Praktikum II „Numerik am Computer“ FS 2024
Prof. Dr. H. Harbrecht Universität Basel
M. Dujella, A. Heinimann, L. Mehrle, M. Morini, M. Ramm, D. Yusofy

---

Serie 6

The Fast and the Fourier Transform

Aufgabe 6.1 (1 Punkt)**:** Lies die Beilage sorgfältig durch und beantworte folgende Fragen:

a) Was ist die Idee der trigonometrischen Interpolation?

b) Wo kann die Fourier-Transformation beispielsweise angewendet werden (betrachte auch Aufgabe 6.3 und 6.5)?

c) Wieso heisst es *schnelle* Fourier-Transformation?

d) Wie funktioniert ein rekursiver Algorithmus? Schaue dir dazu das Beispiel zur Berechnung der Fakultät auf der Beilage an.

Aufgabe 6.2 (0.5 Punkte)**:** Wir betrachten zuerst ein kleines Beispiel für einen rekursiven Algorithmus. Schreibe eine Funktion

```
F_n = fibonacci(n),
```

welche die Fibonacci-Folge, gegeben durch

$$F_1 = 1, \quad F_2 = 1,$$
$$F_n = F_{n-1} + F_{n-2}, \quad n \geq 3,$$

rekursiv berechnet. Dabei solltest du ohne `for`-Schleifen auskommen. Was ist F_{10} und F_{20}?

Aufgabe 6.3 (1.5 + 0.5 + 1 + 0.5 Punkte)**:** In dieser Aufgabe werden wir die schnelle Fourier-Transformation (englisch: *fast Fourier transform* bzw. *FFT*) implementieren, mit der man periodische Funktionen besonders effizient an äquidistanten Stützstellen interpolieren kann (siehe Beilage).

a) Schreibe eine Funktion

```
gamma = fast_fourier_transform(y, n),
```

welche die schnelle Fourier-Transformation implementiert und den Koeffizientenvektor $\boldsymbol{\gamma}$ ausgibt, sodass $\boldsymbol{\beta} = \frac{1}{n}\overline{\boldsymbol{\gamma}}$ (siehe Beilage). Die Datei `fast_fourier_transform.m` von der Webseite enthält eine Vorlage, mit der du starten kannst. Beachte, dass keine `for`-Schleifen nötig sind!

Achtung: Der erste Index eines Vektors in MATLAB ist 1. Passe deshalb den Algorithmus von der Beilage entsprechend an.

b) Um deine Funktion aus Teil a) zu testen, lade zuerst die Funktion `saegezahn.m` von der Webseite herunter. Sie generiert dir zu der 2π-periodischen Sägezahn-Funktion für ein beliebiges n die Stützpunkte (x_k, y_k), mit $k = 0, \ldots, n-1$ und $x_k = 2\pi k/n$. Berechne für $n = 2^7$ zuerst den Koeffizientenvektor $\boldsymbol{\gamma}$ und dann den Vektor $\boldsymbol{\beta}$. Vergleiche das Ergebnis mit dem Output vom MATLAB-Befehl `1 / n * fft(y)`. Die Werte sollten bis auf Rundungsfehler übereinstimmen.

c) Schreibe eine Funktion

`[a, b] = trigon_coeff(beta, n)`,

welche die Koeffizienten a_ℓ und b_ℓ für das Interpolationspolynom $P(x)$ aufstellen soll (siehe Beilage).

Hinweis: Die Koeffizienten können durch Rundungsfehler noch sehr kleine Imaginärteile enthalten. Diese kannst du entfernen, indem du die Koeffizienten auf ihren Realteil reduzierst (MATLAB: `real`).

d) Um deine Funktion zu testen, verwende wieder die Sägezahn-Funktion aus Teil b). Lade ausserdem die Funktion `trigon_interpol.m` von der Webseite herunter, mithilfe derer du das Interpolationspolynom P auswerten kannst.

i) Berechne für $n = 2^2$ die Koeffizienten a_ℓ und b_ℓ. Dabei sollte

$$a_0 = a_2 \approx -1.5708, \quad b_1 = a_1 = -a_0$$

herauskommen. Berechne $P(x_k)$ für $k = 0, \ldots, n-1$. Stimmen die Werte jeweils mit y_k überein?

ii) Interpoliere die Stützstellen für $n = 2^5$ und berechne das Interpolationspolynom $P(x)$ für $x \in [0, 8\pi]$ (Schrittweite $\frac{\pi}{1000}$). Zeichne das Interpolationspolynom.

Aufgabe 6.4 (1 + 0.5 + 1 Punkte)**:** Wir wollen uns nun noch näher mit dem Aufwand der FFT beschäftigen.

a) Schreibe eine Funktion

`beta = beta_coeff(y, n)`,

die den Koeffizientenvektor $\boldsymbol{\beta}$ zum Interpolationspolynom $\Pi(x)$ mithilfe von `for`-Schleifen wie in Gleichung (2) auf der Beilage berechnet.

b) Wiederhole Aufgabe 6.3 b), um die Funktion `beta_coeff` zu testen.

c) Vergleiche den Aufwand der Funktionen `fast_fourier_transform` und `beta_coeff`, indem du $\boldsymbol{\beta}$ für $n = 2^j$, $j = 5, \ldots, 12$ mit beiden Funktionen berechnest lässt und jeweils misst, welche Funktion wieviel Zeit für jedes einzelne n braucht. Du kannst dies auf folgende Weise tun:

```
tic;
beta = beta_coeff(y, n);
t_vector(vector_index) = toc;
```

Mit der Sägezahn-Funktion kannst du dir für jedes n die Stützpunkte generieren lassen. Zeichne die Zeiten, die du erhalten hast, mit `loglog` in einer Grafik. Zeichne für jedes n auch die Vergleichsgeraden gegeben durch $t_1 \cdot n/2^5$ und $t_1 \cdot n^2/2^{10}$, wobei t_1 jeweils der erste gemessene Zeitwert ist. Was fällt dir auf?

Aufgabe 6.5 (0.5 + 0.5 + 0.5 + 1 Punkte)**:** Die Fourier-Transformation kann auch zur Kompression von Bildern benutzt werden. Ein Bild kann zum Beispiel als Tensor $\boldsymbol{I} \in \mathbb{R}^{m \times n \times 3}$ dargestellt werden. Jedes Pixel hat hierbei einen Index (i, j), $i = 1, \ldots, m$, $j = 1, \ldots, n$ und wird durch drei Zahlenwerte beschrieben, welche die Farbintensitäten in Rot, Grün und Blau angeben. Für das Pixel $(1, 1)$ enthält zum Beispiel $\mathbf{I}(1, 1, 1)$ den Wert des Rotanteils, $\mathbf{I}(1, 1, 2)$ den Wert des Grünanteils und $\mathbf{I}(1, 1, 3)$ den Wert des Blauanteils.

Um das Bild zu komprimieren, berechnen wir für jeden Farbanteil die zweidimensionale Fouriertransformation. Dann werden Frequenzen mit kleinen Koeffizienten eliminiert, indem wir diese Koeffizienten auf 0 setzen. Dadurch kann der Speicheraufwand verringert werden, ohne dass das Bild wesentlich anders aussieht.

Lade das Skript `image_compression.m` und die Matrizen `I1.txt`, `I2.txt` und `I3.txt` von der Webseite herunter. Das Skript enthält einen unvollständigen Code, welcher die Fouriertransformation verwenden soll, um ein Bild zu komprimieren.

a) Wende die Fourier-Transformation zuerst auf jede Spalte und dann auf jede Zeile der Matrix `I(:, :, k)` für jeden Farbkanal $k = 1, 2, 3$ an. So erhältst du die Koeffizienten deines Bildes im Frequenzbereich. Speichere diese Werte in einer Matrix $\mathbf{F} \in \mathbb{C}^{m \times n \times 3}$ ab.

   Hinweis: Beachte, dass deine Funktion `fast_fourier_transform` nur den Koeffizientenvektor $\boldsymbol{\gamma}$ für ein gegebenes $\mathbf{y}$ ausgibt. Dies entspricht nicht der Fouriertransformation! In der Beilage steht alles, was du brauchst. Beachte ausserdem, dass das Bild komplexwertig ist, nachdem du die Spalten transformiert hast.

b) Finde nun mit dem Befehl `find` alle Einträge in $\mathbf{F}$, welche *betragsmässig* kleiner als $\epsilon = 10^{-4}$ sind. Setze all diese Einträge auf 0. Speichere die Anzahl dieser Einträge in der Variablen `number_of_removed_coefficients` ab.

c) Führe zuletzt die Fourier-Synthese durch, um vom Frequenzraum wieder in den Ortsraum zu wechseln. Dabei arbeitest du wie vorher zuerst spalten- und dann zeilenweise.

d) Wiederhole diese Aufgabe für $\epsilon = 10^{-3}$ und $\epsilon = 10^{-5}$ und erkläre deine Beobachtungen. Berücksichtige dabei den Output „memory saved: ...“, der von deiner Funktion erzeugt wird.
