(2)
    pk = (0.5,0.5)
    custm = stats.rv_discrete(name='custm', values=(xk, pk))
    # ------ create Signal ------
    register = np.zeros(bits)
    while (np.sum(register)==0):
        register = np.array(custm.rvs(size=bits))
    N = 2**bits-1
    output = np.zeros(N)

    for i in range(0,N):
        output[i] = register[bits-1]
        if bits==6 or bits==7:
            r = np.logical_xor(register[0],register[bits-1])
        if bits==8:
            r = np.logical_xor(np.logical_xor(np.logical_xor(register[0],\
                               register[1]),register[6]),register[7])
        if bits==9:
            r = np.logical_xor(register[3],register[8])
        if bits==10:
            r = np.logical_xor(register[2],register[9])
        if bits==11:
            r = np.logical_xor(register[1],register[10])
        if r:
            r=1
        else:
            r=0
        register = np.append(r, register[0:bits-1])

    output = output - 0.5
    return (output, seedRandom)
```

8.2.3 runme.py

```
# -*- coding: utf-8 -*-
"""
Created on Wed Apr 12 10:00:41 2017

@author: denys
"""

import getH

getH.compute(80e6,40e-3,9,True,True,True,2)
```

8.2.4 FFT.py

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Sat Feb 18 15:49:32 2017

@author: Armin Galetzka, Denys Bast
Returns the FFT amplitude and the frequency vector for a given signal

Input:  singal   --------  signal in time domain
        Fns      --------  sample frequency

Output  frq      --------  frequency vector
        ampl     --------  vector with amplitude
        phase    --------  vector with the phase
        H        --------  vector with complex frequency domain
"""
import numpy as np

def get(signal, Fns):

    # -------- get FFT
    n = signal.size
    H = np.fft.fft(signal)
    amplH = abs(H)

    # get frequency vector and normalize frequency domain
    fn = Fns/2
    df = Fns/n
    frq = np.arange(0,fn,df)
    ind = int(np.round(n/2))
    ampl = np.append(amplH[0]/n, amplH[1:ind]/(n/2))
    H = np.append(H[0]/n, H[1:ind]/(n/2))
    phase = np.angle(H)
    return (frq, ampl, phase, H)
```

Literaturverzeichnis

[1] Frey, M.: *Generierung von Kleinsignal Barrier-Bucket Pulsen mittels Bestimmung von Übertragungsfunktionen an MA-Kavitäten* / GSI Darmstadt. 2015. – Forschungsbericht

[2] Frey, M. ; Hülsmann, P. ; F-Michler ; Klingbeil, H. ; Domont.Yankulova, D. ; Gross, K. ; Hazheim, J.: *Status of the Barrier-Bucket system for the ESR* / GSI Darmstadt, TU Darmstadt. 2015. – Forschungsbericht

[3] Frohberg, Wolfgang ; Kolloschie, Horst ; Löffler, Helmut: *Taschenbuch der Nachrichtentechnik.* Carl Hanser Verlag GmbH & Co. KG, 2008

[4] Gentle, J.: *Random number generation and Monte Carlo methods.* Springer, 2003

[5] Golomb, S.W.: *Shift Register Sequences - A Retrospective Account.* World Scientific Publishing, 2006

[6] Gross, K.: *Übertragung einzelner oder aneinander gereihter Pulse durch Breitband-Kavitäten* / GSI Darmstadt. 2013. – Forschungsbericht

[7] Harzheim, J.: *Allgemeine Überlegungen zur Signalqualität von Einzelsinus BB-Pulsen* / GSI Darmstadt. 2017. – Forschungsbericht

[8] Harzheim, J. ; Domont-Yankulova, D. ; Frey, M. ; Gross, K. ; Klingbeil, H.: *Input Signal generation for Barrier Bucket RF Systems at GSI.* In: *IPAC Kopenhagen*, 2017

[9] Hülsmann, P. ; Frey, M. ; Balss, R. ; Klingbeil, H.: *Technical Concept SIS100 Barrier-Bucket System* / GSI Helmholtzzentrum für Schwerionenforschung. 2015. – Forschungsbericht

[10] Isermann, R. ; Münchhof, M.: *Identification of Dynamic Systems.* Springer, 2011

[11] Jatta, S. ; Gross, K.: *Spectrum of the BB signal at different periodicities* / GSI Darmstadt. 2013. – Forschungsbericht

[12] Lenz, E.: *Identifikation dynamischer Systeme.* Technische Universität Darmstadt, Institut für Automatisierungstechnik, Fachgebiet Regelungstechnik und Mechatronik, 2014

[13] NI: *National Instruments.* `https://www.ni.com/visa/`, 2017. – abgerufen: 13.04.2017

[14] Pintelon, R. ; Schoukens, J.: *System Identification A Frequency Domain Approach.* IEEE Press, 2001

[15] Python: *Python Software Foundation.* `https://www.python.org/downloads/release/python-360/`, 2017. – abgerufen: 13.04.2017