Methodische und Praktische Grundlagen der Informatik 4

Aufgabenblatt 4

mpgi4@cg.tu-berlin.de

WiSe 2013/2014

Allgemeine Hinweise:

- Die Aufgaben sollen in Gruppen bestehend aus zwei bis drei Personen bearbeitet werden. Ausnahmen müssen mit dem jeweiligen Tutor abgesprochen werden.
- Bitte reichen Sie Ihre Lösungen in Form einer ZIP Datei bis **Sonntag, den 12.01.2014, um 23:55 Uhr** auf der ISIS Webseite der Vorlesung ein. Wählen Sie dazu für Ihre Gruppe eine/n Sprecher/in, welche/r die Abgabe durchführt, und tragen Sie in jede abgegebene Datei Name und Email-Adresse aller Gruppenmitglieder ein.
- Wenn eine Aufgabe die Abgabe einer Grafik verlangt, dann muss ein vollständig funktionsfähiges Programm in der Lösung enthalten sein, welches bei der Ausführungen die Grafik erstellt. Wenn zum Erstellen einer Grafik bereits eine Funktion gegeben ist, dann darf diese nicht verändert werden.
- Tragen Sie in die Datei `README.txt` Name und Email-Adresse aller Gruppenmitglieder sowie einige kurze Kommentare zu Ihrer Abgabe ein. Vermerken Sie insbesondere Fehler oder Unvollständigkeiten in Ihrer Abgabe.

Aufgabe 1: Diskrete Fourier-Transformation (4.0 Punkte)

Die diskrete Fourier-Transformation ist eine lineare Transformation eines Eingangssignals. Sie kann damit als eine Matrix realisiert werden.

a) Implementieren Sie die DFT-Matrix der diskreten Fourier-Transform in der Funktion `dftMatrix()` im zur Verfügung gestellten Skelett-Code `dft.py`. (2.0 Punkte)

b) Implementieren Sie die Funktion `dft()`, welche die DFT mit Hilfe der DFT-Matrix berechnet. Testen Sie die Korrektheit Ihrer Implementierung, indem Sie überprüfen, ob Ihre DFT-Matrix unitär ist (d.h. $F^*F = I$ wobei F^* die konjugiert-transponierte von F ist), und ob das Ergebnis für reelle Eingangsdaten symmetrisch um die höchste Frequenz ist. Falls die Eigenschaften nicht erfüllt sind soll `dft()` ein `assert` werfen. (1.0 Punkte)

c) Berechnen Sie mit Hilfe der DFT-Matrix die diskrete Fourier-Transformation von Eingangssignalen der Form $e_i = (0, \cdots, 1, \cdots, 0)$ für welche das i-te Element 1 und alle anderen Null sind. Die Gesamtlänge von e_i soll 128 betragen. Vervollständigen Sie dazu die Funktion `plotHarmonics()`, welche die ersten 10 Ergebnissignale plotten soll, siehe Fig. 1. (1.0 Punkte)

Aufgabe 2: Schnelle Fourier Transformation (5.0 Punkte)

In dieser Aufgabe soll die Cooley-Tukey Variante der schnelle Fourier-Transformation (FFT) für Eingangsdaten der Länge 2^k implementiert werden. Dabei findet zunächst eine Umordnung der Elemente statt, welche bei der *in-place* Durchführung der Fourier-Transformation im zweiten Schritt einen einfachen Zugriff ermöglicht. Anschließend soll die Performance Ihrer FTT Implementierung mit der DFT-Matrix Implementierung aus Aufgabe 1 vergleichen werden.

a) Implementieren Sie die Funktion `shuffleBitReversedOrder()`, welche die Elemente eines gegeben NumPy Arrays so umordnet, dass das der neue Index für ein Element durch die Umkehrung

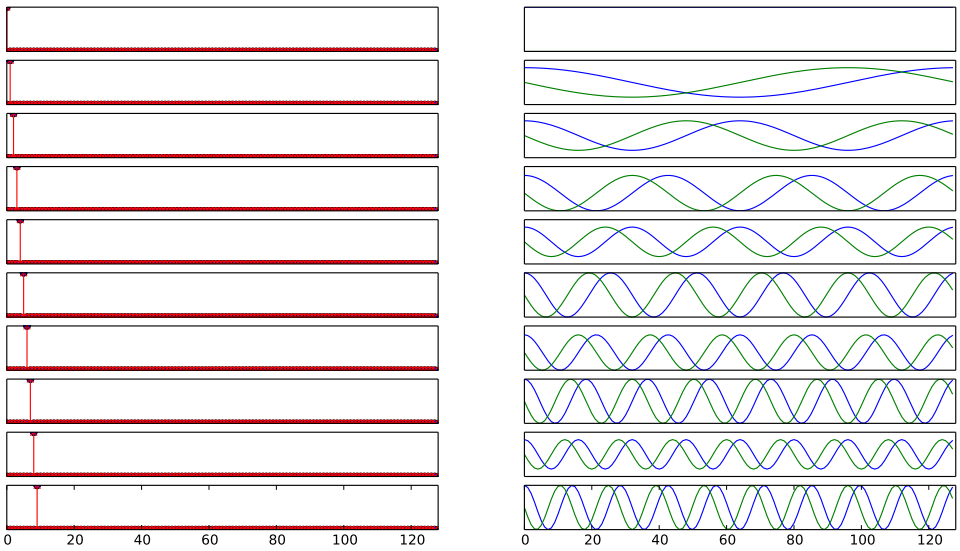
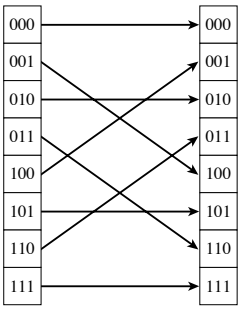


Abbildung 1: Graphische Ausgabe der Funktion `plotHarmonics()`.

der Bitdarstellung des alten Index entsteht (1.0 Punkte). Für ein Array der Länge 8 sieht diese Umordnung zum Beispiel folgendermaßen aus (links vor dem Umordnen und rechts danach):[1]



Eine einfache Möglichkeit, den neuen Index zu erzeugen ist für jedes Bit im alten Index zu überprüfen, ob dieses 0 oder 1 ist, und das umgekehrte Bit zu setzen, in dem man die korrekte Potenz 2^m zu einem vorläufigen neuen Index addiert, in welchem diese Terme akkumuliert werden. Für das Umordnen können Sie ausnutzen, dass dieses symmetrisch ist, d.h., dass immer zwei Elemente getauscht werden und dass dieser Tausch genau einmal erfolgen muss.

b) Implementieren Sie die FFT auf den in a) umgeordneten Daten (3.0 Punkte). Die FFT verwendet implizit einen Baum. Im zweiten Schritt wird nun dieser Baum von den Blättern, gegeben durch die umgeordneten Elemente der Eingangsdaten, zur Wurzel hin verarbeitet. 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 2^{m+1} diskrete Fourier-Transformationen der Länge 2^m. Jede dieser Fourier-Transformationen wird aus elementaren Transformation für zwei Elemente aufgebaut:

$$p = e^{-2\pi ik/2^{m+1}} f[j] \tag{1a}$$
$$f[j] = f[i] - p \tag{1b}$$
$$f[i] = f[i] + p \tag{1c}$$

wobei der Faktor $e^{2\pi ik/2^m}$ den korrekte Frequenzanteil für das Ergebnis bestimmt. Die aktuellen Werte in f enthalten dabei für $m = 0$ die umsortierten Eingangsdaten oder für höhere Ebenen im Baum die Ergebnisse der Fouriertransformationen, welche auf tieferliegenden Schichten des

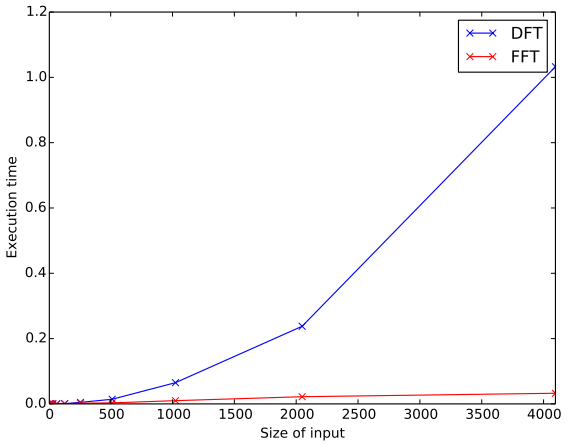
[1] Aus: Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (1992). Numerical Recipes in C: The Art of Scientific Computing. New York, NY, USA: Cambridge University Press.

Baumes berechnet wurden. Der Abstand von i und j ist durch 2^m gegeben, und i läuft über $k, k + 2^{m+1}, k + 2 \cdot 2^{m+1}, \ldots$. Daraus folgt, dass immer weiter auseinanderliegende Elemente zusammengefasst werden, desto höher man im Baum gelangt. Für eine Fourier-Transformation der Länge m muss die Frequenzvariable k von 0 bis 2^m laufen. Aus Effizienzgründen werden auf jeder Ebene des Baumes dabei für jedes k die $n/2^m$ 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 of the tree

  # for all values of k = 0 \ldots 2^m on the current level

    # compute omega factor for current k

    # for all values of i,j with $i = k \cdots n / 2^m$

      # perform elementary transformation
```

Als Beispiel können wir eine Sequenz der Länge 8 betrachten, siehe Anhang. Verifizieren Sie Ihren Code mit Hilfe der Funktion testFFT().

c) Vergleichen Sie die Recheneffizienz für die Durchführung der diskreten Fourier-Transformation mit der DFT-Matrix und der schnellen Fourier-Transformation. Verwenden Sie zufällige Testsignale der Längen 16, 32, 64, 128, 256, 512, 1024, 4096, 8192 und implementieren Sie die Tests in testPerformanceFFT(). Die Ergebnisse sollen mit der in der Funktion zur Verfügung gestellten Funktionalität graphisch dargestellt und gespeichert werden. (1.0 Punkte)



Aufgabe 3: Bonus: Verarbeitung von Audiosignalen (3 Punkte)

a) Erzeugen sie ein mittleres C (f = 263.63 Hz) und speichern Sie den Ton in `./data/mid-c.wav` mit Hilfe des zur Verfügung gestellten Paketes `audio`. Dies soll in der Funktion `midC()` implementiert werden (1.0 Punkte).

b) In der Funktion `filterHighs()` soll ein einfacher Tiefpass-Filter implementiert werden, welcher die hohen Frequenz eines gegebenen Signals unterdrückt. Die Funktionalität soll am Beispiel der Datei `./data/example.wav` demonstriert werden, welche im gegebenen Code bereits geladen ist. Um den Filter zu realisieren soll:

   i) die Fourierdarstellung des Signals berechnet und diese mit dem vorhanden Code dargestellt werden;

   ii) die hohen Frequenzen über 5000 Hz in der Frequenzdarstellung auf Null gesetzt werden;

   iii) das Signal aus dem Frequenzraum wieder in den Ortsraum transformiert werden.

   Die Funktion `filterHighs()` soll das gefilterte Signal im Ortsraum zurückgeben (2.0 Punkte).

Anhang: Beispiel für FFT Ablauf

Für jedes Quadrupel (m, k, i, j) wird eine der elementaren Transformationen in Gleichung 1 ausgeführt (mm $= 2^m$):

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]

mm = 1 , 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]

mm = 1 , 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]

mm = 1 , 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]

mm = 1 , 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]

mm = 2 , 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]

mm = 2 , 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]

mm = 2 , 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]

mm = 2 , 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]

mm = 4 , 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]

mm = 4 , 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]

mm = 4 , 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]

mm = 4 , 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]