Praktikum II "Numerik am Computer" FS 2022
Prof. Dr. H. Harbrecht Universität Basel
J. Bopp, A. Bot, R. Dorau, M. Fallahpour, A. Heinimann, F. Kruse

---

Serie 6

The Fast and the Fourier Transform

zur 15. und 16. KW (11.04.–22.04.2022)

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)?

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

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

Aufgabe 6.2 (1.5+0.5+1.5+0.5+1.5 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).
Achtung: Der erste Index eines Vektors in Matlab ist 1. Passe deshalb den Algorithmus von der Beilage entsprechend an.

b) Um deine Fuktion aus 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 $k = 0, \dots, n-1$ und $x_k = 2\pi k/n$. Vergleiche für $n = 2^1$ und $n = 2^3$ deinen Vektor $\boldsymbol{\beta}$ mit dem Output vom Matlab-Befehl `1/n*fft(y)`. Die Werte sollten numerisch übereinstimmen.

c) Schreibe zwei Funktionen

`[a,b] = trigon_coeff(beta,n)` und `q = trigon_interpol(x,a,b,n)`,

welche die Koeffizienten a_ℓ und b_ℓ für das Interpolationspolynom $P(x)$ aufstellen und es nach Gleichung (1) auf der Beilage in x auswerten. Dabei soll es wieder möglich sein, x als Vektor zu übergeben.
Hinweis: Nach der Berechnung der Koeffizienten a_ℓ und b_ℓ können diese durch Rundungsfehler noch sehr kleine (im Betrag kleiner als 10^{-15}) Imaginärteile enthalten. Diese kannst du entfernen, indem du einfach die Koeffizienten auf ihren Realteil reduzierst (MATLAB: `real`).

d) Um deine beiden Funktionen zu testen, kannst du wieder die Sägezahn-Funktion aus Teil b) verwenden.

   i) Berechne für $n = 2^2$ die Koeffizienten a_ℓ und b_ℓ (Lösung: $a_0 = a_2 \approx -1.5708, b_1 = a_1 = -a_0$.) Berechne $P(x_k)$ für $k = 0, \ldots, n-1$. Stimmen die Werte jeweils numerisch 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.

e) Wir wollen uns nun noch näher mit dem Aufwand der FFT beschäftigen.

   i) Schreibe eine Funktion

   `beta = beta_coeff(y,n)`,

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

   ii) Wiederhole Teilaufgabe b), um deine Funktion `beta_coeff` zu testen.

   iii) Vergleiche den Aufwand deiner beiden Funktionen, indem du $\boldsymbol{\beta}$ für $n = 2^j$, $j = 1, \ldots, 12$, von deinen Funktionen `beta_coeff` und `fast_fourier_transform` berechnen lässt und misst, welche Funktion wieviel Zeit für jedes einzelne n braucht. Du kannst dies auf folgende Weise tun:

```
tic
beta = beta_coeff(y,n);
t(j) = toc;
```

   Dadurch wird zum Beispiel die Zeit, die `beta_coeff` im j-ten Schritt benötigt, im j-ten Eintrag des Vektors $\mathbf{t}$ gespeichert. Mit der Sägezahn-Funktion kannst du dir für jedes n die Stützpunkte generieren lassen. Zeichne die Zeiten, die du erhalten hast, mit `semilogy` in einer Grafik. Zeichne auch die Vergleichsgeraden gegeben durch $t_1 \cdot 2^j$ und $t_1 \cdot (2^j)^2$, $j = 1, \ldots, 12$. Was fällt dir bezüglich der Vergleichsgeraden auf?

Aufgabe 6.3 (7×0.5 Punkte): Die Fourier-Transformation kann zur Kompression von Bildern benutzt werden. Ein Bild kann zum Beispiel als Tensor $\mathbf{I} \in \mathbb{R}^{m \times n \times 3}$ dargestellt werden. Jedes Pixel hat einen Index (i, j), $i = 1, \ldots, m$, $j = 1, \ldots, n$. Jedem Pixel werden drei Zahlenwerte zugeordnet, die für die Farben codieren. Für

das Pixel $(1,1)$ enthält also $\mathbf{I}(1,1,1)$ 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, die nur einen kleinen Koeffizienten haben, eliminiert, indem wir diese Koeffizienten auf 0 setzen. Dadurch kann der Speicheraufwand verringert werden, ohne dass das Bild wesentlich anders aussieht.

a) Lade von der Webseite die Matrizen `I1.txt`, `I2.txt` und `I3.txt` herunter. Sie enthalten die Rot-, Grün- und Blauanteile eines Bildes. Lade die drei 512×512 Matrizen in Matlab mit den Befehlen `load('I1.txt')`, `load('I2.txt')`, `load('I3.txt')`.

b) Generiere einen Tensor $\mathbf{I} \in \mathbb{R}^{512\times512\times3}$, der in $\mathbf{I}(:,:,k)$ jeweils `Ik` enthält, $k = 1,2,3$. Visualisiere dein Bild mit dem Befehl `imshow(I)`.

c) Für jedes `Ik`, $k = 1,2,3$, kannst du die Fourier-Transformation zuerst auf jede Spalte und dann auf jede Zeile anwenden. So erhältst du die Koeffizienten deines Bildes im Frequenzbereich. Speichere diese Werte in einer Matrix $\mathbf{F} \in \mathbb{R}^{512\times512\times3}$ ab.

d) Finde nun in $\mathbf{F}$ mit dem Befehl `find` alle Einträge, die betragsmässig kleiner sind als $\epsilon = 10^{-5}$. Setze all diese Einträge auf 0.

e) Führe nun die Fourier-Synthese durch, um vom Frequenzraum wieder in den Ortsraum zu wechseln. Dies geht wie vorher durch das Anwenden der Fourier-Synthese auf die Spalten und dann auf die Zeilen deiner drei Matrizen.

f) Visualisiere wiederum dein Bild. Was beobachtest du?
Hinweis: Da durch Rundungsfehler noch kleine Imaginäranteile auftauchen können, reduziere deine Werte auf ihren Realteil, bevor du das Bild visualisierst.

g) Wiederhole die Aufgabe für $\epsilon = 10^{-4}$ und $\epsilon = 10^{-3}$. Schau dir wiederum die Bilder an. Berechne ausserdem für alle drei ϵ, wie stark du dein Bild komprimieren kannst, indem du ausrechnest, wieviel Prozent der Pixel auf 0 gesetzt werden.

Allgemeine Informationen zum Praktikum befinden sich auf der Webseite
`http://cm.dmi.unibas.ch/teaching/praktikumII/praktikumII.html`