Computational Statistics
Project Report

20th March 2020

1 Introduction

...

In diesem Projekt reimplementiere ich den vorgeschlagenen Markov Chain Monte Carlo (MCMC) Algorithmus aus dem Paper [Lau and Krumscheid, 2019] und versuche die Ergebnisse der Experimente zu reproduzieren. Der hier betrachtete Paper baut auf den Ideen von [Metropolis et al., 1953, Geyer, 1992, Liu et al., 2000] und [Yang et al., 2019] auf. Der source code unter MIT Lizenz zu diesem Projekt findet sich in dem GitHub repository

`github.com/rinkwitz/Adaptive_Plateau_MCMC`.

Da die Simulation der Markov chains sehr zeitaufwendig ist, kann man schon berechnete Läufe unter dem Link

`drive.google.com/file/d/1MHFLckoims5PYrMWPh2xkQWR0kefofgy/view?usp=sharing`

herunterladen.

2 Adaptive Component-wise Multiple-Try Metropolis Algorithm

Der Kernalgorithmus des Papers [Lau and Krumscheid, 2019] besteht aus einem MCMC Algorithmus der drei Eigenschaften erfüllt. Der Algorithmus schlägt beim sampling mehrere Vorschläge aus verschiedenen Plateauverteilungen vor. Dies passiert unabhängig für alle Komponenten eines samples. Die Plateauverteilungen adaptieren ihre Form in Abhängigkeit von der Frequenz der akzeptierten sample Vorschläge.

2.1 Plateau Proposal Distributions

Der MCMC Algorithmus in [Lau and Krumscheid, 2019] verwendet zum sampling von Vorschlägen *non-overlapping plateau proposal distributions*. Die dabei verwendete grundlegende probability distribution function f ist eine Kombination einer uniform distribution mit exponentiell decaying tails. Dabei ist die Verteilung f konstant um einen Mittelwert μ in dem abgeschlossenen Intervall $[\mu-\delta, \mu+\delta]$ mit $\delta > 0$. Ausserhalb dieses Intervall folgt die Verteilung einem exponentiellen Verfall dessen tail Breite durch ein seitenabhängiges $\sigma_i > 0$ bestimmt wird, je nachdem ob man sich auf der linken oder rechten Seite des Intervalls $[\mu - \delta, \mu + \delta]$ befindet. Definiert man eine unnormalisierte Verteilungsfunktion

$$\tilde{f}(y; \mu, \delta, \sigma_1, \sigma_2) = \begin{cases} \exp\left(-\frac{1}{2\sigma_1^2}[y - (\mu - \delta)]^2\right) & , y < \mu - \delta \\ 1 & , \mu - \delta \leq y \leq \mu + \delta \\ \exp\left(-\frac{1}{2\sigma_2^2}[y - (\mu + \delta)]^2\right) & , y > \mu + \delta \end{cases}$$

und berechnet das folgende Integral

$$\begin{aligned} C(\delta, \sigma_1, \sigma_2) &= \int_{-\infty}^{\infty} \tilde{f}(y; \mu, \delta, \sigma_1, \sigma_2) dy \\ &= \int_{-\infty}^{\mu-\delta} \exp\left(-\frac{1}{2\sigma_1^2}[y - (\mu - \delta)]^2\right) dy + \int_{\mu-\delta}^{\mu+\delta} 1 dy + \int_{\mu+\delta}^{\infty} \exp\left(-\frac{1}{2\sigma_2^2}[y - (\mu + \delta)]^2\right) dy \\ &= \frac{\sqrt{2\pi\sigma_1^2}}{2} + 2\delta + \frac{\sqrt{2\pi\sigma_1^2}}{2} \end{aligned}$$

als die Summen von 2 halben Gausschen Integralen und einem Integral über eine konstatne Funktion, dann ergibt sich die normalisierte Verteilungsfunktion $f(y; \mu, \delta, \sigma_1, \sigma_2) = C(\delta, \sigma_1, \sigma_2)^{-1} \tilde{f}(y; \mu, \delta, \sigma_1, \sigma_2)$ [Lau and Krumscheid, 2019]. Mithilfe von f kann man die Plateau probability density distributions $T_{j,k}$, $j \in \{1, \ldots, M\}$ für die trial proposals der k-ten Komponente definieren als

$$T_{j,k}(x, y) = \begin{cases} f(y; x, \delta_1, \sigma, \sigma) & , j = 1 \\ \frac{1}{2} f(y; x - (2j-3)\delta - \delta_1, \delta, \sigma, \sigma) + \frac{1}{2} f(y; x + (2j-3)\delta + \delta_1, \delta, \sigma, \sigma) & , j = 2, \ldots M-1 \\ \frac{1}{2} f(y; x - (2M-3)\delta - \delta_1, \delta, \sigma_0, \sigma) + \frac{1}{2} f(y; x + (2M-3)\delta + \delta_1, \delta, \sigma, \sigma_1) & , j = M \end{cases}$$

mit Werten $\delta_1, \delta, \sigma, \sigma_0, \sigma_1 > 0$. In Figure 1 sieht man die trial proposal propability density distributions für die Parameter $M = 5, \delta_1 = \delta = 1, \sigma = 0.05, \sigma_0 = \sigma_1 = 0.5$. Man erkennt, dass sich die Verteilungen nur an ihren exponential decaying tails überschneiden. Die äußeren tails fallen durch die größeren σ_0, σ_1 Werte flacher ab als die restlichen tails. In dem Paper [Lau and Krumscheid, 2019] verwenden die Autoren durchgehend die Werte $\delta = \delta_1 = 2, \sigma = 0.05, \sigma_0 = \sigma_1 = 3$.

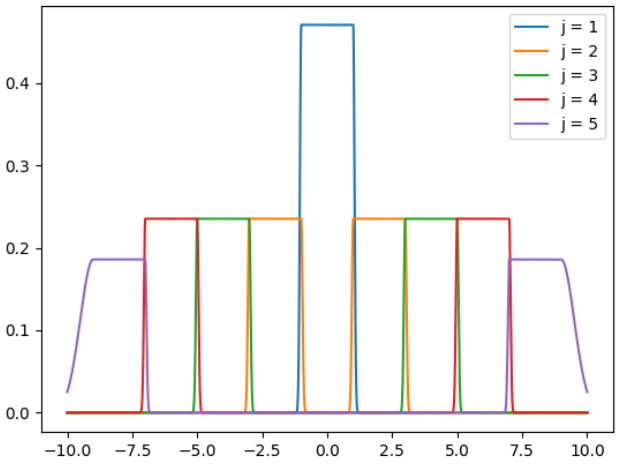


Figure 1: Trial proposal propability density distributions for $M = 5$

2.2 Component-wise Multiple-Try Metropolis

Der *Component-wise Multiple-Try Metropolis* Algorithmus [Lau and Krumscheid, 2019, Algorithm 1], welcher die Grundlage des Sampling Verfahrens bildet, startet von einer Startposition $x_0 \in \mathbb{R}^d$. Für jede MCMC Realisationen x_n mit $n \in \{1, \dots, N\}$ wird für jede der d Komponenten das folgende Verfahren angewendet. Sei $x = (x_1, \dots, x_d)$ der letzte gesampelte Kandidat des MCMC Algorithmus, dann schlägt der Algorithmus trials z_j für $i = 1, \dots, M$ in dem er diese aus den Verteilungen $T_{j,k}(x_k, \cdot)$ sampelt. In meiner Reeimplementierung des Papers verwende ich dazu ein rejection sampling Verfahren [Peng, 2018]. Dabei verwende ich eine gleichförmige Verteilung zum Erzeugen der samples über den folgenden Intervallen

$$I_j = \begin{cases} [x_k - \delta_1 - t_1, x_k + \delta_1 + t_1] \\ \text{for } j = 1 \text{ with } t_1 = \sqrt{-2\sigma^2 \log(0.0001 C(\delta_1, \sigma, \sigma))}, \\ [x_k - 2(j+1)\delta - \delta_1 - t_2, x - 2j\delta - \delta_1 + t_2] \cup [x_k + 2j\delta + \delta_1 - t_2, x + 2(j+1)\delta - \delta_1 + t_2] \\ \text{for } j = 2, \dots, M-1 \text{ with } t_2 = \sqrt{-2\sigma^2 \log(0.0002 C(\delta, \sigma, \sigma))}, \text{ or} \\ [x_k - 2(M+1)\delta - \delta_1 - t_{32}, x_k - 2M\delta - \delta_1 + t_{31}] \cup [x_k + 2M\delta + \delta_1 - t_{31}, x_k + 2(M+1)\delta + \delta_1 + t_{32}] \\ \text{for } j = M \text{ with } t_{31} = \sqrt{-2\sigma^2 \log(0.0002 C(\delta, \sigma, \sigma_0))}, t_{32} = \sqrt{-2\sigma_0^2 \log(0.0002 C(\delta, \sigma, \sigma_0))}. \end{cases}$$

Diese Intervalle ermöglichen es in dem Bereich aus $T_{j,k}(x_k, \cdot)$ effektiv zu samplen wo die probability density function größer als 0.0001 ist. Dazu sampelt man solange ein $u \sim U(0,1)$ und ein $y \sim U(I_i)$ bis

$$u < \frac{T_{j,k}(x_k, y)}{c|I_i|}$$

erfüllt ist wobei $|I_j|$ die Breite des verwendeten Intervalls I_j ist, g_j die probability density function der gleichmäßigen Verteilung über I_j ist und c_j situationsabhängig folgende Werte hat

- $j = 1:\quad c_1 = \sup_{y \in I_1} \frac{T_{1,k}(x_k, y)}{g_1(y)} = \frac{|I_1|}{C(\delta_1, \sigma, \sigma)}$,
- $j = 2:\quad c_j = \sup_{y \in I_j} \frac{T_{j,k}(x_k, y)}{g_j(y)} = \frac{|I_j|}{2C(\delta, \sigma, \sigma)}$, und
- $j = M:\quad c_M = \sup_{y \in I_M} \frac{T_{M,k}(x_k, y)}{g_M(y)} = \frac{|I_M|}{2C(\delta, \sigma, \sigma_0)}$.

Danach werden die zu den trials assoziierten weights

$$w_{j,k} = \pi((z_j; x_{[-k]}))T_{j,k}(x_k, z_j)\lambda_{j,k}(x_k, z_j), \quad \text{for } j = 1, \ldots, M$$

berechnet wobei $(z; x_{[-i]}) \in \mathbb{R}^d$ den Vektor bezeichnet, der in allen Einträgen bis auf den i-ten mit x identisch ist. Der i-te Eintrag nimmt hierbei den Wert z an. Zudem wird die nicht-negative und symmetrische Funktion

$$\lambda_{j,k}(x, y) = |y - x|^{2.5}$$

verwendet. Dabei beziehen sich die Autoren von [Lau and Krumscheid, 2019] auf die Ergebnisse von [Yang et al., 2019]. Ein hohes Gewicht bekommen somit Vorschläge $(z_j; x_{[-k]})$, die eine hohe Wahrscheinlichkeit bezüglichd der Zielverteilung π haben, deren neuer Vorschlag z_j für die k-te Komponente bezüglich $T_{j,k}(x_k, \cdot)$ eine hohe Wahrscheinlichkeit und wegen der potenzierten Abstandsfunktion $\lambda_{j,k}$ weit genug von x_k entfernt ist. Proportional zu den Gewichten $w_{1,k}, \ldots, w_{M,k}$ wird dann ein $y \in \{z_1, \ldots, z_M\}$ zufällig gezogen und für $j = 1, \ldots, M-1$ sampelt der Algorithmus $x_j^* \sim T_{j,k}(y, \cdot)$. Schließlich berechnet man

$$\alpha = \min\left\{1, \frac{w_{1,k}(z_1, x) + \cdots + w_{M,k}(z_M, x)}{w_{1,k}(x_1^*, (y; x_{[-k]})) + \cdots + w_{M-1,k}(x_{M-1}^*, (y; x_{[-k]})) + w_{M,k}(x_k, (y; x_{[-k]}))}\right\}$$

und nimmt mit dieser Wahrscheinlichkeit α den neuen Vorschlag an und setzt $x_n = (y; x_{[-k]})$. Andernfalls verbleibt der Algorithmus bei dem alten Vorschlag und setzt $x_n = x$.

2.3 Adaption of Proposal Distributions

In [Lau and Krumscheid, 2019] wird vorgeschlagen die Breiten δ und δ_0 der Plateauverteilungen adaptiv anzupassen. In meiner Implementierung wird alle L Iterationen gemessen, wie groß die Anzahlen der ausgewählten Vorschläge $c_{j,k}$ von den Plateauverteilungen $T_{j,k}$ in diesem Intervall sind. Wird die mittlere Plateauverteilung überdurschnittlich oft aufgerufen, also $c_{j,k} > L\eta_1$ mit $\eta_1 \in (0, 1)$, dann geht der Algorithmus davon aus, dass die Plateaus zu breit sind und halbiert dementsprechend δ und δ_1 für die folgenden Iterationen. Werden hingegen die äußersten Plateauverteilungen

überdurchschnittlich oft ausgewählt, also $c_{M,k} > \eta_2 L$ mit $\eta_2 \in (0,1)$, dann geht der Algorithmus davon aus, dass die Plateaus zu klein sind und verdoppelt somit δ und δ_1. Die Adaptionen finden nur mit einer immer kleiner werdenden Wahrscheinlichkeit von $\max(0.99^{n-1}, 1/\sqrt{n})$ statt. Diese Implementierung der Adaptionsvorgehensweise basiert auf [Lau and Krumscheid, 2019, Algorithm 2].

3 Experiments and Results

3.1 Performance Measures

3.1.1 Integrated Autocorrelation Times

Die Autoren in dem Paper [Lau and Krumscheid, 2019] verwenden zwei performance measures, um die Wirksamkeit des implementierten Algorithmus zu untersuchen. Dies ist zum einen die integrated autocorrelation times (ACT), welche in R MCMC Simulationen mit jeweils N Schritten für K Komponenten berechnet wird. Sei im Folgenden $X_t^{(r)} = (X_{t,1}^{(r)}, \dots, X_{t,K}^{(r)})$ das Ergebnis der r-ten MCMC Simulation bei Schritt t, wobei $r \in \{1, \dots, R\}$ und $t \in \{1, \dots, N\}$. Die Implementierung benutzt einen *initial positive sequence estimator* wie ihn [Geyer, 1992] verwendet. Dafür wird zunächst die komponentenweise die empirische Autokovarianz bestimmt um die lagged autocovariance $\gamma_{i,k}^{(r)}$ mit $k \in \{1, \dots, K\}$ zu schätzen. Dabei ist

$$\hat{\gamma}_{t,k}^{(r)} = \frac{1}{N} \sum_{i=1}^{N-t} (X_{i,k}^{(r)} - \bar{X_k}^{(r)})(X_{i+t,k}^{(r)} - \bar{X_k}^{(r)})$$

wobei $\bar{X_k}^{(r)} = 1/N \sum_{i=1}^{N} X_{i,k}^{(r)}$ das arithmetische Mittel k-ten Komponente in der r-ten Simulation bezeichnet. Danach schauen wir uns die Summen

$$\hat{\Gamma}_{m,k}^{(r)} = \hat{\gamma}_{2m,k}^{(r)} + \hat{\gamma}_{2m+1,k}^{(r)}$$

von benachbarten Autokovarianzen Paaren an. Schließlich ergibt sich die integrated autocorrelation times als

$$\mathrm{ACT}_k^{(r)} = -\hat{\gamma}_{0,k}^{(r)} + 2 \sum_{i=0}^{m} \hat{\Gamma}_{m,k}^{(r)}$$

wobei m die größte natürliche Zahl ist, sodass $\hat{\Gamma}_{i,k}^{(r)} > 0$ für alle $i \in \{1, \dots, m\}$ gilt [Geyer, 1992]. Schränkt man dies noch weiter ein und fordert zusätzlich noch, dass die Teilfolge $(\hat{\Gamma}_{i,k}^{(r)})_{i=1,\dots,m}$ monoton sein muss, so erhält man einen *initial monotone sequence estimator* [Geyer, 1992]. Dabei zeigt eine kleinere autocorrelation times an, dass aufeinanderfolgende samples eine kleinere Korrelation haben [Lau and Krumscheid, 2019].

3.1.2 Average Squared Jump Distance

Ein weiteres Performance Maß ist die average squared jump distance (ASJD)

$$\text{ASDJ}_k^{(r)} = \frac{1}{N}\sum_{i=1}^{N} |X_{i,k}^{(r)} - X_{i-1,k}^{(r)}|^2$$

für die k-te Komponente und die r-te Wiederholung der Markov chain [Lau and Krumscheid, 2019]. Hierbei sind größere ASJD Werte zu bevorzugen, da sie anzeigen, dass der state-space besser erkundet wird [Lau and Krumscheid, 2019].

3.2 Experiments

3.2.1 Target Distributions

Als Experimente um die Performance der adaptiven plateau-basierten MCMC Methode zu testen, wird in [Lau and Krumscheid, 2019] versucht vier Zelverteilungen mit der Markov chain zu approximieren. Die Zielverteilungen sind dabei

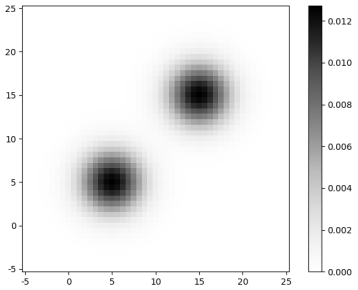
- π_1 : a mixture of 4-dimensional Gaussians $\frac{1}{2}\mathcal{N}(\mu_1, \Sigma_1) + \frac{1}{2}\mathcal{N}(\mu_2, \Sigma_2)$ with $\mu_1 = (5,5,0,0)^T, \mu_2 = (15,15,0,0)^T, \Sigma_1 = \text{diag}(6.25, 6.25, 6.25, 0.01), \Sigma_2 = \text{diag}(6.25, 6.25, 0.25, 0.01)$,
- π_2 : an 8-dimensional banana distribution with density $f \circ \phi$, wobei f die Dichte einer 8-dimensionalen Normalverteilung $\mathcal{N}(0, \Sigma_3)$ mit $\Sigma_3 = \text{diag}(100, 1, \ldots, 1)$ und $\phi(x) = (x_1, x_2 + 0.03x_1^2 - 3, x_3, \ldots, x_8)$ für $x \in \mathbb{R}^8$ ist,
- π_3 : a perturbed 2-dimensional Gaussian mit unnormalisierter Wahrscheinlichkeitsdichte $\tilde{\pi_3}(x) = \exp\left(-x^T A x - \cos\left(\frac{x_1}{0.1}\right) - 0.5\cos\left(\frac{x_2}{0.1}\right)\right)$ für $x \in \mathbb{R}^2$ mit $A = \begin{pmatrix} 1 & 1 \\ 1 & 3/2 \end{pmatrix}$, und
- π_4 : eine 1-dimensional bi-stable Verteilung mit unnormalisierter Verteilung $\tilde{\pi_4}(x) = \exp\left(-x^4 + 5x^2 - \cos\left(\frac{x}{0.02}\right)\right)$ für $x \in \mathbb{R}$.

Diese Verteilungen sind in Figure 2 dargestellt, welche sich an [Lau and Krumscheid, 2019, Figure 3] orientiert. Die Darstellung von π_2 in Figure 2b resultiert aus einer dimensionsreduzierte Vereinfachung, da ein Marginal mit 6 Integrationsvariablen zu rechenintensiv gewesen ist. Dies erkennt man auch daran, dass die Wahrscheinlichkeitsdichte hier größer ist als diejenige in [Lau and Krumscheid, 2019].

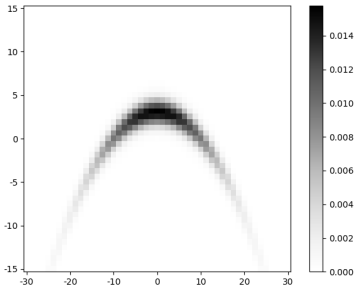
3.2.2 Simulation Parameter Settings

Für die 4 Zielverteilungen werden je $R = 200$ MCMC Simulaitionen durchgeführt mit $N = 4000$ Iterationen für π_1, $N = 10000$ Iterationen für π_2, und $N = 3000$ Iterationen für π_3 und π_4. Die restlichen Parameter sind einheitlich für alle Simulationen. Dies sind

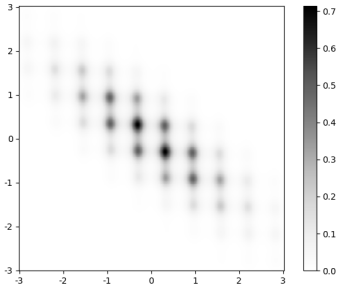
- $M = 5$ verschiedene Plateauvorschlagsverteilungen,



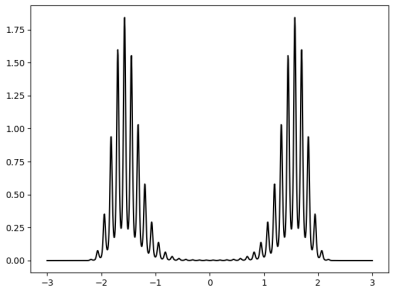
(a) Joint PDF of components 1 and 2 of π_1



(b) Joint PDF of components 1 and 2 of π_2



(c) PDF of π_3



(d) PDF of π_4

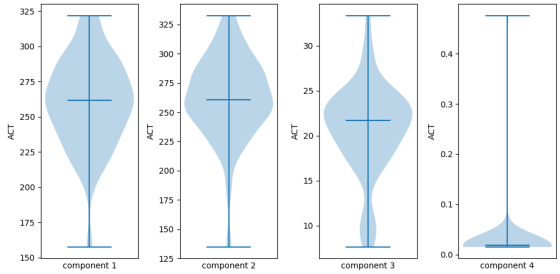
Figure 2: Overview of target distributions

- $\delta = \delta_1 = 2$ Breite der Plateaus,
- $\sigma = 0.05$ und $\sigma_0 = \sigma_1 = 3$ Flachheit der Plateau tails,
- $\eta_1 = \eta_2 = 0.4$ mindest Anteil für eine Adaption,
- $L = 40$ Länge des Intervalls zwischen den Adaptionen, und
- einen burn-in Anteil von 0.5 aller Iterationen.

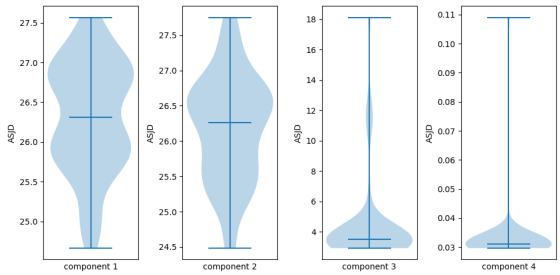
Diese Paramter sind bis auf L, für welchen es keine exakten Angaben gibt, genau wie in [Lau and Krumscheid, 2019] gesetzt.

3.3 Results

Die zu den Experimenten zugehörigen Violinen plots findet man für die Zielverteilungen π_1, π_2, π_3 und π_4 finden sich in den Figures 3, 4, 5 und 6. Dort findet man die komponenteweise Darstellung der ACT Verteilung in den Subfigures 3a, 4a, 5a und 3a, sowie die komponentenweise Darstellung der ASJD Verteilung in den Subfigures 3b, 4b, 5b und 6b. Um die Resultate besser einschätzen zu können sind in den Tabellen 1, 2 und 3 komponentenweise der Median, mean, sowie Minimum und Maximum in gerundeter Form aufgeführt für ACT, log ACT und ASJD der Experimente.



(a) komponenteweise ACT distribution für Zielverteilung π_1



(b) komponenteweise ASJD distribution für Zielverteilung π_1

Figure 3: Violin plots of performance measures for target distribution π_1

	π_1	π_2	π_3	π_4
...				

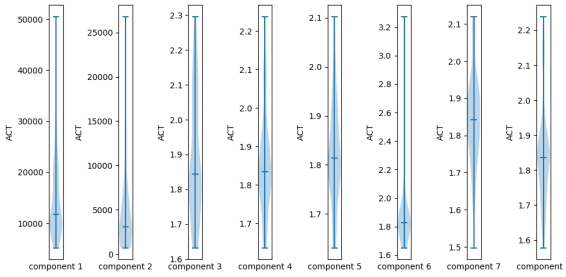
Table 1: Gerundete statistical results of ACT komponentenweise

	π_1	π_2	π_3	π_4
...				

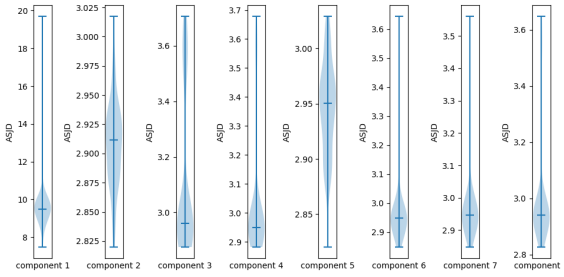
Table 2: Gerundete statistical results of log ACT komponentenweise

...

Table 3: Gerundete statistical results of ASJD komponentenweise

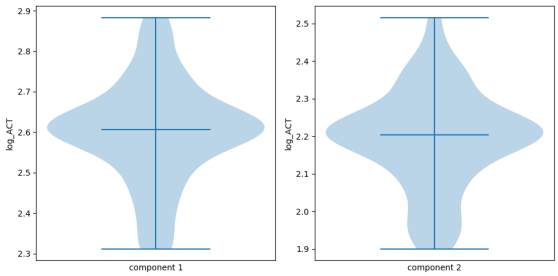


(a) komponenteweise ACT distribution für Zielverteilung π_2

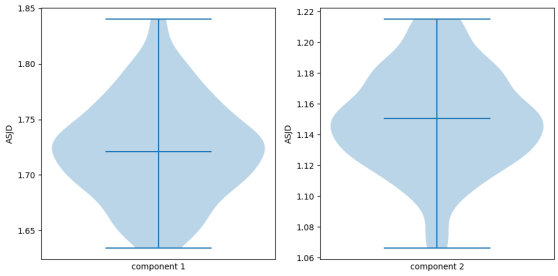


(b) komponenteweise ASJD distribution für Zielverteilung π_2

Figure 4: Violin plots of performance measures for target distribution π_2

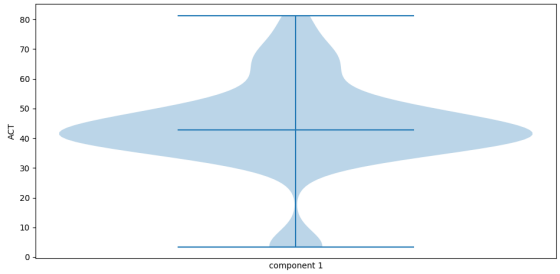


(a) komponenteweise log ACT distribution für Zielverteilung π_3

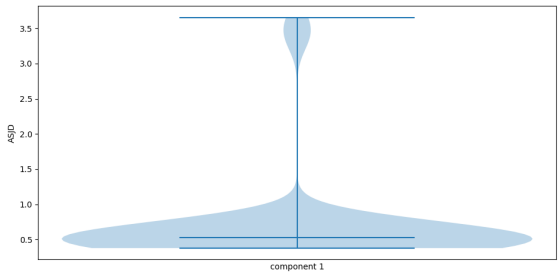


(b) komponenteweise ASJD distribution für Zielverteilung π_3

Figure 5: Violin plots of performance measures for target distribution π_3



(a) komponenteweise ACT distribution für Zielverteilung π_4



(b) komponenteweise ASJD distribution für Zielverteilung π_4

Figure 6: Violin plots of performance measures for target distribution π_4

4 Discussion

TO DO ...

5 Technical Details of Implementation

Die Reimplementierung von [Lau and Krumscheid, 2019] ist in Python 3 geschrieben und es werden die Pakete `Numpy`, `Scipy` und `Matplotlib` benötigt. Die vier Simulationen zur Reproduktion der MCMC Resultate können mit des Skriptes `run_simulations.py` ausgeführt werden, welches alle vier Simulationen parallelisiert mithilfe von `multiprocessing` ausführt. Die einstellbaren Parameter können unter `adjustable parameters` in den Skripten gefunden werden. Dabei folgen sie der Namenskonvention aus diesem Report und dem Paper [Lau and Krumscheid, 2019]. Nachdem die Simulationen ausgeführt wurden, kann mit dem Skript `create_violin_plots.py` die Violinenplots für die Resultate erstellt werden. Zusätzlich gibt es noch die beiden Skripte `create_fig_2b.py` und `create_fig_3.py` mit denen Reproduktionen der Figures 2b und 3 aus [Lau and Krumscheid, 2019] erstellt werden können. Falls man die in der Cloud schon bereitgestellten MCMC Läufe verwenden möchte, so muss man die entpackten `Numpy`-arrays in den Ordner `simulations` kopieren.

References

[Geyer, 1992] Geyer, C. J. (1992). Practical markov chain monte carlo. *Statist. Sci.*, 7(4):473–483.

[Lau and Krumscheid, 2019] Lau, F. D.-H. and Krumscheid, S. (2019). Plateau proposal distributions for adaptive component-wise multiple-try metropolis.

[Liu et al., 2000] Liu, J., Liang, F., and Wong, W. (2000). The multiple-try method and local optimization in metropolis sampling. *Journal of The American Statistical Association - J AMER STATIST ASSN*, 95:121–134.

[Metropolis et al., 1953] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092.

[Peng, 2018] Peng, R. D. (2018). Advanced statistical computing. `https://bookdown.org/rdpeng/advstatcomp/rejection-sampling.html`. Accessed: 2020-03-17.

[Yang et al., 2019] Yang, J., Craiu, R., and Rosenthal, J. (2019). Adaptive component-wise multiple-try metropolis sampling. *Journal of Computational and Graphical Statistics*.