Projekt.

Bearbeiten bis: **Montag, 31.07.2023**

Gegeben sei eine Menge $\Omega \subset \mathbb{R}^n$ und eine Menge von Punkten $X := \{\mathbf{x}_0, \dots, \mathbf{x}_{N-1}\} \subset \Omega$. Weiter sei $k : \Omega \times \Omega \to \mathbb{R}$ eine Kernfunktion, welche abseits der Diagonale hinreichend schnell abklingt. Wir sind daran interessiert, die Kernmatrix

$$\mathbf{K} := \begin{bmatrix} k(\mathbf{x}_0, \mathbf{x}_0) & \dots & k(\mathbf{x}_0, \mathbf{x}_{N-1}) \\ \vdots & & \vdots \\ k(\mathbf{x}_{N-1}, \mathbf{x}_0) & \dots & k(\mathbf{x}_{N-1}, \mathbf{x}_{N-1}) \end{bmatrix} \in \mathbb{R}^{N \times N} \tag{1}$$

mit einem schnellen Verfahren zu approximieren. Anders als bisher greifen wir dabei jedoch nicht auf hierarchische Matrizen zurück, sondern verwenden einen geschickten Basiswechsel für $\mathbf{K}$, so dass viele Einträge sehr klein und daher vernachlässigbar werden, weshalb nur noch $\mathcal{O}(N \log N)$ Einträge übrig bleiben. Die wichtigste Zutat für dieses Verfahren sind *Samplets* [2], welche orthogonale Wavelets auf diskreten Punkten sind. Der Einfachheit halber wollen wir uns auch hier auf das Einheitsintervall beschränken. Deshalb wählen wir $\Omega := I := [0, 1)$ und betrachten $N = 2^J$ dyadische Punkte $x_{J,\ell} = 2^{-J}\ell$, $\ell = 0, \dots, 2^J - 1$, wobei wir die Indizes wieder in den Clustern

$$\nu_{j,\ell} := \left\{ 2^{J-j}\ell, \dots, 2^{J-j}(\ell+1) - 1 \right\}, \qquad \ell = 0, \dots, 2^j - 1$$

zusammenfassen. Die Samplet-Konstruktion funktioniert prinzipiell allerdings auch auf beliebigen Punktverteilungen, vergleiche [2].

Samplets

Mit jedem Punkt $x \in X$ assoziieren wir das *Dirac*-Mass δ_x, definiert durch

$$\int_I f(t) \, \mathrm{d}\delta_x(t) := f(x).$$

Zugehörig zu X definieren wir den Vektorraum $\mathcal{X} := \operatorname{span}\{\delta_{x_0}, \dots, \delta_{x_{N-1}}\} \subset C(I)'$, ausgestattet mit dem Innenprodukt

$$\langle u, v \rangle_{\mathcal{X}} := \sum_{\ell=1}^{N} u_\ell v_\ell, \qquad \text{wobei } u = \sum_{\ell=1}^{N} u_\ell \delta_{x_\ell}, \quad v = \sum_{\ell=1}^{N} v_\ell \delta_{x_\ell}.$$

Damit lässt sie die Kernmatrix aus (1) schreiben als

$$\mathbf{K} = \left[\langle k, \mu \otimes \nu \rangle_{I \times I} \right]_{\mu,\nu \in \mathcal{X}}, \qquad \langle k, \mu \otimes \nu \rangle_{I \times I} := \int_I \int_I k(x, y) \, \mathrm{d}\mu(x) \, \mathrm{d}\nu(y).$$

Auf dem Raum $\mathcal{X}$ wollen wir nun eine Multiskalenbasis einführen. Wir suchen dazu Räume $\mathcal{X}_0 \subset \mathcal{X}_1 \subset \dots \subset \mathcal{X}$ mit zugehörigen Orthonormalbasen $\Phi_j := \{\phi_{j,\ell} : \ell \in \mathcal{I}_j\}$, wobei $\mathcal{I}_j$ ein passendes Indexset sei mit $|\mathcal{I}_j| = 2^j$. Jedes Skalierungsmass $\phi_{j,\ell}$ ist dabei eine Linearkombination der Dirac-Masse in $\mathcal{X}$. Da $\mathcal{X}_{j-1} \subset \mathcal{X}_j$ gilt, existiert ein orthogonales Komplement $\mathcal{S}_j$, so dass

$$\mathcal{X}_j = \mathcal{X}_{j-1} \oplus \mathcal{S}_j, \qquad \mathcal{X}_{j-1} \perp \mathcal{S}_j$$

ist. Dabei existiert auch für $\mathcal{S}_j$ eine Basis

$$\Sigma_j := \{\sigma_{j,\ell} : \ell \in \mathcal{I}_j^\Sigma\}, \qquad \mathcal{I}_j^\Sigma := \mathcal{I}_j \setminus \mathcal{I}_{j-1}.$$

Rekursiv gilt also $\mathcal{X} =: \mathcal{X}_J = \mathcal{X}_0 \oplus \mathcal{S}_1 \oplus \mathcal{S}_2 \oplus \dots \oplus \mathcal{S}_J$, wobei

$$\Sigma := \Phi_0 \cup \Sigma_1 \cup \Sigma_2 \cup \dots \cup \Sigma_J$$

eine Basis für $\mathcal{X}$ ist.

Verschwindende Momente

Wir haben bisher nicht vorgegeben, wie die Masse $\sigma_{j,\ell}$, welche wir fortan als Samplets bezeichnen, genau definiert sein sollen. Wir haben lediglich gefordert, dass sie Linearkombination aus Dirac-Massen in $\mathcal{X}$ sein sollen. Deshalb ist es möglich, die Beziehung

$$\int_I p(t)\,\mathrm{d}\sigma_{j,\ell}(t) = 0 \qquad \text{für alle Polynome } p \text{ vom Grad} \leq q$$

für ein $q \in \mathbb{N}_0$ zu fordern. Wir sagen dann, dass die Samplets $q+1$ *verschwindende Momente haben*. Allgemein können wir jede hinreichend glatte Funktion als Taylor-Reihe schreiben, womit das Taylor-Polynom für diese Samplets verschwindet, was in der Fehleranalysis entscheidend ist.

Wir wollen im Folgenden den Fall $q = 0$, welcher in einer *Haar*-Basis resultiert, betrachten. Für die Konstruktion von Samplets mit mehreren verschwindenen Momenten sei auf [2, 3] verwiesen. Wir definieren $\phi_{J,\ell} := \delta_{x_\ell}$ für $\ell \in \mathcal{I}_J := \{0, \dots, N-1\}$ und rekursiv

$$\phi_{j-1,\ell} := \frac{\phi_{j,2\ell} + \phi_{j,2\ell+1}}{\sqrt{2}}, \qquad \ell \in \mathcal{I}_{j-1} := \{0, \dots, 2^{j-1}-1\}, \tag{2}$$

$$\sigma_{j,2^{j-1}+\ell} := \frac{\phi_{j,2\ell} - \phi_{j,2\ell-1}}{\sqrt{2}}, \qquad \ell \in \mathcal{I}_{j-1}. \tag{3}$$

Insbesondere gilt also

$$\begin{bmatrix} \phi_{j-1,0} \\ \phi_{j-1,1} \\ \vdots \\ \phi_{j-1,2^{j-1}-2} \\ \phi_{j-1,2^{j-1}-1} \\ \hline \sigma_{j,2^{j-1}} \\ \sigma_{j,2^{j-1}+1} \\ \vdots \\ \sigma_{j,2^j-2} \\ \sigma_{j,2^j-1} \end{bmatrix} = \underbrace{\frac{1}{\sqrt{2}} \left[\begin{array}{cccccccccc} 1 & 1 & & & & & & & & \\ & & 1 & 1 & & & & & & \\ & & & & \ddots & \ddots & & & & \\ & & & & & & 1 & 1 & & \\ & & & & & & & & 1 & 1 \\ \hline 1 & -1 & & & & & & & & \\ & & 1 & -1 & & & & & & \\ & & & & \ddots & \ddots & & & & \\ & & & & & & 1 & -1 & & \\ & & & & & & & & 1 & -1 \end{array}\right]}_{=:\mathbf{T}_{j-1}} \begin{bmatrix} \phi_{j,0} \\ \phi_{j,1} \\ \vdots \\ \phi_{j,2^j-2} \\ \phi_{j,2^j-1} \end{bmatrix}.$$

Ausgehend von $j = J$ wird aus dieser Beziehung rekursiv die Samplet-Basis gebildet über die Anwendung der Transformationsmatrizen $\mathbf{T}_{J-1}, \mathbf{T}_{J-2}, \dots, \mathbf{T}_0$ auf den richtigen Abschnitt des Vektors.

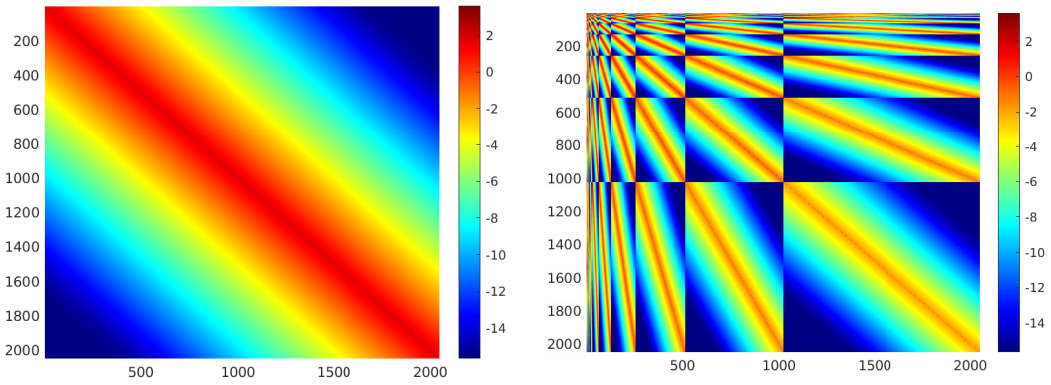
Aufgabe 1. Schreiben Sie eine Funktion

```
function T = haarTransform(J),
```

welche die Transformationsmatrix für die Haar-Basis aufstellt. Benutzen Sie die Funktion `vismat`, um sowohl die Ausgangsmatrix $\mathbf{K}$ als auch die Samplet-Matrix $\mathbf{S} = \mathbf{T}\mathbf{K}\mathbf{T}^\mathsf{T}$ in einer logarithmischen Farbskala zu visualisieren. Verwenden Sie dafür den Exponentialkern mit Korrelationslänge $\sigma = 0.02$ sowie das Level $J = 11$.

Nun wollen wir ausnutzen, dass der Kern schnell abklingt und daher für einen grossen Abstand $|x - y|$ hinreichend glatt ist. Deshalb werden viele Einträge sehr klein, sofern die zugehörigen Samplets einen grossen Abstand besitzen. Deshalb kann man ein Kompressionsschema bauen, das derartige Einträge vernachlässigt. Konkret gilt

$$\operatorname{supp} \sigma_{j,2^{j-1}+\ell} = \operatorname{supp} \phi_{j-1,\ell} \subset I_{j-1,\ell}, \tag{4}$$



wobei $I_{j,\ell}$ das mit dem Cluster $v_{j,\ell}$ assoziierte Intervall bezeichnet. Wir setzen im Folgenden jeden Eintrag $\langle k,\ \sigma_{j,\ell} \otimes \sigma_{j',\ell'} \rangle_{I \times I}$ zu Null, welcher der Zulässigkeitsbeziehung

$$\operatorname{dist}\left(I_{j-1,\ell},\ I_{j'-1,\ell'}\right) \geq \eta \max\left\{2^{-j+1},\ 2^{-j'+1}\right\} \tag{5}$$

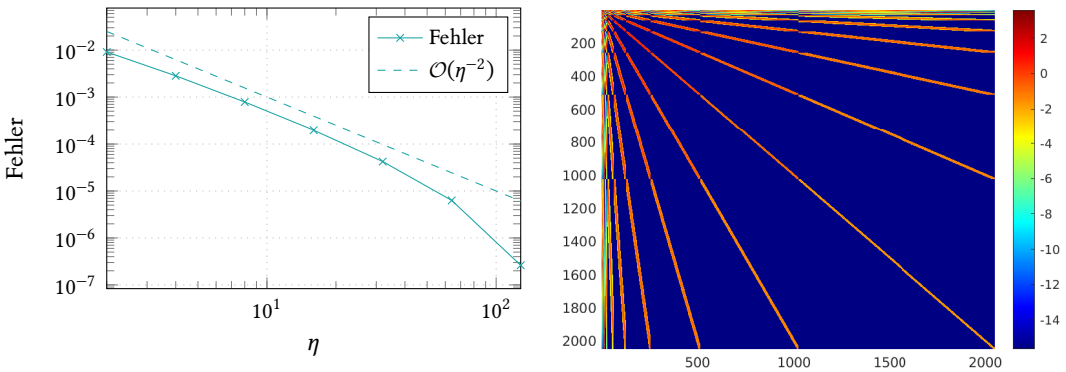
genügt. Für die komprimierte Matrix gilt derweil die Fehlerabschätzung

$$\|\mathrm{S} - \mathrm{S}^{\eta}\|_{\mathrm{F}} \leq C\eta^{-2}\|\mathrm{S}\|_{\mathrm{F}}.$$

Aufgabe 2. Schreiben Sie eine Funktion

```
function S = sampletCompression(S, eta),
```

welche eine gegebene, transformierte Matrix S gemäss (5) komprimiert. Messen Sie für $J = 11$, $\eta = 2^m$, $m = 1, 2, \ldots, 7$ den Fehler und stellen Sie ihn in einem *loglog*-Plot gegen η dar. Zeichnen Sie zusätzlich die Vergleichsgerade $\mathcal{O}(\eta^{-2})$ ein.



Direkte Assemblierung

Es lässt sich zeigen, dass die resultierende Matrix insgesamt nur $\mathcal{O}(N \log N)$ nichttriviale Einträge enthält. Wir haben bisher aber lediglich die volle Matrix aufgestellt und transformiert, womit wir mindestens N^2 Operationen benötigt haben. Wenn man beim Aufstellen rekursiv vorgeht, kann die Matrix auch in $\mathcal{O}(N \log N)$ Operationen augestellt werden, vgl. [1, 2]. Wichtig dabei ist das folgende Resultat:

Aufgabe 3. Zeigen Sie: Sind $v_{j,\ell}$ und $v_{j',\ell'}$ zwei zulässige Cluster, so sind auch alle Paare $(v_{j,\ell},\ v_{j'+1,2\ell'+i'})$, $(v_{j+1,2\ell+i},\ v_{j',\ell'})$ sowie $(v_{j+1,2\ell+i},\ v_{j'+1,2\ell'+i'})$ für alle $(i,i') \in \{0,1\}^2$ zulässig.

Wir betrachten zuerst ein festes Level $1 \leq j \leq J-1$. Aus (4) folgt, dass jedes Cluster auf dem Level $j-1$ mit dem Träger von genau einem Samplet auf Level j sowie einem Skalierungsmass auf Level $j-1$ identifiziert werden kann. Definieren wir nun die Matrizen

$$\mathbf{K}_{(j,\ell),(j',\ell')} \coloneqq \begin{bmatrix} \langle k,\ \phi_{j-1,\ell} \otimes \phi_{j'-1,\ell'} \rangle_{I\times I} & \langle k,\ \phi_{j-1,\ell} \otimes \sigma_{j',2^{j'-1}+\ell} \rangle_{I\times I} \\ \langle k,\ \sigma_{j,2^{j-1}+\ell} \otimes \phi_{j'-1,\ell'} \rangle_{I\times I} & \langle k,\ \sigma_{j,2^{j-1}+\ell} \otimes \sigma_{j',2^{j'-1}+\ell} \rangle_{I\times I} \end{bmatrix}, \tag{6}$$

so folgen aus den Rekursionsgleichungen (2) und (3) die Rekursionen

$$\mathbf{K}_{(j,\ell),(j',\ell')} = \frac{1}{\sqrt{2}} \begin{bmatrix} \mathbf{K}^{(1,:)}_{(j+1,2\ell),(j',\ell')} + \mathbf{K}^{(1,:)}_{(j+1,2\ell+1),(j',\ell')} \\ \mathbf{K}^{(1,:)}_{(j+1,2\ell),(j',\ell')} - \mathbf{K}^{(1,:)}_{(j+1,2\ell+1),(j',\ell')} \end{bmatrix}, \tag{7}$$

$$\mathbf{K}_{(j,\ell),(j',\ell')} = \frac{1}{\sqrt{2}} \left[\mathbf{K}^{(:,1)}_{(j,\ell),(j'+1,2\ell')} + \mathbf{K}^{(:,1)}_{(j,\ell),(j'+1,2\ell'+1)},\ \mathbf{K}^{(:,1)}_{(j,\ell),(j'+1,2\ell')} - \mathbf{K}^{(:,1)}_{(j'+1,2\ell'+1)} \right]. \tag{8}$$

Kombiniert man diese beiden Gleichungen, so gilt auch noch

$$\mathbf{K}_{(j,\ell),(j',\ell')} = \frac{1}{2} \begin{bmatrix} \kappa^0_0 + \kappa^0_1 + \kappa^1_0 + \kappa^1_1 & \kappa^0_0 - \kappa^0_1 + \kappa^1_0 - \kappa^1_1 \\ \kappa^0_0 + \kappa^0_1 - \kappa^1_0 - \kappa^1_1 & \kappa^0_0 - \kappa^0_1 - \kappa^1_0 + \kappa^1_1 \end{bmatrix}, \quad \kappa^m_n = \mathbf{K}^{(1,1)}_{(j+1,2\ell+m),(j'+1,2\ell'+n)}. \tag{9}$$

Mit diesen drei Gleichungen können wir nun rekursiv einen Matrixblock aufstellen.

Algorithmus recursivelyDetermineBlock

Input: Kernfunktion k, Level J, $\eta > 0$, Cluster $v_{j,\ell}$, $v_{j,\ell'}$.
Output: Matrixblock $\mathbf{K}$

1: **if** $j = J-1$ und $j' = J-1$ **then**
2: assembliere den Block $\mathbf{K}_{(j,\ell),(j',\ell')}$ aus (6)
3: **else if** die Cluster sind zulässig gemäss (5) **then**
4: nähere $\mathbf{K}_{(j,\ell),(j',\ell')}$ an
5: **else if** $j = J-1$ **then**
6: berechne $\mathbf{K}_{(j,\ell),(j',\ell')}$ rekursiv aus (8)
7: **else if** $j' = J-1$ **then**
8: berechne $\mathbf{K}_{(j,\ell),(j',\ell')}$ rekursiv aus (7)
9: **else**
10: berechne $\mathbf{K}_{(j,\ell),(j',\ell')}$ rekursiv aus (9)
11: **end if**

In diesem Algorithmus gibt es noch eine Sache zu präzisieren: Sind zwei Cluster zulässig, wollen wir den zugehörigen Matrixblock effizient, aber doch hinreichend genau approximieren, was auf Zeile 4 geschieht. Die einfachste Möglichkeit ergibt sich, wenn man $\phi_{j,2\ell+i} \approx 2^{-j-1}(2\ell+i)$, $i = 0,1$ setzt. Analog geht man für $\phi_{j',2\ell'+i}$ vor. Danach wird der Block wie üblich transformiert.

Diesen Algorithmus verpacken wir nun in einen weiteren Zwischenalgorithmus, welcher eine «Zeile» der Matrix aufstellt. Auch in jenem Algorithmus gilt es, wieder zwei Zeilen zu präzisieren. In der letzten Zeile 17 werden die relevanten Einträge abgespeichert. Da wir die Samplet-Matrix aufstellen wollen, sind das im Falle $j, j' \neq 0$ nur die Samplets, ansonsten benötigen wir eine Fallunterscheidung. Konkret speichern wir

$$\begin{cases} \texttt{(i, j, v)} = (2^j + \ell + 1,\ 2^{j'} + \ell' + 1,\ \mathbf{K}(2,2)), & \text{falls } j > 0,\ j' > 0, \\ \texttt{(i, j, v)} = (2^j + \ell + 1,\ i',\ \mathbf{K}(2,i')), & \text{falls } j > 0,\ j' = 0, \quad i' = 1,2, \\ \texttt{(i, j, v)} = (i,\ 2^{j'} + \ell' + 1,\ \mathbf{K}(i,2)), & \text{falls } j = 0,\ j' > 0, \quad i = 1,2, \\ \texttt{(i, j, v)} = (i,\ i',\ \mathbf{K}(i,i')), & \text{falls } j = 0,\ j' = 0, \quad i,i' = 1,2. \end{cases}$$

Diese Einträge können dann ganz einfach den Matrizen $\mathbf{I}$, $\mathbf{J}$ und $\mathbf{V}$ angehängt werden.

Weiter müssen die Matrixblöcke $\mathbf{K}$ in den Zeilen 11 und 14 in der Matrix $\mathbf{K}_{\texttt{tmp}}$ geladen respektive gespeichert werden. Dafür nehmen wir die Matrix $\mathbf{I}_{\texttt{tmp}}$ zur Hilfe: Muss ein Matrixblock $\mathbf{K}_{(j,\ell),(j',\ell')}$ gespeichert werden, so hängen wir dem Vektor $\mathbf{I}_{\texttt{tmp}}$ den Eintrag $i := (2^{j'} + \ell' - 1)(2^J - 1) + 2^j + \ell$ an, was der Position in der vektorisierten Matrix entspricht. Die Matrix $\mathbf{K}$ speichern wir in $\mathbf{K}_{\texttt{tmp}}(:,\ 2m-1\ :\ 2m)$, wobei m die Länge des Vektors $\mathbf{I}_{\texttt{tmp}}$ bezeichne. Muss der Matrixblock geladen werden, wird entsprechend $\mathbf{I}_{\texttt{tmp}}$ nach i durchsucht und danach kann die Matrix aus $\mathbf{K}_{\texttt{tmp}}$ herausgelesen werden.

Algorithmus setupRow

Input: Kernfunktion k, Level J, $\eta > 0$, Cluster $\nu_{j,\ell}$, $\nu_{j,\ell'}$, Vektoren $\mathbf{I}$, $\mathbf{J}$, $\mathbf{V}$, $\mathbf{I}_{\texttt{tmp}}$, Matrix $\mathbf{K}_{\texttt{tmp}}$
Output: Matrixblock $\mathbf{K}$, Vektoren $\mathbf{I}$, $\mathbf{J}$, $\mathbf{V}$, $\mathbf{I}_{\texttt{tmp}}$, Matrix $\mathbf{K}_{\texttt{tmp}}$

1: **if** $j \neq J-1$ **then**
2: **for** $i = 0, 1$ **do**
3: sind die Cluster $\nu_{j+1,2\ell+i}$ und $\nu_{j',\ell'}$ zulässig, berechne $\mathbf{K}_{(j+1,2\ell+i),(j',\ell')}$ mit *recursivelyDetermineBlock*, ansonsten mit *setupRow*
4: **end for**
5: berechne $\mathbf{K}_{(j,\ell),(j',\ell')}$ gemäss (7)
6: **else**
7: **if** $j' = J-1$ **then**
8: berechne $\mathbf{K}_{(j,\ell),(j',\ell')}$ mit *recursivelyDetermineBlock*
9: **else**
10: **for** $i = 0, 1$ **do**
11: sind die Cluster $\nu_{j,\ell}$ und $\nu_{j'+1,2\ell'+i}$ zulässig oder gilt $j' = J-2$, so berechne $\mathbf{K}_{(j,\ell),(j',2\ell'+i)}$ mit *recursivelyDetermineBlock*, ansonsten finde $\mathbf{K}_{(j,\ell),(j',2\ell'+i)}$ in $\mathbf{K}_{\texttt{tmp}}$
12: **end for**
13: berechne $\mathbf{K}_{(j,\ell),(j',\ell')}$ gemäss (8)
14: füge den Matrixblock $\mathbf{K}_{(j,\ell),(j',\ell')}$ zu $\mathbf{K}_{\texttt{tmp}}$ hinzu
15: **end if**
16: **end if**
17: speichere die relevanten Einträge in $\mathbf{I}$, $\mathbf{J}$ und $\mathbf{V}$

Nun benötigen wir zur Assemblierung noch einen letzten Algorithmus, um eine «Spalte» aufzustellen:

Algorithmus setupColumn

Input: Kernfunktion k, Level J, $\eta > 0$, Cluster $\nu_{j,\ell'}$, Vektoren $\mathbf{I}$, $\mathbf{J}$, $\mathbf{V}$, $\mathbf{I}_{\texttt{tmp}}$, Matrix $\mathbf{K}_{\texttt{tmp}}$
Output: Matrixblock $\mathbf{K}$, Vektoren $\mathbf{I}$, $\mathbf{J}$, $\mathbf{V}$, $\mathbf{I}_{\texttt{tmp}}$, Matrix $\mathbf{K}_{\texttt{tmp}}$

1: **if** $j' \neq J-1$ **then**
2: berechne die Spalten für $\nu_{j'+1,2\ell'}$ und $\nu_{j'+1,2\ell'+1}$ mit *setupColumn*
3: **end if**
4: berechne die Einträge zu $\nu_{0,0}$ und $\nu_{j',\ell'}$ mittels *setupRow*

Es sei angemerkt, dass auch die Speicherkosten durch $\mathcal{O}(N \log N)$ beschränkt sind, wenn man nach Zeile 11 den Matrixblock direkt wieder aus $\mathbf{K}_{\texttt{tmp}}$ löscht, vgl. [1, 2].

Aufgabe 4. Schreiben Sie Funktionen

```
function K = recursivelyDetermineBlock(k, J, eta, j, l, jp, lp),
function [II, JJ, VV, K, Ktmp, Itmp] = setupRow(II, JJ, VV, ...
            k, J, eta, j, l, jp, lp, Ktmp, Itmp),
function [II, JJ, VV, Ktmp, Itmp] = setupColumn(II, JJ, VV, ...
            k, J, eta, jp, kp, Ktmp, Itmp),
```

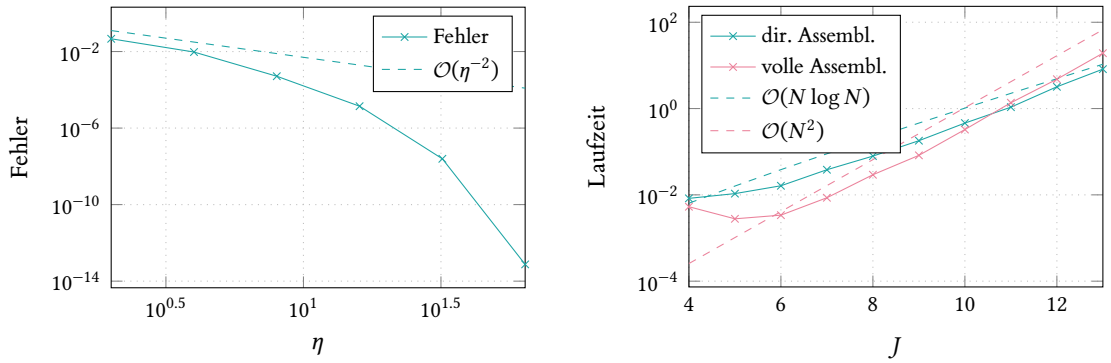
welche die obigen drei Algorithmen implementieren. Schreiben Sie zusätzlich eine Funktion

```
function S = assembleCompressedMatrix(k, J, eta),
```

welche erst die Vektoren $\mathbf{I}$, $\mathbf{J}$ und $\mathbf{V}$ sowie $\mathbf{K}_{\texttt{tmp}}$ und $\mathbf{I}_{\texttt{tmp}}$ initialisiert und danach `setupColumn` für $j', \ell' = 0$ aufruft. Die fertige, komprimierte Samplet-Matrix $\mathbf{S}^\eta$ erhalten Sie schliesslich mit dem Befehl `S = sparse(II, JJ, VV)`.

Aufgabe 5. Schreiben Sie ein Skript, welches analog zu Aufgabe 2 die Fehler für die komprimierte Matrix misst. Benutzen Sie den Exponentialkern mit Korrelationslänge $\sigma = 0.02$ sowie $J = 8$ und $\eta = 2^m$, $m = 1, 2, \dots, 7$.

Aufgabe 6. Schreiben Sie ein Skript, welches für den Exponentialkern mit $\sigma = 0.02$, $\eta = 1.5$ und $j = 4, 5, \dots, 13$ die Laufzeit für die Assemblierung der Systemmatrix misst. Messen Sie dabei die Laufzeit sowohl für die direkte Assemblierung als auch für die volle Assemblierung und anschliessende Transformation und Kompression. Vergleichen Sie die Laufzeiten in einem *semilogy*-Plot und zeichnen Sie zusätzlich die Vergleichsgeraden $\mathcal{O}(N \log N)$ und $\mathcal{O}(N^2)$.



Aufgabe 7. Verfassen Sie einen kurzen, aber prägnanten Projektbericht in LaTeX. Dieser soll die händischen Berechnungen, Konvergenzplots und Visualisierungen aus den Aufgaben beinhalten. Geben Sie für die numerischen Beispiele auch jeweils deren Aufbau an. Abschliessend geben Sie Ihren Code sowie Ihren Bericht in einem *zip*-File namens `projekt-name.zip` ab. Für die nachfolgende Besprechung ist individuell ein Termin zu vereinbaren.

Literatur

[1] D. Alm, H. Harbrecht, and U. Krämer. The $\mathcal{H}^2$-wavelet method. *Journal of Computational and Applied Mathematics*, 267:131–159, 2014.

[2] H. Harbrecht and M. Multerer. Samplets: Construction and scattered data compression. *Journal of Computational Physics*, 471:111616, 2022.

[3] J. Tausch and J. White. Multiscale bases for the sparse representation of boundary integral operators on complex geometry. *SIAM Journal on Scientific Computing*, 24(5):1610–1629, 2003.