

## Aufgabenblatt 5 (Praxis)

wr@cg.tu-berlin.de

WiSe 2022/2023

### Allgemeine Hinweise:

- Die Aufgaben sind von jeder/m Studierenden *einzel*n zu bearbeiten und abzugeben (Plagiate werden entsprechend der Studienordnung geahndet).
- Verwenden Sie die vorgegebene Code-Basis. **Sie dürfen keine weiteren Module importieren.** Die zu implementierenden Funktionen befinden sich in der Datei `main.py`. Ihr Code ist an den mit `# TODO: ...` gekennzeichneten Stellen einzufügen. Die NumPy-Funktionen, welche Sie zur Lösung einer Aufgabe nicht verwenden dürfen, sind unter `Forbidden` in der Docstring Beschreibung der entsprechenden Funktion aufgelistet.
- Wir stellen einige rudimentäre Unit-Tests zur Verfügung, welche Sie verwenden sollen, um die Funktionalität ihres Codes zu testen. Sie sollten diese Tests während der Implementierung Ihrer Lösung vervollständigen (Funktionalität beschrieben in Python `unittest`). Sie können die Tests mit dem Aufruf `python3 tests.py -v [Tests.test_<function>]` ausführen.
- Bitte reichen Sie die Datei `main.py` mit Ihren Lösungen bis **Sonntag, den 29.01.2023, um 20:00 Uhr** auf <https://autolab.service.tu-berlin.de> mit ihren Zugangsdaten ein. Ein mehrfacher Upload bis zum Abgabende ist möglich. Die letzte Version wird bewertet.

### Aufgabe 1: Diskrete Fourier-Transformation (2 Punkte)

In dieser Aufgabe soll die diskrete Fourier-Transformation (DFT) mit Hilfe der DFT-Matrix realisiert werden.

#### Aufgabe 1.1: DFT-Matrix (1 Punkte)

Implementieren Sie die Funktion `dft_matrix()`, welche die normierte DFT-Matrix der Größe  $n \times n$  zurückgeben soll.

#### Aufgabe 1.2: Überprüfung Unitäre Matrix (0.5 Punkt)

Implementieren Sie die Funktion `is_unitary()`, welche überprüft, ob das Argument eine unitäre Matrix ist (d.h.  $F^* F = I$  wobei  $F^*$  die konjugiert-transponierte von  $F$  ist und  $F \in \mathbb{R}^{N \times N}$ ).

#### Aufgabe 1.3: DFT von $\delta$ -Impulsen (0.5 Punkt)

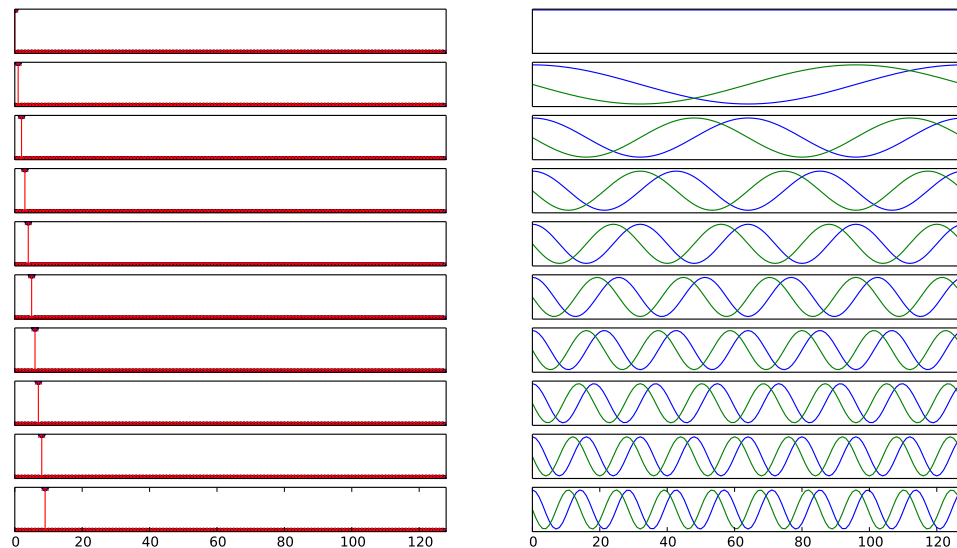
Berechnen Sie in der Funktion `create_harmonics()` mit Hilfe der DFT-Matrix die diskrete Fourier-Transformation von diskreten  $\delta$ -Impulsen der Form  $\mathbf{e}_i = (0, \dots, 1, \dots, 0)^T$  für welche das  $i$ -te Element 1 ist und alle anderen 0 sind. Die Gesamtlänge von  $\mathbf{e}_i$  soll 128 betragen. Vergleichen Sie die Visualisierung der Signale durch `plot_harmonics()` mit Abbildung 1.

### Aufgabe 2: Schnelle Fourier-Transformation (2 Punkte)

In dieser Aufgabe soll die Cooley-Tukey Variante der schnellen Fourier-Transformation (FFT) für Eingangsdaten der Länge  $2^k$  implementiert werden.

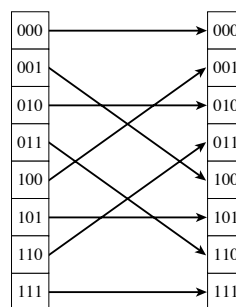
#### Aufgabe 2.1: Umordnung der Eingangsdaten (0.5 Punkt)

Implementieren Sie die Funktion `shuffle_bit_reversed_order()`, welche die Elemente eines gegebenen Arrays so umordnet, dass der neue Index für ein Element durch die Umkehrung der Bitdarstellung

Abbildung 1: Graphische Ausgabe der Funktion `plot_harmonics()`.

des alten Index entsteht.

Für ein Array der Länge 8 sieht diese Umordnung zum Beispiel folgendermaßen aus (links vor dem Umordnen und rechts danach):<sup>1</sup>



### Aufgabe 2.2: Schnelle Fourier-Transformation (1.5 Punkte)

Implementieren Sie die FFT in der Funktion `fft()`. Die FFT verwendet implizit einen Baum, welcher von den Blättern, gegeben durch die umgeordneten Elemente der Eingangsdaten, zur Wurzel hin verarbeitet wird. Das Ergebnis ist die gesuchte Fourier-Transformation.

Nummerieren wir die Ebenen des Baumes von den Blättern beginnend mit  $m = 0$ , so erfolgen auf jeder Ebene  $n/2^{m+1}$  diskrete Fourier-Transformationen der Länge  $2^{m+1}$ . Jede dieser Fourier-Transformationen wird aus elementaren Transformationen für zwei Elemente aufgebaut:

$$p = e^{-2\pi i k / 2^{m+1}} f[j] \quad (1a)$$

$$f[j] = f[i] - p \quad (1b)$$

$$f[i] = f[i] + p \quad (1c)$$

wobei der Faktor  $e^{-2\pi i k / 2^{m+1}}$  den korrekten Frequenzanteil für das Ergebnis bestimmt. Die aktuellen Werte in  $f$  enthalten dabei für  $m = 0$  die umsortierten Eingangsdaten und für höhere Ebenen im Baum die Ergebnisse der Fourier-Transformationen, welche auf tieferliegenden Schichten des Baumes berechnet wurden. Der Abstand von  $i$  und  $j$  ist durch  $2^m$  gegeben, und  $i$  läuft über  $k, k + 2^{m+1}, k + 2 \cdot 2^{m+1}, k + 3 \cdot 2^{m+1} \dots < n$ . Daraus folgt, dass zunehmend weiter auseinanderliegende Elemente zusammengefasst werden, je höher man im Baum aufsteigt. Für eine Fourier-Transformation der Länge

<sup>1</sup>Aus: Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (1992). Numerical Recipes in C: The Art of Scientific Computing. New York, NY, USA: Cambridge University Press.

$2^{m+1}$  muss die Frequenzvariable  $k$  von 0 bis  $2^m$  (exklusive) laufen. Aus Effizienzgründen werden auf jeder Ebene des Baumes dabei für jedes  $k$  die  $n/2^{m+1}$  elementaren Transformationen in Gleichung 1 ausgeführt bevor  $k$  erhöht wird. Es erfolgt also eine Umordnung der beiden innersten Schleifen. In Pseudo-Code haben wir damit also:

```
# for all levels m of the tree

# for all values of k = [0,1,...,2^m[ on the current level

# compute omega factor for current k

# for all values of i,j with i = [k,k+2^(m+1),k+2*2^(m+1),...,n[

# perform elementary transformation
```

Der Ablauf der FFT für ein Signal der Länge 8 ist im Anhang gezeigt. Vergleichen Sie mit Hilfe des zur Verfügung gestellten Test-Codes die Abweichung eines Signals nach Vor- und Rücktransformation (siehe Abbildung 2) sowie die Rechenzeit, welche Sie mit `dft_matrix()` und mit `fft()` benötigen (siehe Abbildung 3).

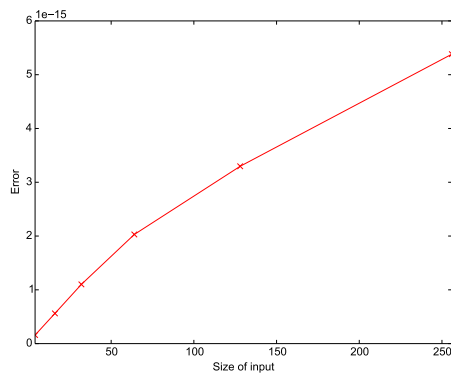


Abbildung 2: Fehler zwischen originalem Signal und vor- und rückwärts transformiertem Signal mit der FFT.

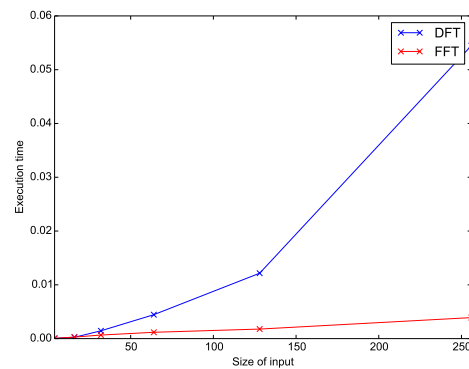


Abbildung 3: Vergleich der benötigten Rechenzeit bei Berechnung mit den Funktionen `dft_matrix()` und `fft()`.

### Aufgabe 3: Verarbeitung von Audiosignalen (1 Punkte)

#### Aufgabe 3.1: Tonerzeugung (0.5 Punkt)

Erzeugen sie in der Funktion `generate_tone()` ein mittleres C: Sinusschwingung mit der Frequenz  $f = 261.63$  Hz, abgetastet im Intervall  $[0, 1]$ s mit einer vorgegebenen Anzahl an Abtastwerten (`num_samples`). Der zur Verfügung gestellte Test-Code speichert den von Ihnen zurückgegebenen Ton in `data/mid-c.wav`. Zum Vergleich ist die Datei `data/mid-c_ref.wav` gegeben.

#### Aufgabe 3.2: Tiefpassfilter (0.5 Punkt)

In der Funktion `low_pass_filter()` soll ein einfacher Tiefpassfilter implementiert werden, welcher die hohen Frequenzen eines gegebenen Signals unterdrückt. Um den Filter zu realisieren soll:

- die Fourierdarstellung des Signals berechnet werden,
- die hohen Frequenzen über 1000 Hz in der Frequenzdarstellung auf Null gesetzt werden,
- das Signal aus dem Frequenzraum wieder in den Ortsraum transformiert werden.

Der zur Verfügung gestellte Test-Code speichert das von Ihnen gefilterte Signal in `data/speech-filtered.wav`. Vergleichen sie dieses mit der vorgegebenen Datei `data/speech-filtered_ref.wav`.

### Anhang: Beispiel für FFT Ablauf

Für jedes Quadrupel  $(m, k, i, j)$  wird eine der elementaren Transformationen in Gleichung 1 ausgeführt:

Eingangsdaten: [0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8]

Nach Shuffle:

[0.1+0.j 0.5+0.j 0.3+0.j 0.7+0.j 0.2+0.j 0.6+0.j 0.4+0.j 0.8+0.j]

$m = 0, k = 0 : i = 0, j = 1$

[0.6+0.j -0.4+0.j 0.3+0.j 0.7+0.j 0.2+0.j 0.6+0.j 0.4+0.j 0.8+0.j]

$m = 0, k = 0 : i = 2, j = 3$

[0.6+0.j -0.4+0.j 1.0+0.j -0.4+0.j 0.2+0.j 0.6+0.j 0.4+0.j 0.8+0.j]

$m = 0, k = 0 : i = 4, j = 5$

[0.6+0.j -0.4+0.j 1.0+0.j -0.4+0.j 0.8+0.j -0.4+0.j 0.4+0.j 0.8+0.j]

$m = 0, k = 0 : i = 6, j = 7$

[0.6+0.j -0.4+0.j 1.0+0.j -0.4+0.j 0.8+0.j -0.4+0.j 1.2+0.j -0.4+0.j]

$m = 1, k = 0 : i = 0, j = 2$

[1.6+0.j -0.4+0.j -0.4+0.j -0.4+0.j 0.8+0.j -0.4+0.j 1.2+0.j -0.4+0.j]

$m = 1, k = 0 : i = 4, j = 6$

[1.6+0.j -0.4+0.j -0.4+0.j -0.4+0.j 2.0+0.j -0.4+0.j -0.4+0.j -0.4+0.j]

$m = 1, k = 1 : i = 1, j = 3$

[1.6+0.j -0.4+0.4j -0.4+0.j -0.4-0.4j 2.0+0.j -0.4+0.j -0.4+0.j -0.4+0.j]

$m = 1, k = 1 : i = 5, j = 7$

[1.6+0.j -0.4+0.4j -0.4+0.j -0.4-0.4j 2.0+0.j -0.4+0.4j -0.4+0.j -0.4-0.4j]

$m = 2, k = 0 : i = 0, j = 4$

[3.6+0.j -0.4+0.4j -0.4+0.j -0.4-0.4j -0.4+0.j -0.4+0.4j -0.4+0.j -0.4-0.4j]

$m = 2, k = 1 : i = 1, j = 5$

[3.6+0.j -0.4+0.966j -0.4+0.j -0.4-0.4j -0.4+0.j -0.4-0.166j -0.4+0.j -0.4-0.4j]

$m = 2, k = 2 : i = 2, j = 6$

[3.6+0.j -0.4+0.966j -0.4+0.4j -0.4-0.4j -0.4+0.j -0.4-0.166j -0.4-0.4j -0.4-0.4j]

$m = 2, k = 3 : i = 3, j = 7$

[3.6+0.j -0.4+0.966j -0.4+0.4j -0.4+0.166j -0.4+0.j -0.4-0.166j -0.4-0.4j -0.4-0.966j]