Programmierblatt 2.

Bearbeiten bis: **Sonntag, 16.04.2023**

Nachdem wir auf dem ersten Programmierblatt eine Oberfläche aus gestreuten Punkten berechnet haben, wollen wir nun dieses Verfahren beschleunigen. Dazu benötigen wir geeignete Matrixnäherungsverfahren.

Pivotisierte Cholesky-Zerlegung

Ein einfaches Verfahren zur Niedrigrangapproximation einer symmetrischen und positiv semi-definiten Matrix $\mathbf{K} \in \mathbb{R}^{N\times N}$ bietet die pivotisierte Cholesky-Zerlegung. Ziel ist die Berechnung einer linken, unteren Dreiecksmatrix $\mathbf{L} \in \mathbb{R}^{N\times M}$ mit $M \ll N$, so dass $\mathbf{K} \approx \mathbf{L}\mathbf{L}^\intercal$. Da bereits das Aufstellen der Matrix $\mathbf{A}$ für grosse N teuer wird, wollen wir insbesondere nur diejenigen Einträge der Kernmatrix $\mathbf{K}$ berechnen, welche wir wirklich brauchen.

Algorithmus pivotisierte Cholesky-Zerlegung

Input: Kernfunktion $k : \mathbb{R}^s \times \mathbb{R}^s \to \mathbb{R}$, Punkte $\mathbf{x}_1, \dots, \mathbf{x}_N$ Genauigkeit $\texttt{tol} \geq 0$.
Output: $\mathbf{L} \in \mathbb{R}^{N\times M}$ mit $\mathbf{L}\mathbf{L}^\intercal \approx \mathbf{K}$, Permutationsvektor $\mathbf{p}$.

1: setze $\mathbf{p} := \big[1, 2, \dots, N\big]$, $\mathbf{d} := \big[k(\mathbf{x}_j, \mathbf{x}_j)\big]_{j=1}^{M}$, trace $:= \|\mathbf{A}\|_{\mathrm{tr}}$, $M := 1$
2: **while** $M \leq N$ and trace $\geq$ `tol` **do**
3: setze pivot $:= \operatorname{argmax}_{\ell\in\{M,M+1,\dots,N\}} \mathbf{d}_{p_\ell}$ (finde Pivotelement)
4: vertausche $p_{\text{pivot}} \leftrightarrow p_M$
5: setze $\mathbf{L}_{p_M,M} := \sqrt{\mathbf{d}_{p_M}}$ (update $\mathbf{L}$)
6: setze $\mathbf{L}_{p_{M+1:N},M} := \big[k(\mathbf{x}_{p_{M+1}}, \mathbf{x}_{p_M}), \dots, k(\mathbf{x}_{p_N}, \mathbf{x}_{p_M})\big]^\intercal / \mathbf{L}_{p_M,M}$
7: update $\mathbf{L}_{p_{M+1:N},M} := \mathbf{L}_{p_{M+1:N},M} - \mathbf{L}_{p_{M+1:N},1:M-1} \cdot \mathbf{L}^\intercal_{p_M,1:M-1} / \mathbf{L}_{p_M,M}$
8: update $\mathbf{d}_{p_{M:N}} := \mathbf{d}_{p_{M:N}} - \big[\mathbf{L}^2_{p_M,M}, \mathbf{L}^2_{p_{M+1},M}, \dots, \mathbf{L}^2_{p_N,M}\big]^\intercal$ (update $\mathbf{d}$)
9: update trace $:= \|\mathbf{d}_{p_{M:N}}\|_1$ (berechne $\|\mathbf{A} - \mathbf{L}\mathbf{L}^\intercal\|_{\mathrm{tr}}$)
10: setze $M := M + 1$
11: **end while**

Aufgabe 1. Schreiben Sie eine Funktion

```
function [L, p] = pivchol(k, P, tol),
```

welche die pivotisierte Cholesky-Zerlegung implementiert. Achten Sie darauf, dass Sie nicht die ganze Matrix $\mathbf{K}$ assemblieren! Testen Sie Ihre Funktion mit der Kernfunktion

$$k(x, y) = e^{-|x-y|^2}, \qquad \texttt{xs = linspace(0, 1, 4001).}$$

Stellen Sie anschliessend die volle Kernmatrix und die pivotisierte Cholesky-Zerlegung mit Abschneidefehler `tol = 1e-9` auf. Der Fehler in der Frobenius-Norm sollte von der Grössenordnung `1e-10` sein.

Kennen wir eine Niedrigrangapproximation $\mathbf{L}\mathbf{L}^\intercal \approx \mathbf{K}$, können wir damit die Koeffizienten der Levelset-Funktion effizient berechnen. Eine einfache Option bietet hierbei ein CG-Verfahren,

welches für jene Matrix $\mathbf{L}$ das lineare Gleichungssystem $(\mathbf{L}\mathbf{L}^\intercal + \alpha\mathbf{I})\mathbf{x} = \mathbf{b}$ löst. Eine Alternative bietet die Verwendung der Pseudoinverse. Es gilt nämlich

$$\mathbf{L}^+ = (\mathbf{L}^\intercal\mathbf{L})^{-1}\mathbf{L}^\intercal,$$

wobei $\mathbf{L}^\intercal\mathbf{L} \in \mathbb{R}^{M\times M}$ verhältnismässig klein ist. Allerdings müssen wir auch hier wieder regularisieren und verwenden folglich

$$(\mathbf{L}\mathbf{L}^\intercal)^+ \approx \mathbf{L}(\mathbf{L}^\intercal\mathbf{L} + \alpha\mathbf{I})^{-2}\mathbf{L}^\intercal$$

zur Berechnung der Koeffizienten.

Aufgabe 2. Schreiben Sie eine Funktion

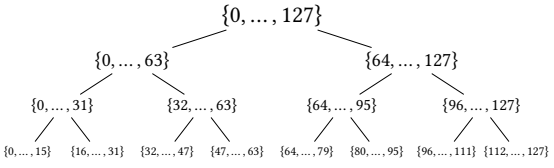
```
function cs = get_coefficients_lowrank(k, p_surf, ...
              p_in, p_out, alpha, cg),
```

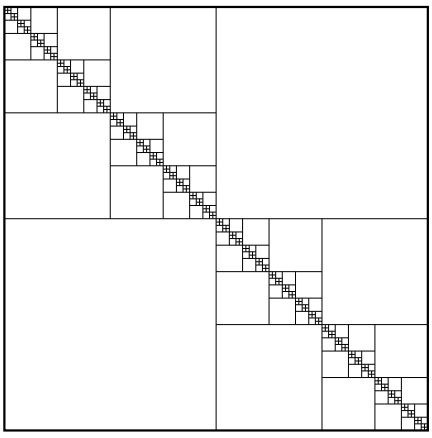
welche die Koeffizienten der Levelset-Funktion per Niedrigrangapproximation berechnet. Dabei sei `cg` eine *logical*-Variable, welche angibt, ob das regularisierte Gleichungssystem mithilfe des CG-Verfahrens oder mithilfe der Singulärwertzerlegung (jeweils mit Regularisierungsparameter α) gelöst werden soll. Visualisieren Sie für alle drei Methoden die `cat`-Geometrie mit $\sigma = 0.6$ und dem Gauß-Kern.

Aufgabe 3. Vergleichen Sie die Laufzeiten der beiden Methoden mit dem \-Solver aus Programmierblatt 1. Wählen Sie dazu verschiedene σ und verschiedene Kerne. Was fällt Ihnen auf?

Blockweise Niedrigrangapproximation

Die pivotisierte Cholesky-Zerlegung ist nur bei glatten Kernen effizient, da sie auf einer globalen Niedrigrangapproximation beruht. Im Fall nichtglatter Kernfunktionen muss die Matrix in geeignete Matrixblöcke zerlegt werden. Dies führt auf das Konzept der hierarchischen Matrizen. Die einzelnen Matrixblöcke werden wir dann mittels *adaptiver Kreuzapproximation (ACA)* annähern.





Der Einfachkeit halber geben wir uns von nun an äquidistante, dyadische Punkte auf dem Einheitsintervall $I := [0, 1)$ vor, das heisst, für $j \in \mathbb{N}$ sei $x_{j,\ell} = \ell \cdot 2^{-j}$, $\ell = 0, \dots, 2^j - 1$. Ist dann $|x - y| \geq C$ hinreichend gross, so ist $k(x, y)$ klein, da der Kern mit dem Abstand der Punkte voneinander abklingt. Gibt es nun Indexmengen $\mu, \nu \subset \{0, \dots, 2^j - 1\}$, so dass $|x_k - x_\ell| \geq C$ für alle $k \in \mu$ und $\ell \in \nu$, so lässt sich der zu μ, ν gehörige Matrixblock gut approximieren. Da allerdings im Allgemeinen der Matrixblock

$$\mathbf{K}_{\mu,\nu} := \left[k(x, y)\right]_{\substack{x\in\mu \\ y\in\nu}}$$

nicht symmetrisch ist, können wir dazu keine pivotisierte Cholesky-Zerlegung mehr verwenden. Abhilfe bietet die adaptive Kreuzapproximation, bei welcher eine Matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ angenähert wird über $\mathbf{A} \approx \mathbf{L}_\ell \mathbf{R}_\ell$, wobei $\mathbf{L} \in \mathbb{R}^{n\times \ell}$ und $\mathbf{R} \in \mathbb{R}^{\ell\times n}$. Deren Berechnung ergibt sich aus dem folgenden Algorithmus. Auch hier ist wieder darauf zu achten, dass nur Matrixeinträge berechnet werden, welche wir wirklich benötigen.

Algorithmus Adaptive Cross Approximation

Input: Matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, Toleranz ε
Output: Matrizen $\mathbf{L}_k = [\boldsymbol{\ell}_1, \dots, \boldsymbol{\ell}_k]$, $\mathbf{R}_k = [\mathbf{r}_1, \dots, \mathbf{r}_k]^\mathsf{T}$

wähle einen zufälligen Pivotindex $i_k \in \{1, \dots, n\}$
for $k = 1, \dots, n$ **do**
    setze $\mathbf{r}_k := \mathbf{a}_{i_k,:}^\mathsf{T} - \sum_{j=1}^{k-1} [\boldsymbol{\ell}_j]_{i_k} \mathbf{r}_j$
    wähle $j_k := \arg\max_i |[\mathbf{r}_k]_i|$
    setze $\mathbf{r}_k := \mathbf{r}_k / [\mathbf{r}_k]_{j_k}$
    setze $\boldsymbol{\ell}_k := \mathbf{a}_{:,j_k} - \sum_{i=1}^{k-1} [\mathbf{r}_i]_{j_k} \boldsymbol{\ell}_i$
    if $\|\boldsymbol{\ell}_k\|_2 \|\mathbf{r}_k\|_2 \le \varepsilon \|\mathbf{L}_k \mathbf{R}_k\|_F$ **then**
        breche ab
    end if
    wähle $i_{k+1} = \arg\max_j |[\boldsymbol{\ell}_k]_j|$ unter der Nebenbedingung,
    dass i_{k+1} noch nie ein Pivotelement war.
end for

Wir wollen uns im Folgenden ein maximales Level $J \in \mathbb{N}$ vorgeben. Durch dyadische Unterteilung des Intervalls erhalten wir auf jedem Level $0 \le j \le J$ die Cluster

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

Anschaulich entspricht $\nu_{j,\ell}$ also dem k-ten Teilintervall des j-mal unterteilten Einheitsintervalls. Da offensichtlich auch $\nu_{j,\ell} = \nu_{j+1,2\ell} \dot\cup \nu_{j+1,2\ell+1}$ gilt, bezeichnen wir $\nu_{j+1,2\ell}$ und $\nu_{j+1,2\ell+1}$ als Söhne von $\nu_{j,\ell}$ und analog dazu $\nu_{j,\ell}$ als deren Vater.

Die Kernmatrix $\mathbf{K}$ lässt sich nun ausgehend vom Wurzelcluster $\nu_{0,0}$ rekursiv berechnen. Es sei angemerkt, dass wir aufgrund der Symmetrie der Kernmatrix in jeder Iteration nur einen Nebendiagonalblock berechnen müssen: Wird dieser Block approximiert durch $\mathbf{LR}$, so wird der andere offensichtlich durch $\mathbf{R}^\mathsf{T}\mathbf{L}^\mathsf{T}$ approximiert.

Algorithmus Blockweise Niedrigrangapproximation

Input: Cluster $\nu_{j,\ell}$, Level J, maximales Level $L < J$, Polynomgrad r, Kernfunktion k
Output: hierarchische Matrix $\mathbf{T}$

if $j = L$ **then**
    assembliere die volle Matrix $\mathbf{K}_{\nu_{j,\ell},\nu_{j,\ell}}$
else
    berechne eine Niedrigrangapproximation $\mathbf{K}_{\nu_{j+1,2\ell},\nu_{j+1,2\ell'+1}} \approx \mathbf{LR}$
    berechne $\mathbf{T}.\texttt{son1} := \mathbf{T}_{\nu_{j+1,2\ell},\nu_{j+1,2\ell}}$ und $\mathbf{T}.\texttt{son2} := \mathbf{T}_{\nu_{j+1,2\ell+1},\nu_{j+1,2\ell+1}}$ rekursiv mit diesem Algorithmus
end if

Die hierarchische Matrix wird in einem Baum abgespeichert, wobei jeder Knoten ein *struct* ist. Jeder Knoten enthält als Feld, gemäss Voraussetzung, `K`, oder aber `L` und `R` sowie `son1` und `son2`. Die *son*-Variablen sind dabei wieder *structs*, und zwar mit den gleichen Feldern.

Aufgabe 4. Schreiben Sie eine Funktion

```
function T = build_Hmatrix(J, ker, tol, L, j, l),
```

welche die blockweise Niedrigrangapproximation der Kernmatrix auf $I \times I$ berechnet. Verwenden Sie zur Berechnung des Nebendiagonalblocks die adaptive Kreuzapproximation und achten Sie darauf, nur Kernauswertungen vorzunehmen, die Sie auch wirklich benötigen.

Haben wir die Kernmatrix aufgestellt, können wir rekursiv das Matrix-Vektor-Produkt einer hierarchischen Matrix $\mathbf{T}$ mit einem Vektor $\mathbf{x} \in \mathbb{R}^{2^J}$ berechnen.

Algorithmus Matrix-Vektor-Multiplikation

Input: hierarchische Matrix $\mathbf{T}$, Vektor $\mathbf{x} \in \mathbb{R}^{2^J}$, maximales Level $L < J$, Level j
Output: $\mathbf{b} = \mathbf{T} \cdot \mathbf{x}$

1: **if** $j = L$ **then**
2: setze $\mathbf{b} = \mathbf{T} \cdot \mathbf{x}$
3: **else**
4: unterteile $\mathbf{b}^\intercal = [\mathbf{b}_1^\intercal, \mathbf{b}_2^\intercal]$ und analog $\mathbf{x}^\intercal = [\mathbf{x}_1^\intercal, \mathbf{x}_2^\intercal]$
5: berechne rekursiv mit diesem Algorithmus

$$\mathbf{b}_1 = \mathbf{T}.\texttt{son1} \cdot \mathbf{x}_1 + \mathbf{L} \cdot (\mathbf{R} \cdot \mathbf{x}_2), \qquad \mathbf{b}_2 = \mathbf{R}^\intercal \cdot (\mathbf{L}^\intercal \cdot \mathbf{x}_1) + \mathbf{T}.\texttt{son2} \cdot \mathbf{x}_2$$

6: **end if**

Aufgabe 5. Schreiben Sie eine Funktion

```
function b = Hmatrix_vector_mult(T, x, L, j),
```

welche die Matrix-Vektor-Multiplikation gemäss dem gegebenen Algorithmus implementiert.

Aufgabe 6. Testen Sie Ihre Implementierung. Benutzen Sie $J = 12$ und $L = 7$, um die hierarchische Matrix $\mathbf{T}$ aufzustellen und stellen Sie zusätzlich die volle Matrix $\mathbf{K}$ auf. Generieren Sie anschliessend $N = 10$ normierte und zentrierte Zufallsvektoren der Länge 2^J. Messen Sie jeweils die Fehler

$$\|\mathbf{T}\mathbf{x} - \mathbf{K}\mathbf{x}\|_2.$$

Benutzen Sie dafür den Gauß-Kern mit Korrelationslänge $\sigma = 1$ und $\texttt{tol} = 10^{-4}$. Ihre Fehler sollten im Bereich 10^{-6} liegen.

Aufgabe 7. Wiederholen Sie Aufgabe 6 für verschiedene Kernfunktionen und verschiedene Korrelationslängen. Wie verhält sich die blockweise Niedrigrangapproximation im Vergleich zur pivotisierten Cholesky-Zerlegung aus Aufgabe 2?

Aufgabe 8. Wiederholen Sie Aufgabe 6 für die Level $J = 8, \ldots, 15$. Messen Sie die Laufzeiten für je 10 Matrix-Vektor-Multiplikationen für die volle und die hierarchische Matrix. Plotten Sie die Laufzeiten in einem *loglog*-Plot gegen die Grösse des Vektors. Was fällt dabei auf?