Technische Universität Darmstadt
FG Kommunikationstechnik
Prof. Dr.-Ing. Anja Klein

Kommunikationstechnik I

Lösung zur 2. Python-Übung

Aufgabe 1: **Zufallszahlen**

1.1 Grundlegend werden Zufallszahlen in Python mit Hilfe von Zufallszahlengeneratoren erzeugt. Wie auch sonst in Programmiersprachen üblich müssen die Zufallszahlengeneratoren initialisiert werden. `NumPy` stellt hierzu explizit das Modul `random` zur Verfügung. Die in Python verwendeten Zufallszahlengeneratoren erzeugen nur Pseudo-Zufallszahlen. Weiterführende, detailliertere Hintergrundinformationen können der Modul Dokumentation entnommen werden. Für die nachfolgenden Aufgaben wird auf eine explizite Initialisierung eines Zufallsgenerators verzichtet.

In Python sind diverse Funktionen integriert, die Realisierungen bestimmter Zufallszahlen erzeugen. Mit dem Befehl `np.rand()` wird eine Realisierung einer gleichverteilten Zufallszahl aus dem Intervall [0 1] erzeugt. Dieser Befehl kann mit einer Größenangabe kombiniert werden, sodass ein Array oder eine Matrix mit zufälligen und unabhängigen Realisierungen der Zufallszahl entsteht. Eine Realisierung einer auf dem Intervall $[a\ b]$ gleichverteilten Zufallszahl kann mit dem Befehl `a+(b-a)*np.rand()` erzeugt werden.

Abbildung 1 zeigt gemessene Häufigkeiten für 1.000.000 Realisierungen der Zufallszahlen mit den verschiedenen Verteilungen. Beachten Sie, dass die betrachteten Realisierungen nicht perfekt gleichverteilt sind, wie Sie an dem ausgefransten Verlauf erkennen können. Je mehr Realisierungen Sie betrachten, desto glatter werden die Verläufe der Häufigkeitsverteilungen. Eine perfekte Verteilung kann nur bei Betrachtung unendlich vieler Realisierungen erzielt werden.

`NumPy` stellt zudem zwei Funktionen `np.mean()` und `np.std()` zur Verfügung, mit denen der Erwartungswert und die Standardabweichung von Realisierungen einer Zufallsvariablen bestimmt werden können.

1.2 In Python besteht die Möglichkeit mit Hilfe von `np.random.rand()` eine Realisierung einer Zufallszahl, die gleichwahrscheinlich 0 und 1 annimmt, zu erzeugen. Allerdings lässt sich so nicht der zweite

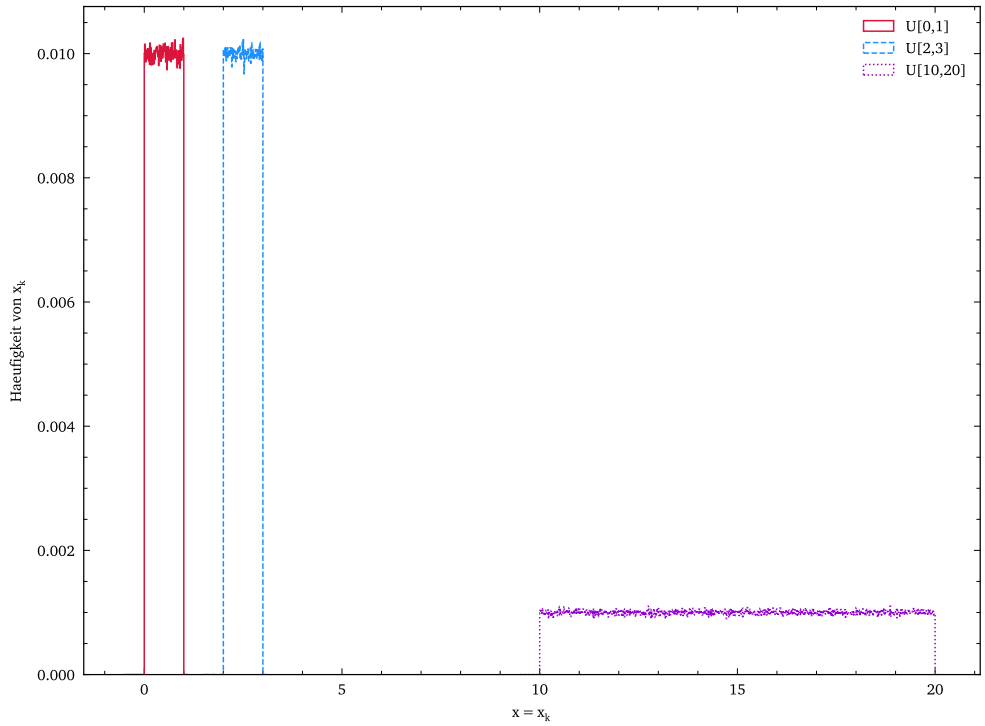

Abbildung 1: Häufigkeit der Realisierung $x = x_k$

Teil dieser Aufgabe lösen, sodass ein Alternativweg vorgeschlagen wird. Es werden zunächst N Realisierungen einer gleichverteilten Zufallszahl aus dem Intervall [0 1] generiert. Diese werden mit einem Schwellwert x_{SW}, der die Wahrscheinlichkeiten darstellt verglichen. Für den ersten Teil läge dieser bei $x_{\mathrm{SW}} = 0.5$, für den zweiten Teil ist der Schwellwert bei $x_{\mathrm{SW}} = 0.3$ festzusetzen. Ein Vergleich liefert einen boolschen Ausdruck, der den Wert `True` annimmt, falls die Realisierung oberhalb des Schwellwertes liegt. Durch casting der resultierenden boolschen Datentypen in Integer werden daraus die binären Werte 0 (`False`) oder 1 (`True`) generiert.

1.3 Mit der Funktion `np.random.randn()` lassen sich Realisierungen einer gaußverteilten Zufallszahl mit Erwartungswert 0 und mit Standardabweichung 1 erzeugen. Die Wahrscheinlichkeitsdichtefunktion der Zufallsvariablen ist gegeben durch

$$f_x(x) = \frac{1}{\sqrt{2\pi}} \mathrm{e}^{-\frac{x^2}{2}}.$$

Abbildung 2 zeigt die Wahrscheinlichkeitsdichtefunktion einer erwartungswertfreien gaussverteilten Zufallszahl mit Standardabweichung 1 und die gemessenen relativen Häufigkeiten der Realisierungen in den Intervallen der Form $[x_k - 0,5\ x_k + 0,5)$.

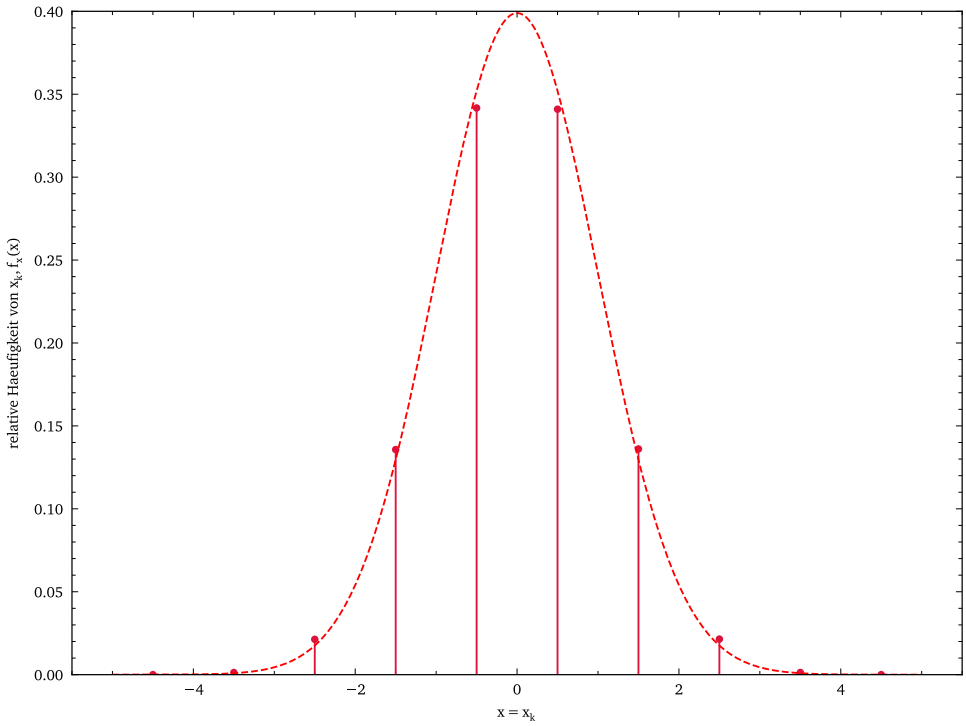

Abbildung 2: Relative Häufigkeit von Realisierungen und Wahrscheinlichkeitsdichtefunktion einer erwartungswertfreien gaussverteilten Zufallszahl mit Standardabweichung 1

Realisierungen beliebiger gaussverteilter Zufallsvariablen mit Erwartungswert m und Standardabweichung s lassen sich durch den Befehl `m+s*np.random.randn()` erzeugen.

Aufgabe 2: **Fourier-Transformation**

2.1 Das Signal $x(t)$ kann geschrieben werden als

$$x(t) = 2d_1\left(\frac{t}{2}\right) - d_1(t)$$

wobei $d_1(t)$ ein Dreiecksimpuls ist, der bei $t = -1$ linear ansteigt, für $t = 0$ den Wert 1 erreicht, dann wieder linear abfällt und für Werte $t < -1$ und $t > 1$ den Wert 0 annimmt. Wir verwenden die Linearitäts- und Skalierungseigenschaft der Fouriertransformation, sowie die Eigenschaft, dass die Fouriertransformierte von $d_1(t)$ durch $\mathrm{si}^2(f)$ gegeben ist und erhalten für die Fouriertransformierte von $x(t)$

$$X(f) = 4\mathrm{si}^2(2f) - \mathrm{si}^2(f).$$

Um die Fouriertransformierte mit Python zu bestimmen, benötigen wir zuerst eine grobe Abschätzung der Bandbreite des Signals.

Da das Signal keine großen Sprünge besitzt, ist die Bandbreite umgekehrt proportional zu der Zeitdauer des Signals. Die Zeitdauer beträgt 4 s. Um sicher zu gehen, nehmen wir eine Bandbreite an, die zehnmal der Inversen der Zeitdauer entspricht:

$$BW = 10\frac{1}{4\,\mathrm{s}} = 2.5\,\mathrm{Hz} \tag{1}$$

Die Nyquist-Frequenz ergibt sich als zweimal die Bandbreite zu 5 Hz. Das Abtastintervall ist also gegeben durch $T_s = 1/f_s = 0.2\,\mathrm{s}$. Mit diesen Parametern kann die Fouriertransformierte mit Hilfe der bereitgestellten Funktion `fftseq()` berechnet werden, welche wiederum `NumPys` Funktion `np.fft.fft()` verwendet. Die Funktion berechnet eine Frequenzauflösung von etwa 0.01 Hz, welche für die anschließende Darstellung verwendet wird. Abbildung 3 zeigt das Amplitudenspektrum von $x(t)$ analytisch berechnet sowie das Ergebnis, welches aus der Verwendung von `fftseq()` resultiert.

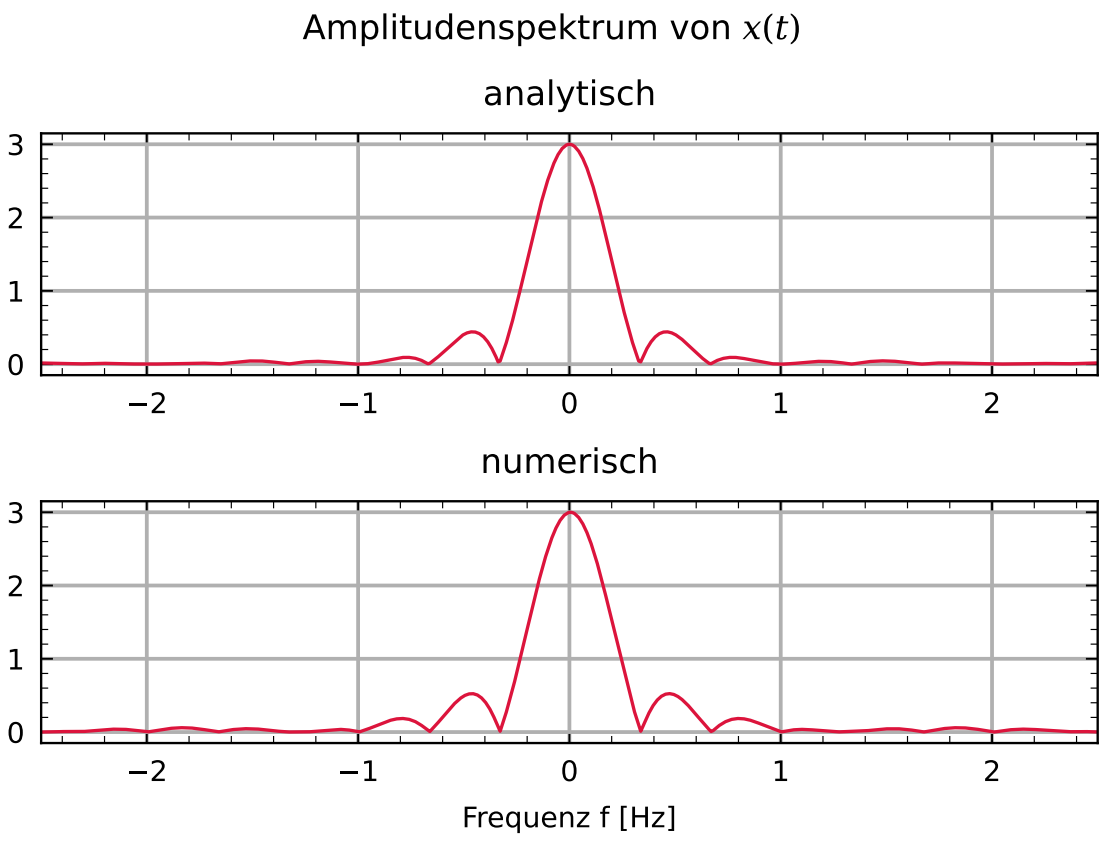

Abbildung 3: Amplitudenspektrum von $x(t)$

2.2 Betrachten wir einen Vektor $\underline{d}$, bei dem das n_0-te Element 1 ist. Die IFFT des Vektors $\underline{d}$:

$$s_k = \sum_{n=0}^{N-1} d_n \mathrm{e}^{j2\pi kn/N} = \mathrm{e}^{j2\pi kn_0/N}, \quad k = 0, 1, \cdots, N-1$$

Nach dem Anhängen von Nullen erhalten wir den Vektor $\underline{s}'$ mit Länge $N_1 = 8N$:

$$s'_k = s_k \quad \text{für } k = 0, 1, \cdots, N-1 \tag{2}$$
$$s'_k = 0 \quad \text{für } k = N, N+1, ..., N_1 - 1 \tag{3}$$

Die DFT des Vektors $\underline{s}'$ ist ein Vektor $\underline{d}'$ mit Länge N_1:

$$d'_n = \sum_{k=0}^{N_1-1} s'_k \mathrm{e}^{-j2\pi nk/N_1} \tag{4}$$
$$= \sum_{k=0}^{N-1} \mathrm{e}^{j2\pi n_0 k/N} \mathrm{e}^{-j2\pi nk/N_1} \tag{5}$$
$$= \sum_{k=0}^{N-1} \mathrm{e}^{j2\pi(n_0/N - n/N_1)k} \tag{6}$$
$$= \frac{\mathrm{e}^{j2\pi(n_0 - nN/N_1)} - 1}{\mathrm{e}^{2\pi(n_0/N - n/N_1)} - 1} \tag{7}$$
$$= \mathrm{e}^{j2\pi(n_0/N - n/N_1)(N-1)/2} \frac{\sin(\pi(n_0 - nN/N_1))}{\sin(\pi(n_0/N - n/N_1))} \tag{8}$$

für $n = 0, 1, \cdots, N_1 - 1 = 8N - 1$. Abbildung 4 zeigt das Amplitudenspektrum der Elemente von $\underline{d}'$. Man erkennt, dass die einzelnen Spektren orthogonal im diskreten Frequenzraster $\Omega_n = {}^{2\pi n}/_N$ sind.

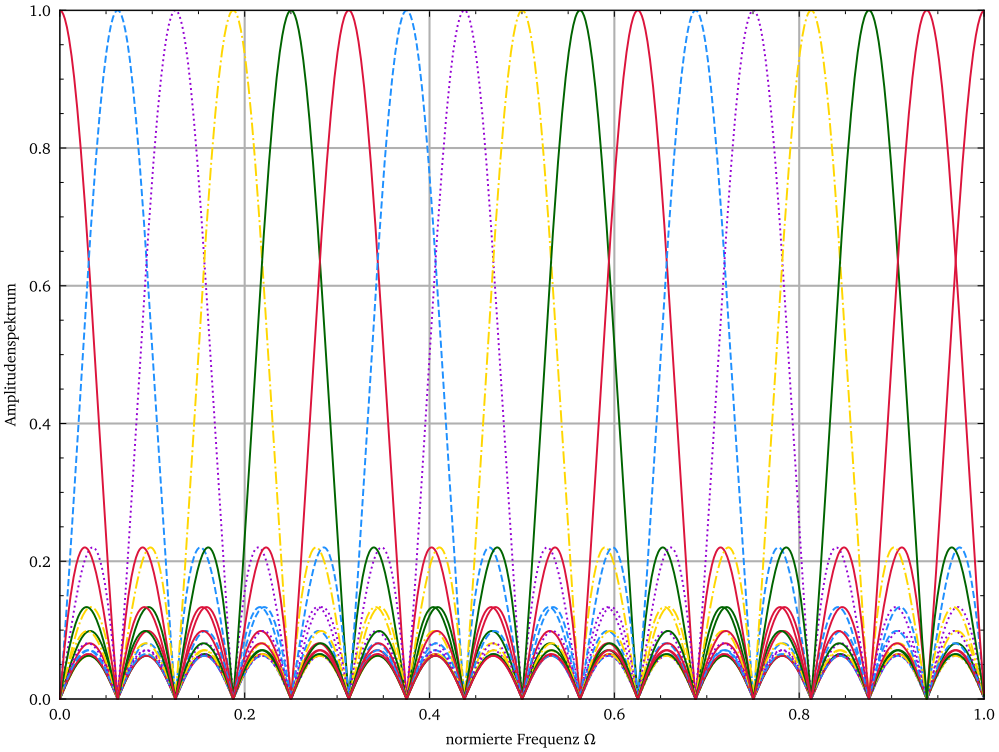

Abbildung 4: Amplitudenspektrum von s