Technische Hochschule Nürnberg

Monte-Carlo-Integration & Unsicherheitsquantifizierung

Deep Dive, Diagnosen, Tricks

Wintersemester 2025/26,
Benedikt Mangold

Unser Plan

1. Monte-Carlo Integration
   - Schätzer, Standardfehler, Konfidenzintervalle
   - Beispiele
   - Was kann schief gehen?
   - Monte-Carlo für Wahrscheinichkeiten (Flächen / Volumen)
   - Replikation und Stabilität
   - Fehlerabschätzung & Stoppkriterien
   - Effizienz
2. Importance Sampling
3. Praktische Fallstricke
   - Unberücksichtigte Abhängigkeit
   - Reporting: Unsicherheit transparent kommunizieren
4. Mini-Demo
5. Ausblick

Literatur

- **SciPy** `scipy.stats` **– Referenz** (Verteilungen, Sampling, Dichten).
  https://docs.scipy.org/doc/scipy/reference/stats.html
- **SciPy Tutorial: Statistics** – kompakter Einstieg in `scipy.stats`.
  https://docs.scipy.org/doc/scipy/tutorial/stats.html
- **Bootstrap-Konfidenzintervalle (STAT 200, Penn State)** – anschauliche Einführung.
  https://online.stat.psu.edu/stat200/lesson/4/4.4
- **SciPy:** `scipy.stats.bootstrap` – API & Beispiele.
  https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.bootstrap.html
- **Monte-Carlo-Integration in Python** – kurzer Theorie-& Praxisartikel.
  https://boyangzhao.github.io/posts/monte-carlo-integration
- **Importance Sampling – anschauliche Einführung mit Python** (Blog/Tutorial).
  https://aleksandarhaber.com/what-is-importance-sampling-explanation-with-python-simulations/

Lernziele (Kapitel 2) i

Aufbauend auf Kapitel 1

Lernziele

- **Vertiefen (Verstehen):** Monte-Carlo-*Integration* als Erwartungswert-Berechnung; Bedingungen für Existenz & Konvergenz (Messbarkeit, Integrabilität).
- **Absichern (Theorie):** LLN & CLT für MC-Mittel; SE-Skalierung $\propto 1/\sqrt{N}$; saubere SE-/CI-Herleitung und richtige Interpretation der CIs.
- **Anwenden (Praxis):** SE schätzen (i.i.d. vs. abhängig), Replikationen & Coverage prüfen, Batch-Means für abhängige Ausgaben.
- **Steuern (Planung):** Stoppkriterien nach ε-Präzision; Stichprobengrößenplanung aus Pilotdaten; Zeit-/Budget-Modell ($T_0 + N\,t_{\text{eval}}$).

Lernziele (Kapitel 2) ii

Aufbauend auf Kapitel 1

Lernziele (fortgesetzt)

- **Effizienz (Verbessern):** Varianzreduktion starten — *Importance Sampling* (Idee, Gewichte, ESS, Diagnostik) für seltene Ereignisse.
- **Reflektieren (Bewerten):** Bias–Varianz-Trade-off (z. B. Trunkierung/Winsorisierung), Nichtlinearitäten ($h(\hat{\mu}) \Rightarrow$ potenzieller Bias; transparentes Reporting ($\hat{\mu} \pm \mathrm{SE}$, CI, *N*, RNG/Seed).

ohm

Monte-Carlo Integration

Voraussetzungen

Voraussetzungen (Minimal)

Sei $X_1, \ldots, X_N \overset{\text{i.i.d.}}{\sim} f$, $g : \mathbb{R}^d \to \mathbb{R}$ messbar. Setze $Y_i = g(X_i)$ mit $\mu = \mathbb{E}[Y]$.

- $\mathbb{E}[|Y|] < \infty$ ($\Rightarrow$ Erwartungswert existiert)
- Für CLT zusätzlich: $\mathrm{Var}(Y_1) = \sigma^2 < \infty$

MC-Integration in einem Satz

Erwartungswerte als Integrale (Fundament)

Zielgröße

Gegeben eine Dichte $f(x)$ und ein messbares[1] g:

$$\mu = \mathbb{E}[g(X)] = \int g(x)\, f(x)\, dx.$$

Monte-Carlo-Schätzer

Ziehe eine Stichprobe $X_1, \ldots, X_N \overset{i.i.d.}{\sim} F$ und setze

$$\hat{\mu}_N = \frac{1}{N} \sum_{i=1}^{N} g(X_i).$$

Dann ist $\mathbb{E}[\hat{\mu}_N] = \mu$ (unverzerrt) und $\mathrm{Var}(\hat{\mu}_N) = \sigma^2/N$ mit $\sigma^2 = \mathrm{Var}[g(X)]$.

[1] Alle stetigen Funktionen sind messbar

Warum funktioniert MC?

Gesetz der großen Zahlen (LLN)

Gesetz der großen Zahlen (LLN)

$$\bar{Y}_N = \frac{1}{N}\sum_{i=1}^{N} Y_i \xrightarrow{\text{f.s.}} \mu \qquad (n \to \infty).$$

Der MC-Mittelwert $\hat{\mu}_N = \bar{Y}_N$ konvergiert fast sicher zum wahren Erwartungswert μ, daher ist MC *konsistent*.

- Schätzungen stabilisieren sich, Fehler heben sich auf.
- Aber *wie schnell*? $\Rightarrow$ Zentraler Grenzwertsatz (CLT).

Warum funktioniert MC?

Zentraler Grenzwertsatz (CLT)

Zentraler Grenzwertsatz (CLT)

Falls $\sigma^2 = \mathrm{Var}[g(X)] < \infty$:

$$\sqrt{N}\,(\bar{Y}_N - \mu) \xrightarrow{d} \mathcal{N}(0, \sigma^2) \quad \Rightarrow \quad \bar{Y}_N \approx \mathcal{N}\Big(\mu,\ \frac{\sigma^2}{N}\Big) \text{ für großes } N.$$

- Asymptotisch gilt: $\mathrm{SE}(\hat{\mu}_n) \approx \sigma/\sqrt{N}$
- Z.B.: Viermal so viele Samples $\Rightarrow$ SE halbiert sich.
- Ist σ unbekannt kann sie durch s geschätzt werden und es ergibt sich das asymptotische 95%-CI: $\hat{\mu}_n \pm 1.96\ \frac{s}{\sqrt{N}}$

Warum funktioniert MC?

Zentraler Grenzwertsatz (CLT)

Schätzung der Varianz

Bei absichtlicher oder unabsichtlicher Verletzung der i.i.d. Annahmen, z.B. Abhängigkeiten[2]
$\Rightarrow$ tatsächliche Varianz größer.

Übung

- Woher kennen Sie den Wert 1.96 von Folie 9?
- Überlegen Sie, welche Auswirkung eine Verletzung der i.i.d Annahme auf ein Konfidenzintervall hätte.

[2] z. B. zeitliche Abhängigkeiten, siehe später

Standardfehler in der Praxis

Schätzung aus den Daten

- Stichprobenvarianz:

$$s^2 = \frac{1}{N-1} \sum_{i=1}^{N} (g(X_i) - \hat{\mu}_N)^2$$

- **SE-Schätzer:**

$$\widehat{\mathrm{SE}} = \frac{s}{\sqrt{N}}$$

- Voraussetzung: (nahezu) unabhängige Ziehungen $\Rightarrow$ sonst Unterschätzung.

Punkt- vs. Intervallschätzung

- Punktschätzung $\hat{\mu}_N$ ist fast sicher nie der tatsächliche, unbekannte Wert.
- Intervallschätzung (asymptotisch, z-Approximation, **C**onfidence **I**ntervall): Asymptotisches $100(1-\alpha)\%$-CI

$$\hat{\mu}_N \pm z_{1-\alpha/2} \cdot \widehat{\mathrm{SE}}$$

- Typisch: $\alpha = 0{,}05 \Rightarrow 95\%$-Intervall mit $z_{0.975} \approx 1{,}96$
- Bei kleinem *N*: t-Heuristik (robust selten nötig bei MC, da *N* groß wählbar).

Interpretation eines CI

Richtig: Ein 95 %-Konfidenzintervall ist ein zufälliges Verfahren, das bei vielen gleichartigen Stichproben in etwa 95 % der Fälle ein Intervall liefert, das den wahren (fixen) Parameter enthält
Falsch: "Mit 95 % Wahrscheinlichkeit liegt der wahre Parameter in *diesem* konkret beobachteten Intervall."

Im Reporting: Immer $\hat{\mu}$ *mit* SE/CI, keine „nackten" Punktwerte

Python-Pattern

Mittel & SE (wiederverwendbar)

```python
import numpy as np

def mc_mean_se(values):
    """Return (mean, standard error) for 1D array."""
    N = values.size
    mu = values.mean()
    se = values.std(ddof=1) / np.sqrt(N)
    return mu, se

# Beispiel: g(x)=exp(-x**2), X~U(0,1)
rng = np.random.default_rng(42)
x = rng.random(200_000)
gx = np.exp(-x**2)
mu, se = mc_mean_se(gx)
print(f"mu={mu:.6f}, 95%CI=({mu - 1.96*se:.6f},{mu + 1.96*se:.6f})")
```

Gutartiges Beispiel

Erwartungswert & Varianz existieren (MC „unkritisch“)

Erwartungswert & Varianz

Sei $X \sim \mathcal{U}(0,1)$ und $g(x) = \sqrt{x}$, Zielgröße: $\mu = \mathbb{E}[g(X)]$. Dann ist

$$\mathbb{E}[\sqrt{X}] =$$

$$\mathrm{Var}(\sqrt{X}) =$$

MC-Praxishinweis

Für $N = 10^5$ ergibt sich $\mathrm{SE} \approx \sqrt{1/(18 \cdot 10^5)} \approx 0.00236$; ein 95%-CI wäre $\hat{\mu} \pm 1.96 \cdot \mathrm{SE}$, typischerweise sehr nahe um $2/3$.

Zielgröße mit existierendem Erwartungswert

Integrabilität & Konvergenz nicht gegeben, weil die Varianz nicht existiert

Setting (Beispiel: Pareto mit endlichem Mittel)

Sei $X \sim \mathrm{Pareto}(x_m = 1, \alpha = 1.5)$ mit Dichte

$$f(x) = \alpha\, x_m^{\alpha}\, x^{-(\alpha+1)} = \tfrac{3}{2}\, x^{-2.5}, \quad x \geq 1.$$

Wir betrachten $g(x) = x$ und damit die Zielgröße $\mu = \mathbb{E}[g(X)] = \mathbb{E}[X]$.

Konvergenz

$$\mathbb{E}[X] =$$

$\int |g(x)|\, f(x)\, dx < \infty \Rightarrow$ Erwartungswert existiert, MC-Schätzer ist sinnvoll.

MC-Integration in einem Satz

Erwartungswerte als Integrale (Fundament)

Monte-Carlo-Schätzer

Ziehe eine Stichprobe $X_1, \ldots, X_N \overset{i.i.d.}{\sim} F$ und setze

$$\hat{\mu}_N = \frac{1}{N} \sum_{i=1}^{N} g(X_i).$$

Dann ist $\mathbb{E}[\hat{\mu}_N] = \mu$ und $\mathrm{Var}(\hat{\mu}_N) = \sigma^2/N$ mit $\sigma^2 = \mathrm{Var}[g(X)]$.

- Der Monte-Carlo Schätzer ist unverzerrt[3], nicht-lineare Transformationen von Funktionen von $\hat{\mu}$ sind es oft nicht[4]
- Die Varianz des Monte-Carlo Schätzers sinkt mit steigendem N, die Schätzgenauigkeit steigt

[3] Siehe nächste Folie

[4] z. B. $\exp(\hat{\mu})$

MSE des MC-Schätzers

Definition, Zerlegung, Konsequenzen

Definition & Zerlegung

Für einen Schätzer $\hat{\theta}$ für θ ist die **mittlere quadratische Abweichung (MSE)**

$$\mathrm{MSE}(\hat{\theta}) = \mathbb{E}\Big[(\hat{\theta} - \theta)^2\Big] = \underbrace{\mathrm{Var}(\hat{\theta})}_{\text{Streuungsanteil}} + \underbrace{\big(\mathbb{E}[\hat{\theta}] - \theta\big)^2}_{\text{Bias}^2}.$$

Beweis:

Bias Variance Tradeoff

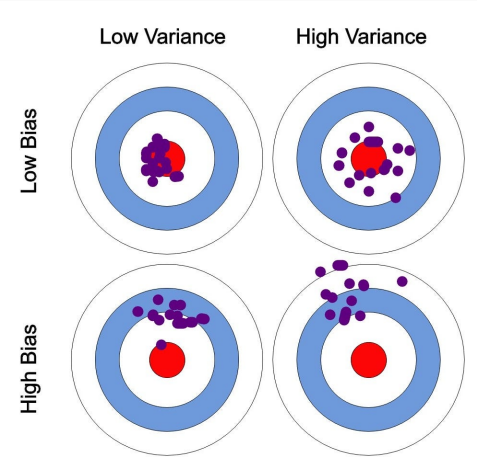


Figure 1: https://towardsdatascience.com/what-is-bias-variance-tradeoff

MSE des MC-Schätzers

Definition, Zerlegung, Konsequenzen

Für den MC-Mittelwert $\hat{\mu}_N = \frac{1}{N}\sum_{i=1}^{N} g(X_i)$

$$\mathbb{E}[\hat{\mu}_N] = \mu, \qquad \mathrm{Var}(\hat{\mu}_N) = \frac{\sigma^2}{N}, \quad \sigma^2 = \mathrm{Var}[g(X)].$$

$$\Rightarrow \mathrm{Bias} = 0 \Rightarrow \boxed{\mathrm{MSE}(\hat{\mu}_N) = \frac{\sigma^2}{N}}, \quad \mathrm{RMSE} = \sqrt{\mathrm{MSE}} = \frac{\sigma}{\sqrt{N}}.$$

Praxisfolgen

Verdoppeln von N senkt RMSE nur um Faktor $\sqrt{2}$ (langsamer Ertragszuwachs).

MC-Integration in einem Satz

$X \sim \mathcal{U}(0,1)$, $g(x) = \sqrt{x}$. **Analytisch:** $\int_0^1 \sqrt{x}\, dx = 2/3$

```python
import numpy
rng = np.random.default_rng(42)

def mc_integral_sqrtx(N, rng):
    x = rng.random(N)    # X_i ~ U(0,1)
    gx = np.sqrt(x) # g(X_i)
    mu_hat = gx.mean()  # MC-Schätzer
    se_hat = gx.std(ddof=1) / np.sqrt(N)  # Standardfehler
    lcl = mu_hat - 1.96*se_hat
    ucl = mu_hat + 1.96*se_hat
    return mu_hat, se_hat, (lcl, ucl)
```

MC-Integration in einem Satz

$X \sim \mathcal{U}(0,1)$, $g(x) = \sqrt{x}$. **Analytisch:** $\int_0^1 \sqrt{x}\, dx = 2/3$

```python
# Analytischer Wert
truth = 2/3

# Einfache Demo für ein N
N = 200_000
mu_h, se_h, (lcl, ucl) = mc_integral_sqrtx(N, seed=123)

print(f"N={N:>7d} | mu_h={mu_h:.6f} | SE={se_h:.6f}")
print(f"95% CI= [{lcl:.6f}, {ucl:.6f}] | truth={truth:.6f}")
```

Zu beachten: In diesem Fall hätte es keine MC-Integration gebraucht, die Lösung ist geschlossen darstellbar.

Wann ist MC-Integration sinnvoll?

Entscheidungshilfen

MC ist ein *any-time* Verfahren: Zwischenstände sind aussagekräftig und lassen sich mit SE/CI kommunizieren.

Gut geeignet

- hohe Dimension, komplizierte Integrationsbereiche
- nur Black-Box-Auswertung von g möglich
- seltene Ereignisse/Indikatoren (mit IS[5])
- Modellunsicherheit[6] statt Messfehler dominiert

Weniger geeignet

- glatte, niedrigdimensionale Integrale $\Rightarrow$ numerische Integrationsverfahren besser
- bekannte geschlossene Formeln
- $\mathrm{Var}[g(X)] = \infty$ (schwere Tails), $\lightning$
- $\mathbb{E}[g(X)] = \infty$ (kein Erwartungswert) $\lightning$

Ist $h(x) = g(x)f(x)$ gutartig integrierbar? Falls nein, $g(X)$ stabilisieren (Transformation/IS).

[5] Importance Sampling
[6] Modellannahmen, Parametern oder Szenarien

Was kann schief gehen?

Pareto-Beispiel: Erwartungswert endlich, Varianz ist unendlich, $\alpha = 1.5,\ x_m = 1$

Zweites Moment (divergent)

$$\mathbb{E}[X^2] =$$

$$\Rightarrow \quad \mathrm{Var}(X) =$$

Allgemeines Kriterium (Pareto)

Für $X \sim \mathrm{Pareto}(x_m, \alpha)$ gilt: $\mathbb{E}[X^k] < \infty \iff \alpha > k$.
Hier: $\alpha = 1.5 > 1 \Rightarrow$ Mittelwert existiert; $\alpha = 1.5 \not> 2 \Rightarrow$ zweiter Moment divergiere $\Rightarrow$ Varianz unendlich.

Was kann schief gehen?

Integrabilität & Konvergenz nicht gegeben, weil die Varianz nicht existiert

Konsequenz für Monte Carlo

LLN gilt (endlicher Mittelwert) $\Rightarrow \hat{\mu}_n \to \mu$ ($n \to \infty$);
CLT scheitert (Varianz ∞) $\Rightarrow$ keine $\hat{\mu} \pm 1.96\, s/\sqrt{n}$-CIs

Bemerkung

Trotz endlichem Mittel ist hier $\mathrm{Var}(X) = \infty$ (weil $\alpha \leq 2$).
$\Rightarrow$ Monte-Carlo-Mittel bleibt unverzerrt und konsistent (LLN), aber CIs via CLT sind heikel; Konvergenz kann langsam/„ruppig“ wirken.

Was kann schief gehen?

Worst Case Szenario: Divergenz & Konsequenz

Beispiel: Cauchy-Verteilung $t-$Verteilung mit einem Freiheitsgrad

$X \sim \text{Cauchy}(0,1)$ mit

$$f(x) = \frac{1}{\pi(1+x^2)}, \qquad g(x) = x$$

Keine Konvergenz

$$\int_{\mathbb{R}} |x| \, \frac{1}{\pi(1+x^2)} \, dx = \frac{2}{\pi} \int_0^{\infty} \frac{x}{1+x^2} \, dx = \frac{2}{\pi} \left[\tfrac{1}{2} \ln(1+x^2) \right]_0^{\infty} = \frac{1}{\pi} \lim_{R \to \infty} \ln(1+R^2) = \infty.$$

Was kann schief gehen?

Worst Case Szenario: Divergenz & Konsequenz

Konsequenz für Monte Carlo

- Wenn $\int |g(x)|\, f(x)\, dx = \infty$, dann ist $\mathbb{E}[g(X)]$ **nicht** definiert
- Ein MC-Mittel $\frac{1}{N} \sum g(X_i)$ hat *keine* wohldefinierte Zielgröße
  - „Schätzer“ ist instabil
  - CIs sinnlos

$\Rightarrow$ Modell/Frage anpassen (z. B. andere Funktionale, Trunkierungen, Transformationen), bis Integrierbarkeit erfüllt ist.

Möglicher Fix: Modell/Frage anpassen, bis Integrierbarkeit gilt

Ausgangsproblem

Problem

Sei

$$X \sim \mathrm{Pareto}(x_m = 1, \alpha = 1)$$

mit Dichte

$$f(x) = \alpha x^{-(\alpha+1)} = x^{-2}$$

für $x \geq 1$. Für $g(x) = x$ gilt

$$\mathbb{E}[X] = \int_1^\infty x \cdot x^{-2}\, dx = \int_1^\infty x^{-1}\, dx = \infty$$

$\Rightarrow$ **nicht integrierbar**.

Möglicher Fix: Modell/Frage anpassen, bis Integrierbarkeit gilt

Beispiel für Transformation

Reparatur A: *Anderes Funktional* wählen (Transformation)

Wähle $g(x) = \log(1+x)$ (wachsend, aber *bounded growth*[7]). Dann

$$\begin{aligned}\mathbb{E}[\log(1+X)] &= \int_1^\infty \log(1+x)\, x^{-2}\, dx \overset{\text{Partielle Integration}}{=} \Big[-\tfrac{\log(1+x)}{x} \Big]_1^\infty + \int_1^\infty \frac{1}{x(1+x)}\, dx. \\ &= 0 - (-\log 2) + \int_1^\infty \frac{1}{x(1+x)}\, dx = \log 2 + \Big[\log x - \log(1+x) \Big]_1^\infty \\ &= \log 2 + \log 2 = 2 \log 2 \ < \infty\end{aligned}$$

Fazit: Die *geänderte* Zielgröße ist integrierbar und per Monte-Carlo gut schätzbar.

[7] wächst sehr langsam, damit werden die schweren Ränder der Paretoverteilung “kompensiert”

Möglicher Fix: Modell/Frage anpassen, bis Integrierbarkeit gilt

Beispiel für Trunkierung

Reparatur B: Trunkierung/Winsorisierung

Definiere z. B.

$$g_T(x) = \min(x, T) \qquad \text{(Winsorisierung)}$$

oder

$$g_T(x) = x\,\mathbf{1}\{x \le T\} \qquad \text{(Trunkierung)}$$

Dann ist

$$\mathbb{E}[g_T(X)] < \infty \quad \forall \quad T$$

und MC funktioniert zuverlässig — die Zielgröße ist bewusst *geändert* (Bias–Varianz-Trade-off[8] via T).

[8] T kleiner, Varianz runter (nicht mehr unendlich), dafür Bias rauf (und umgekehrt)

Bias–Varianz–Trade-off

Trunkierung/Winsorisierung und MSE

Zusammenfassung

Wenn $\int |g| f = \infty$: Zielgröße/Transformation so wählen, dass Integrabilität gilt (z. B. $\log(1+x),\ \arctan x$, Trunkierung).
Effekt einer Trunkierung bei T auf MSE (Trade-off):

$$\mathrm{MSE}(T) = \underbrace{\frac{\mathrm{Var}(g_T(X))}{n}}_{\downarrow \text{ mit } T\downarrow} + \underbrace{\mathrm{Bias}(T)^2}_{\uparrow \text{ mit } T\downarrow}$$

Ein “optimales” T balanciert beide Effekte, und minimiert so den MSE

Bias–Varianz–Trade-off

Trunkierung/Winsorisierung und MSE

Praxis-Heuristik zur Wahl von T

- Wähle T so, dass $\mathrm{Var}(g_T)/N$ (bzw. CI-Breite) stark sinkt, während $\mathrm{Bias}(T)$ klein bleibt.
- Umsetzung:
  - T als hohes Quantil (z. B. 99%) aus Pilotdaten[9]
  - Sensitivitätsanalyse über mehrere Werte von T
  - Bericht: $\hat{\theta}_T$ und das Konfidenzintervall *plus* Bias-Abschätzung (z. B. % der Beobachtungen, die oberhalb von T)
- Minimierung der $\widehat{\mathrm{MSE}}(T) = \widehat{\mathrm{Var}}(g_T)/N + \widehat{\mathrm{Bias}}(T)^2$ über Pilot- oder Cross-Validation.

[9] Teil der tatsächlichen Daten oder Daten aus einer “Machbarkeitsstudie”

Indikatorfunktion

Definition (Indikator)

Für ein Ereignis A ist die *Indikatorfunktion* definiert durch

$$\mathbf{1}_A(x) = \begin{cases} 1, & x \in A, \\ 0, & x \notin A. \end{cases}$$

Indikatorfunktion & Erwartungswert = Wahrscheinlichkeit

Schlüsselidee für Volumen via MC

Zentrale Identität

Sei X eine Zufallsvariable und A ein Ereignis. Dann gilt

$$\mathbb{E}\big[\mathbf{1}_A(X)\big] \;=\; \mathbb{P}(X \in A).$$

Beweis-Skizze:

- $\mathbf{1}_A(X)$ ist Bernoulli verteilt mit Erfolgswahrscheinlichkeit (p), wobei $p = \mathbb{P}(X \in A)$
- Erwartungswert einer Bernoulli verteilten Zufallsvariablen ist p.

$\Rightarrow$ Wählt man bei einer Monte-Carlo Simulation für $g(x)$ die Indikator Funktion[10], schätzt man durch Monte-Carlo die Wahrscheinlichkeit, dass A eintritt.

[10] Die Funktion ist also 1 wenn ein interessierendes Ereignis A eintritt, 0 sonst

Indikatorfunktion & Erwartungswert = Wahrscheinlichkeit

Schlüsselidee für Volumen via MC

Volumen als Wahrscheinlichkeit (Uniform-Verteilung)

Ist $X \sim \mathcal{U}(D)$ *gleichverteilt* auf einer messbaren Domäne $D \subset \mathbb{R}^d$, und $A \subseteq D$, dann

$$\mathbb{P}(X \in A) \;=\; \frac{\mathrm{Vol}(A)}{\mathrm{Vol}(D)} \quad \Rightarrow \quad \mathbb{E}\big[\mathbf{1}_A(X)\big] \;=\; \frac{\mathrm{Vol}(A)}{\mathrm{Vol}(D)}.$$

Umgeformt ergibt sich: $\mathrm{Vol}(A) = \mathrm{Vol}(D) \cdot \mathbb{E}[\mathbf{1}_A(X)]$.

Link zur nächsten Folie

Für $D = [0,1]^d$ und $A = \{x : \|x\|_2 \leq 1\}$ ist

$$\mathbb{E}[\mathbf{1}_A(X)] = \mathbb{P}(X \in A) = \mathrm{Vol}(A) \text{ (da } \mathrm{Vol}(D) = 1).$$

Damit reduziert sich die *Volumenbestimmung* der Einheitskugel auf das *Schätzen eines Erwartungswerts* via Monte Carlo.

Beispiel

Volumenabschätzung per Indikator

- **Fragestellung:** Wie groß ist das Volumen der d-dimensionalen Einheitskugel?
  - Analytisch ist das nur für kleine d einfach, in höheren Dimensionen sehr schwierig.
  - Motivation: Typische Situation in MC, wir wollen ein Integral oder Volumen bestimmen, das keine geschlossene Form hat.
- **Monte-Carlo-Idee:** Ziehe $X \sim \mathcal{U}([0,1]^d)$ und wähle

$$g(x) = \mathbf{1}\{\|x\|_2 \leq 1\},$$

$\|\cdot\|_2$ ist die euklidische Norm.

Beispiel

Volumenabschätzung per Indikator

Die Indikatorfunktion macht das *Volumen* (Flächenanteil) zu einem *Erwartungswert*:

$$\mu = \mathbb{E}[g(X)] = \int_{[0,1]^d} g(x)\,dx = \int_{[0,1]^d} \mathbf{1}\{\|x\|_2 \leq 1\}\,dx = \mathbb{P}(\|X\|_2 \leq 1).$$

- μ ist genau der Anteil des Einheitswürfels, der innerhalb der Kugel liegt.
- Mit dem bekannten Volumen des Würfels $= 1$ entspricht μ dem Kugelvolumen im positiven Quadranten
- Für das Volumen der ganze Kugel multipliziert man mit das Ergebnis mit 2^d.

Monte-Carlo-Schätzer:

$$\hat{\mu}_N = \frac{1}{N} \sum_{i=1}^{N} g(X_i)$$

= Trefferquote der N Stichprobenpunkte.

Beispiel

Volumenabschätzung per Indikator

Warum sinnvoll?

- Verallgemeinerbar auf sehr hohe Dimensionen, wo deterministische Methoden (Quadratur) scheitern.
- Anschauung: “Wir schätzen Volumina durch Zufallspunkte im Raum zählen” , einfaches, aber mächtiges Prinzip.

Hinweis

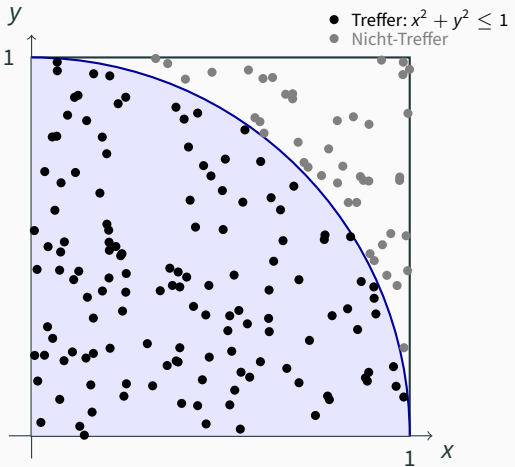
- Indikatorfunktionen erzeugen hohe Varianz (0/1-Ausgabe)
- Mögliche Gegenmaßnahme: *Importance Sampling* oder *Varianzreduktion*[11]

[11] Kapitel 2 und 3

Geometrische Visualisierung (2D): Volumen via Indikator

Recap aus dem letzten Kapitel

- Monte Carlo ersetzt das Integral durch die Trefferquote $\hat{\mu} = \hat{p}$.
- Für die *gesamte* Kreisfläche gilt in 2D: $|\text{Kreis}| = 4 \cdot \mu$; allgemein in d-Dimensionen: $|\text{Kugel}| = 2^d \cdot \mu$.



Replikationen & Stabilität

Erklärung anhand eines Beispiels

Setup

- Zielgröße: $\mu = \int_0^1 \sqrt{x}\, dx = \frac{2}{3}$
- Monte-Carlo: Ziehe $X_i \sim \mathcal{U}(0,1)$, und berechne $\hat{\mu} = \frac{1}{N} \sum \sqrt{X_i}$.

Replikationen (Testet die externe Stabilität)

Führen Sie R *unabhängige* Läufe (Seeds) durch und erhalten $\hat{\mu}^{(1)}, \ldots, \hat{\mu}^{(R)}$.

- **Mittel der Replikate** $\bar{\mu}_R = \frac{1}{R} \sum_r \hat{\mu}^{(r)}$
  $\Rightarrow$ Testet *Bias*: Liegt $\bar{\mu}_R$ nahe $2/3$?).
- **Streuung der Replikate** $\mathrm{sd}(\hat{\mu}^{(r)})$
  $\Rightarrow$ Stanrardabweichung der $\mu^{(r)}$) sollte $\approx \sigma/\sqrt{N}$ sein (im Beispiel ist $\sigma^2 = \mathrm{Var}(\sqrt{X}) = \frac{1}{18}$).
- **Deckung (Coverage)**: Anteil der Replikations-CIs $[\hat{\mu}^{(r)} \pm 1.96\, \widehat{SE}^{(r)}]$
  $\Rightarrow \approx 95\%$ der CIs sollten den wahren Wert $2/3$ enthalten

Replikationen & Stabilität

Erklärung anhand eines Beispiels

Batching (Testet die interne Stabilität)

- Ein *langer* Lauf (N groß) wird in B Blöcke zerlegt.
- Pro Block wird $\bar{Y}_{\text{Block}}$ berechnetn
- SE aus Varianz der Blockmittel[12]

$$\widehat{SE}_{\text{BM}} = \frac{\text{sd}(\bar{Y}_{\text{Block}})}{\sqrt{B}}.$$

Nützlich bei Abhängigkeiten (z. B. zeitlich), bei i.i.d. liefert BM und $s/\sqrt{N}$ ähnliche SE.
$\Rightarrow$ Unterschiede deuten auf Abhängigkeiten hin

Später mehr dazu.

[12] Englisch: Barch-Means SE (BM)

Python: Replikationen & Batching für $\int_0^1 \sqrt{x}\,dx$ i

```python
import numpy as np

TRUTH = 2/3   # E[sqrt(U(0,1))]
SIG2  = 1/18  # Var(sqrt(U(0,1)))
SEED = 42
rng = np.random.default_rng(SEED)

def mc_once(N, rng):
    x  = rng.random(N)
    gx = np.sqrt(x)
    mu = gx.mean()
    se = gx.std(ddof=1) / np.sqrt(N)
    lcl, ucl = mu - 1.96*se, mu + 1.96*se
    return mu, se, lcl, ucl
```

Python: Replikationen & Batching für $\int_0^1 \sqrt{x}\,dx$ ii

```
15
16  def replicate(N=100_000, R=50, seed=42):
17      mus, ses, cov = [], [], 0
18      for r in range(R):
19          sese = np.random.SeedSequence([seed, int(N), r])
20          mu, se, lcl, ucl = mc_once(N, np.random.default_rng(sese))
21          mus.append(mu)
22          ses.append(se)
23          cov += int(lcl <= TRUTH <= ucl) # was passier hier?
24      mus, ses = np.array(mus), np.array(ses)
25
26      summary = {
27          "mean_of_estimates": mus.mean(),
28          "sd_of_estimates": mus.std(ddof=1),
29          "mean_SE_reported": ses.mean(),
```

Python: Replikationen & Batching für $\int_0^1 \sqrt{x}\,dx$ iii

```
30          "theory_sd": np.sqrt(SIG2/N),
31          "coverage_95": cov / R
32      }
33      return summary, mus, ses
34
35  def batch_means(values, B=30):
36      n = len(values)
37      m = n // B # warum hier //?
38      use = m * B
39      y = np.asarray(values[:use]).reshape(B, m).mean(axis=1)
40      se_bm = y.std(ddof=1) / np.sqrt(B)
41      return se_bm, B, m
42
43  # Demo
44  N, R = 100_000, 50
```

Python: Replikationen & Batching für $\int_0^1 \sqrt{x}\,dx$ iv

```
summary, mus, ses = replicate(N=N, R=R, seed=123)
print("Replikations-Summary:", summary)

# Batching auf einem langen Lauf:
rng = np.random.default_rng(123)
x   = rng.random(600_000)
gx = np.sqrt(x)
se_naive = gx.std(ddof=1)/np.sqrt(gx.size)
se_bm, B, m = batch_means(gx, B=60)
print(f"SE_naive={se_naive:.4e} | SE_BM={se_bm:.4e} (B={B}, m={m})")
```

Replikationen & Stabilität

Erklärung anhand eines Beispiels

Was lernen wir?

- $\bar{\mu}_R$ nahe $2/3 \Rightarrow$ kein nennenswerter Bias
- $\mathrm{sd}(\hat{\mu}^{(r)}) \propto 1/\sqrt{N} \Rightarrow$ korrekte Skalierung
- Coverage $\approx 95\% \Rightarrow$ SE/CI stimmig.

Wenn hier Abweichungen beobachtet werden, deuten diese auf auf Implementations-/RNG-/Abhängigkeitsprobleme hin

Zielgenauigkeit & Stichprobengröße

- **Absolute Präzision** (Zielhalbbreite): Finde N, so dass

$$z_{1-\alpha/2} \cdot \frac{s}{\sqrt{N}} \leq \varepsilon$$

- **Relative Präzision:** Finde N, so dass

$$\frac{z_{1-\alpha/2} \cdot \frac{s}{\sqrt{N}}}{|\hat{\mu}|} \leq \delta$$

Beispiel aus der Praxis: ε ist vorgegeben (Präzisionsangaben) und die Frage ist, wie viele Wiederholungen N benötigt werden um diese Präzision zu erreichen:

1. *Pilot* N_0 laufen lassen, s schätzen (und ggf. $\hat{\mu}$)
2. Aus vorgegebenem Ziel-ε (oder δ) die benötigte Stichprobengröße N berechnen
3. Falls $N > N_0$:
   - weitere Samples ziehen, bis N erreicht ist.
   - sonst: geforderte Zielgenauigkeit bereits erreich.

Zielgenauigkeit

Aufgabe

Wir schätzen eine Wahrscheinlichkeit $p = \mathbb{P}(A)$ per Monte Carlo mit $\mathbf{1}\{A\} \in \{0, 1\}$.

- **Pilotlauf:** $N_0 = 100$ Ziehungen, davon 27 Treffer $\Rightarrow \hat{p}_0 = 0{,}27$.
- 95%-Konfidenz ($z_{0{,}975} \approx 1{,}96$).
- In diesem Setting ist $s \approx \sqrt{\hat{p}_0(1 - \hat{p}_0)}$.

Bestimmen Sie für die beiden folgenden Kriterien die nötige Gesamtstichprobe N^* und die zusätzlich benötigten Ziehungen $N^* - N_0$ (falls nötig).

1. **Absolute Präzision:** $\varepsilon = 0{,}02$.
2. **Relative Präzision:** Ziel $\delta = 10\%$ relativ zu $|\hat{\mu}| \approx \hat{p}_0$.

Hinweise. Immer *nach oben* runden. s wird aus dem aktuellsten Pilot geschätzt.

Zielgenauigkeit

Lösung

Sequenzielles Stoppkriterium[13]

- Bei komplexeren Stoppkriterien (z.B. bei mehreren Kriterien, Quantile) ist der Zusammenhang zwischen N und dem Kriterium nicht immer eindeutig.
- Das sequenzielle Stoppkriterium erhöht die Stichprobe Schrittweise (in Batches), solange, bis das Stoppkriterium erfüllt ist.
- **Vorteil:** Rechenzeit $\approx$ Bedarf
- **Nachteil:** Unklar, wie lange eine Simulation läuft (weniger Planbarkeit)
- **Achtung:** Abhängigkeiten $\Rightarrow$ konservativere SE-Schätzung nötig
  - Fehlannahmen (Unabhängigkeit der Stichprobe ist nicht gegeben) ist der SE tatsächlich größer als geschätzt
  - Da die meisten Stoppkriterien auf dem (geringen) SE beruhen würde man in diesem Fall **zu früh** mit der Simulation aufhören

[13] Möglichkeit, die Simulation frühzeitig zu beenden (z.B. Ergebnis ist gut genug)

Sequenzielles Stoppkriterium i

Heuristik, Code

```python
import numpy as np

seed = 42
rng = np.random.default_rng(seed)

def mc_stop_until_se(g, sampler, rng, target_se=1e-3, batch=10_000):
    """Ziehe in Batches bis geschätzter SE <= target_se ist.
    Returns (mu, se, n, history)."""

    values = []
    n = 0
    while True: # warum brauchen wir hier eine while schleife?
        x = sampler(batch, rng)
```

Sequenzielles Stoppkriterium ii

Heuristik, Code

```
14          gx = g(x)
15          values.append(gx)
16          n += batch
17          # was macht np.concatenate?
18          all_vals = np.concatenate(values)
19          mu = all_vals.mean()
20          se = all_vals.std(ddof=1)/np.sqrt(n)
21          if se <= target_se:
22              return mu, se, n, all_vals
23
24
25
26
27
```

Sequenzielles Stoppkriterium iii

Heuristik, Code

```
28  # Beispiel: g(x)=exp(-x**2), X~U(0,1)
29  # Was bedeutet der Unterstrich?
30
31  mu, se, n, _ = mc_stop_until_se(
32      g=lambda x: np.exp(-x**2),
33      sampler=lambda m, rng: rng.random(m),
34      rng=rng
35      target_se=3e-4,
36  )
37
38  print(f"mu={mu:.6f}, se={se:.2e}, N={n}")
```

Effizienz

Zielklarheit vor Optimierung

Ausgangspunkt

- Für den MC-Mittelwert gilt (asymptotisch): $\mathrm{SE}^2 \approx \sigma_{\text{eff}}^2 / N$
- In der Statistik ist Präzision definiert durch $1/\mathrm{SE}^2$
- Zeitmodell für die Dauer einer Simulation in Abhängigkeit von N:

$$T(N) \approx T_0 + N \cdot t_{\text{eval}},$$

  also Start-/Setupkosten T_0 + Kosten pro Sample).

Effizienz

Zielklarheit vor Optimierung

Zielmetrik für beide Größen:

$$\frac{\text{Präzision}}{\text{Zeit}}$$

Zwei gleichwertige Metriken für Effizienz

1. **Maximiere Präzision pro Zeit:**
2. **Minimiere Gesamtaufwand für feste Präzision:**

Effizienz

Zielklarheit vor Optimierung

Intuition

- Reduktion von SE ($\mathrm{SE}^2 \downarrow$):
  - über $\sigma^2_{\text{eff}} \downarrow$ (IS[14]/Varianzreduktion[15])
  - $N \uparrow$
- Reduktion von $T(N)$:
  - $t_{\text{eval}} \downarrow$ (schnellere Auswertung)
  - $T_0 \downarrow$ (schlankes Setup)

Die Effizienz-Metrik koppelt beides fair: Präzision *und* Zeit.

Mini-Rechnung

Halbiere $\mathrm{SE} \Rightarrow$ Viertel der $\mathrm{SE}^2 \Rightarrow$ dafür N vervierfachen (teuer!).
Dagegen: $10\times$ geringere σ^2_{eff} via IS $\Rightarrow$ gleicher Effekt wie $10\times$ größerer N, aber ohne Extrazeit

[14] Noch in diesem Kapitel
[15] Kapitel 3

Schneller werden: Broadcasting & Preallocation i

$t_{\text{eval}} \downarrow$ durch NumPy-Patterns

Broadcasting (Schleifen loswerden)

Idee: Rechenregeln auf ganze Arrays anwenden, statt Python-Schleifen.

$$\mathtt{x} \in \mathbb{R}^{N \times d}, \quad \mathtt{g(x)} \Rightarrow \text{vektorisierte Operationen (C-Loop in NumPy)}.$$

Effekt: Massive Reduktion von Python-Overhead $\Rightarrow t_{\text{eval}} \downarrow$.

Preallocation (einmal anlegen, wiederverwenden)

Idee: Speicher für Arrays *vorher* anlegen, statt in jeder Iteration neu zu alloziieren.
Effekt: weniger Garbage-Collector-/Allocator-Kosten; stabilere Laufzeiten.

Schneller werden: Broadcasting & Preallocation ii

$t_{\text{eval}} \downarrow$ durch NumPy-Patterns

Mini-Code (Schlecht $\rightarrow$ Gut)

```
# Schlecht: Python-Schleife, wiederholte Append/Allokation
vals = []
for _ in range(N):
    u = rng.random()# scalar
    vals.append(np.sqrt(u))# list grows each time
    mu = np.mean(vals)

# Gut: Broadcasting + Preallocation
u = rng.random(N)    # vector draw
gx = np.sqrt(u)  # vectorized transform
mu = gx.mean()    # fast reduction
```

Zeitmodell & Pilotlauf i

Bestimmung von t_{eval} aus N_0

Zeitmodell rekapituliert

$T(N) \approx T_0 + N \cdot t_{\text{eval}}$.

- T_0: Startkosten (Imports, RNG-Setup, Datenladen).
- t_{eval}: *durchschnittliche* Zeit, um ein Sample auszuwerten (Sampling + $g(x)$ + Aggregation)

Pilotlauf N_0: wozu?

- **Schätzt Streuung** $s^2 \Rightarrow$ Ziel-N für gewünschte ε bestimmen.
- **Schätzt Zeiten:** Empirisch $t_{\text{eval}} \approx \frac{T(N_0) - T_0}{N_0}$.
  Messe $T(N_0)$ *und* trockener Start (T_0) separat (z. B. 100 leere Iterationen / Warmup).

Nutzen: Prüfung, ob $N(\varepsilon)$ ins Budget passt ($T_0 + N(\varepsilon)\, t_{\text{eval}} \leq$ Budget).

Daumenregeln:

- Ist t_{eval} hoch: zuerst beschleunigen (Profiling $\Rightarrow$ Vektorisierung/NumPy).
- Dominiert die Varianz: IS/Varianzreduktion testen
  $\Rightarrow 10\times$ Effizienzgewinn schlägt $10\times$ mehr N).
- Nutze *SE-Ziel* statt fixes N; stoppe, wenn $\widehat{SE} \leq \varepsilon$ (oder das Budget aufgebraucht ist)

Profiling Tools

Laufzeit pro Zeile Code - wo ist das Bootleneck?

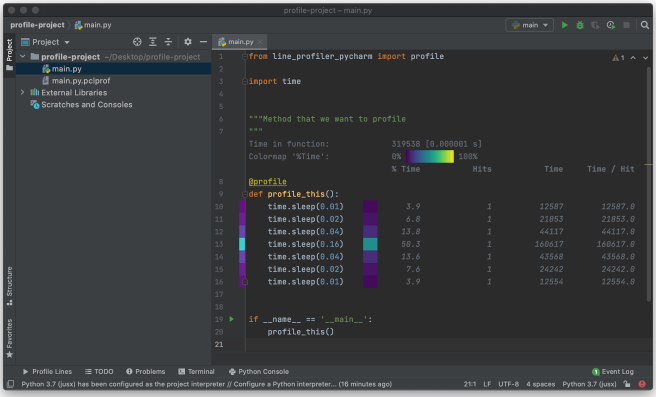


Figure 2: https://plugins.jetbrains.com/plugin/16536-line-profiler

ohm

Importance Sampling

Change of Measure

Grundidee

- Bei seltenen Ereignissen braucht man viele Ziehungen, bis das interessierende Ereignis auftritt.
- Änderung der Ziehung, so dass das interessierende Ereignis häufiger auftritt als normal

$\Rightarrow$ Effizientere Nutzung von N

Change of Measure

Intuition & Formel

Importance sampling: Das Ziel ist immer noch Schätzen von

$$\mu = \mathbb{E}[g(X)] = \int g(x) f(x)\, dx.$$

Umformen ergibt

$$\int g(x) f(x)\, dx = \int g(x) f(x) \frac{q(x)}{q(x)}\, dx = \int \underbrace{g(x) \frac{f(x)}{q(x)}}_{g'(x)} q(x)\, dx$$

Wenn man also aus q anstatt aus f zieht (z.B. weil einfacher zu samplen), erhält man als einen neuen MC-Schätzer:

$$\hat{\mu}^{\mathrm{IS}} = \frac{1}{N} \sum_{i=1}^{N} g(X_i) \underbrace{\frac{f(X_i)}{q(X_i)}}_{w(X_i)}, \quad X_i \sim q.$$

Wie finde ich ein geeignetes q?

Change of Measure

Intuition & Formel

Asymptotische Varianz[16] von $\hat{\mu}^{\mathrm{IS}}$ ist

$$Var(\hat{\mu}^{\mathrm{IS}}) = \frac{1}{N}\left(\int \frac{g(x)^2 f(x)^2}{q(x)} - \mu^2\right).$$

- Der Faktor $\frac{1}{q(x)}$ kann die Varianz in Bereichen von x aufblähen, in denen $q(x)$ sehr klein ist, obwohl $|g(x)|f(x)$ groß ist. In diesem Fall ist das Gewicht riesig.
- Insbesondere in den Rändern der Verteilung[17] sollte q also mindestens so groß sein wie $|g(x)|f(x)$.
- Optimal (theoretisch): $q^*(x) \propto |g(x)|f(x)$ (proportional)

[16] diese soll möglichst klein sein
[17] tail

Wahl der Vorschlagsdichte q

- **Leicht zu sampeln:** Verschiebe/strecke bekannte Dichten z. B. Normal, Exponential, Cauchy (vorsichtig), Mischungen aus Verteilungen
- **Tail-Matching:** q darf *nicht dünner* in den relevanten Tails sein als $f \Rightarrow$ sonst Gewichtsexplosion.
- **ESS (Effective Sample Size):**
  - $\mathrm{ESS} \approx \frac{(\sum_{i=1}^{N} w_i)^2}{\sum_{i=1}^{N} w_i^2}$
  - Jede Wahl von q führt zu einer Menge von w_i
  - Niedrige ESS $\Rightarrow$ Instabilität.

ESS

Mini-Aufgabe

Die folgende Tabelle zeigt zwei Möglichkeiten für die Wahl der Gewichte:

i	1	2	3	4
w_i	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
$\tilde{w}_i$	$\frac{1}{2}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$

- Berechnen Sie die ESS für beide Gewichte
- Vergleichen Sie die Werte und diskutieren Sie das Ergebnis.

IS-Beispiel

Problem: Ränder der Normalverteilung

Ziel: Schätze $\mu = \mathbb{P}(Z > 3.5)$ für $Z \sim \mathcal{N}(0,1)$ (der wahrer Wert ist $\approx 2.32 \times 10^{-4}$, also sehr klein)

$$\mu = \mathbb{E}_\phi\big[\mathbf{1}\{Z > 3.5\}\big], \quad \text{mit } g(z) = \mathbf{1}\{z > 3.5\}$$

Ansatz 1: Naives Monte Carlo[18]

Ziehe $Z_i \sim \mathcal{N}(0,1)$.

$$\hat{\mu}_{\text{naiv}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}\{Z_i > 3.5\} \quad \text{(Anteil der Treffer)}$$

- **Problem:** Man braucht $N \approx 1/\mu \approx 4300$ Samples für *einen einzigen* Treffer!
- Die Varianz $\mathrm{Var}(\hat{\mu}_{\text{naiv}}) = \frac{\mu(1-\mu)}{N} \approx \mu/N$ ist klein, aber der *relative Fehler* $\mathrm{SE}/\mu \approx 1/\sqrt{N\mu}$ ist riesig.

[18] Auch hier wird die Warhrscheinlichkeit wieder mit der Indikatorfunktion als Erwartungswert dargestellt

IS-Beispiel

Problem: Ränder der Normalverteilung

Ansatz 2: Importance Sampling (IS)

Wir brauchen mehr Samples in der Region $\{z > 3.5\}$.

- **Zieldichte:** $f(x) = \phi(x) = \phi(x; 0, 1)$.
- **Vorschlag (Mean-Shift):** $q(x) = \phi(x; \mu_q, 1)$ mit $\mu_q = 3.5$.
- Wir "schieben" die Sampling-Verteilung dorthin, wo das Ereignis stattfindet.

- Allgemein liegen unter $q = \mathcal{N}(\mu_q, 1)$ liegen mehr Stichprobenziehungen *dort, wo der Integrand groß ist*, nämlich in der Region $Z > z_0$; die Gewichte korrigieren den Bias.
- Die Varianz $\mathrm{Var}(\hat{p}_{\mathrm{IS}})$ sinkt stark wenn q die Zielregion gut trifft (kleine Gewichtsstreuung).

IS-Beispiel: Normal-Tail

Herleitung der Gewichte

Das IS-Gewicht ist $w(x) = \frac{f(x)}{q(x)}$, also

$$w(x) = \frac{\phi(x; 0, 1)}{\phi(x; \mu_q, 1)} = \frac{\frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}x^2)}{\frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}(x - \mu_q)^2)}$$

$$w(x) = \exp\left(-\tfrac{1}{2}x^2 + \tfrac{1}{2}(x - \mu_q)^2\right)$$

$$w(x) = \exp\left(-\tfrac{1}{2}x^2 + \tfrac{1}{2}(x^2 - 2x\mu_q + \mu_q^2)\right)$$

$$w(x) = \exp\left(-x\mu_q + \tfrac{1}{2}\mu_q^2\right)$$

IS-Beispiel: Normal-Tail

Herleitung der Gewichte

IS-Schätzer (Mean-Shift)

Ziehe $X_i \sim \mathcal{N}(\mu_q, 1)$ und berechne:

$$\hat{\mu}_{\text{IS}} = \frac{1}{N} \sum_{i=1}^{N} \underbrace{\mathbf{1}\{X_i > 3.5\}}_{g(X_i)} \underbrace{\exp(-\mu_q X_i + \tfrac{1}{2}\mu_q^2)}_{w(X_i)}$$

(Hier mit $\mu_q = 3.5$ und $z_0 = 3.5$)

IS-Beispiel: Python-Implementierung i

```python
import numpy as np
from scipy.stats import norm

Z0 = 3.5
MU_Q = 3.5 # Wir zentrieren q auf dem Schwellenwert
N_NAIVE = 5_000_000
N_IS = 50_000 # 100x weniger Samples!
SEED = 42
rng = np.random.default_rng(SEED)

# 1. Naiver Ansatz (braucht viel N)
z = rng.standard_normal(N_NAIVE)
p_naive = (z > Z0).mean()
se_naive = np.sqrt(p_naive * (1 - p_naive) / N_NAIVE)

```

IS-Beispiel: Python-Implementierung ii

```
16  # 2. Importance Sampling
17  # Ziehe X_i ~ N(mu_q, 1)
18  x = rng.normal(loc=MU_Q, scale=1.0, size=N_IS)
19
20  # Berechne Gewichte w(x) = f(x)/q(x) = N(x;0,1) / N(x;MU_Q,1)
21  w = norm.pdf(x, 0, 1) / norm.odf(x, MU_Q, 1) # likelihood ratio
22
23  # Berechne g(x) * w(x)
24  g = (x > Z0).astype(float)
25  samples_is = g * w
26  p_is = samples_is.mean()
27  se_is = samples_is.std(ddof=1) / np.sqrt(N_IS)
28
29
30
```

IS-Beispiel: Python-Implementierung iii

```
31  # --- Ergebnisse ---
32  print(f"Wahrer Wert: {norm.sf(Z0):.3e}")
33  print(f"Naiv (N={N_NAIVE}): {p_naive:.3e} +/- {1.96*se_naive:.3e}")
34  print(f"IS (N={N_IS}):   {p_is:.3e} +/- {1.96*se_is:.3e}")
```

IS-Beispiel: Ergebnisse & Diagnostik

Lassen Sie den Code laufen und notieren Sie den Output hier

Interpretation

- IS erreicht dieselbe Präzision (gleiche CI-Breite) mit **100-mal weniger** Rechenaufwand ($N = 50k$ vs $N = 5M$).
- Die ESS ist hoch. Das ist ein gutes Zeichen! Die Gewichte sind nicht degeneriert.
- **Übung:** Was passiert, wenn $\mu_q = 10$ gewählt wird? (Spoiler: ESS bricht ein, Varianz explodiert).

IS-Fallstricke & Diagnostik Zusammenfassung

Importance Sampling ist mächtig, aber gefährlich. "Mit großer Macht kommt große Verantwortung."

- **Gewichts-Degeneration:** Das Hauptproblem. Wenige Samples X_i erhalten fast das gesamte Gewicht ($\tilde{w}_i \gg 0$).
  - **Folge:** Der Schätzer $\hat{\mu}$ wird durch diese 1-2 Samples dominiert. Die geschätzte Varianz ist riesig (oder trügerisch klein, wenn man Pech hatte und kein großes Gewicht gezogen hat).
- **Falsche Tails:** Wenn $q(x)$ "dünnere" Ränder hat als $f(x) \cdot |g(x)|$, d.h. $q(x) \to 0$ *schneller* als der Zähler.
  - **Folge:** Die Varianz des IS-Schätzers ist *unendlich*, auch wenn der Erwartungswert existiert. Der Schätzer konvergiert nie.

Immer prüfen!

1. **Gewichts-Plot:** Histogramm/Boxplot der $\log(\tilde{w}_i)$. Gibt es extreme Ausreißer?
2. **Stabilität:** Führe die Simulation 3-5 Mal mit neuen Seeds durch. Schwankt $\hat{\mu}_{\text{IS}}$ dramatisch?

Praktische Fallstricke

Woher kommt Abhängigkeit?

Bisher: Annahme $X_i \overset{\text{i.i.d.}}{\sim} f$. Was, wenn die $g(X_i)$ nicht unabhängig sind?

- **Modellseitig (subtil):** Simulation von stochastischen Prozessen.
  - **Warteschlange:** Die Wartezeit W_i von Kunde i hängt stark von der Wartezeit W_{i-1} ab.
  - **Finanzzeitreihe:** Aktienkurs P_t (heute) hängt vom Aktienkurs P_{t-1} (gestern) ab.
- **Technisch (vermeidbar):** Falsches Seeding bei Parallelisierung, Wiederverwendung desselben RNG-Streams für abhängige Prozesse.

Konsequenz: Die $Y_i = g(X_i)$ sind autokorreliert

Wenn Y_i und Y_{i+k} korreliert sind, also

$$\rho_k = \mathrm{Corr}(Y_i, Y_{i+k}) \neq 0,$$

ist die i.i.d.-Annahme verletzt.

Folgen von (positiver) Autokorrelation

Der Zentrale Grenzwertsatz (CLT) gilt oft weiterhin, aber die Varianz ändert sich!

- Für einen stationären Prozess Y_i mit $\mathrm{Var}(Y_i) = \sigma^2$ und Autokorrelation ρ_k gilt:

$$\mathrm{Var}(\bar{Y}_N) = \frac{\sigma^2}{N}\left(1 + 2\sum_{k=1}^{N-1}(1 - \tfrac{k}{N})\rho_k\right) \approx \frac{\sigma^2_{\text{eff}}}{N}$$

- **Effektive Varianz:** $\sigma^2_{\text{eff}} = \sigma^2(1 + 2\sum_{k\geq 1}\rho_k)$.
- Die Summe $1 + 2\sum \rho_k$ wird oft **Autocorrelation Time** (τ) genannt.

Die Gefahr: Falsche Sicherheit

- Bei positiver Korrelation ($\rho_k > 0$, häufigster Fall) ist $\sigma^2_{\text{eff}} > \sigma^2$.
- Der naive SE-Schätzer $s/\sqrt{N}$ **unterschätzt** den wahren Standardfehler massiv!
- **Folge:** Konfidenzintervalle sind viel zu eng, $\hat{\mu} \pm 1.96\, s/\sqrt{N}$ hat eine viel geringere Deckungswahrscheinlichkeit als 95% (z.B. nur 60%). Man ist "overconfident".

Diagnose: Autokorrelations-Plot (ACF)

Wie finden wir heraus, ob wir ein Problem haben?

- **ACF-Plot:** Wir plotten die geschätzte Korrelation $\hat{\rho}_k$ gegen den "Lag" k.
- **Tool:** `statsmodels.graphics.tsaplots.plot_acf`
- **Interpretation:**
  - **i.i.d. (gut):** Nur $\hat{\rho}_0 = 1$. Alle anderen $\hat{\rho}_k$ ($k \geq 1$) sind nahe Null (innerhalb des blauen Konfidenzbandes).
  - **Autokorreliert (Problem):** $\hat{\rho}_k$ fällt nur langsam auf Null ab.

Diagnose: Autokorrelations-Plot (ACF)

Effective Sample Size (ESS) - Alternative Sicht

Eine andere Art, den Varianz-Effekt zu messen:

$$N_{\text{eff}} = \frac{N}{1 + 2\sum_{k \geq 1} \rho_k} = \frac{N}{\tau}$$

- N_{eff} ist die Anzahl *unabhängiger* Samples, die denselben Standardfehler wie unsere N *abhängigen* Samples hätten.
- Beispiel: $N = 1,000,000$, aber $\tau = 100$. Dann ist $N_{\text{eff}} = 10,000$. Unsere 1 Million Samples sind nur so gut wie 10.000 unabhängige.

Abhilfe: Batch-Means (BM)

Robuste SE-Schätzung

Wenn wir wissen, dass die Autokorrelation nach einem Lag m abklingt ($\rho_k \approx 0$ for $k \geq m$), können wir die Daten "verklumpen".

- **Idee:** Teile die lange Sequenz $Y_1, \ldots, Y_N$ in B Blöcke (Batches) der Länge m (so dass $N = B \cdot m$).
- Berechne pro Block den Mittelwert (das "Batch-Mean"):

$$\bar{Y}_b = \frac{1}{m} \sum_{i=(b-1)m+1}^{bm} Y_i, \quad \text{für } b = 1, \ldots, B$$

- **Annahme:** Wenn m groß genug ist, sind die Blockmittel $\bar{Y}_1, \ldots, \bar{Y}_B$ (nahezu) unkorreliert.
- Jetzt können wir den Standard-SE-Schätzer auf diese B Blockmittel anwenden!

Abhilfe: Batch-Means (BM)

Robuste SE-Schätzung

Batch-Means Standardfehler (SE-BM)

$$s_{\mathrm{BM}}^2 = \frac{1}{B-1}\sum_{b=1}^{B}(\bar{Y}_b - \bar{Y}_N)^2 \quad \text{(Varianz der Blockmittel)}$$

$$\widehat{\mathrm{SE}}_{\mathrm{BM}}(\bar{Y}_N) = \frac{s_{\mathrm{BM}}}{\sqrt{B}} = \sqrt{\frac{\mathrm{Var}(\bar{Y}_b)}{B}}$$

Wahl von B, m: Wähle B nicht zu klein (z.B. $B \geq 30$), damit die Varianzschätzung s_{BM}^2 stabil ist. m ergibt sich dann als N/B.

Batch-Means, einfache Implementierung i

```python
import numpy as np

def batch_means_se(values, B = 30):
    """SE via Batch-Means für 1D-Array; schneidet Ende ab,
    damit B Blöcke exakt passen."""

    n = values.size
    m = n // B # m = Länge jedes Batches (Integer-Division)

    if m == 0:
        print("Warnung: Zu wenig Daten für B Batches.")
        return np.nan, B, m

    n_use = m * B # Gesamtzahl der Samples, die genutzt werden

```

Batch-Means, einfache Implementierung ii

```
16      # Schneide 'values' zu, damit es durch B teilbar ist
17      y_trimmed = values[:n_use]
18
19      # Reshape in B Zeilen und m Spalten
20      y_batched = y_trimmed.reshape(B, m)
21
22      # Berechne Mittelwert jeder Zeile (axis=1) -> B Blockmittel
23      y_means = y_batched.mean(axis=1)
24
25      # Wende Standard-SE-Formel auf die B Blockmittel an
26      # ddof=1 für N-1 im Nenner der Varianz
27      se = y_means.std(ddof=1) / np.sqrt(B)
28
29      return se, B, m
```

Fallstrick 2: Instabile Varianz (Schwere Ränder)

Dies ist das Problem aus Folie 26, aber es ist so wichtig, dass wir es wiederholen.

- **Problem:** Wir wollen $\mu = \mathbb{E}[g(X)]$ schätzen, aber $\mathrm{Var}[g(X)] = \infty$.
- (Erinnerung: Pareto mit $\alpha = 1.5$, Cauchy-Verteilung).
- **Symptome:**
  - Der laufende Mittelwert $\hat{\mu}_N$ *konvergiert nicht* stabil. Er "springt" jedes Mal, wenn ein extremer Ausreißer $g(X_i)$ gezogen wird.
  - Der naive SE-Schätzer $s/\sqrt{N}$ konvergiert ebenfalls nicht. Der Zähler s wächst mit N.
  - Der $1/\sqrt{N}$-Plot der Konvergenz (aus der Mini-Demo) wird keine Gerade mit Steigung -0.5 sein.

Fallstrick 2: Instabile Varianz (Schwere Ränder)

Konsequenz: CLT gilt nicht!

Die Annahme $\sigma^2 < \infty$ des Zentralen Grenzwertsatzes ist verletzt.

- Die Verteilung von $\sqrt{N}(\bar{Y}_N - \mu)$ konvergiert nicht gegen eine Normalverteilung, sondern (ggf.) gegen eine "stabile Verteilung" (oft mit unendlicher Varianz).
- Standard-Konfidenzintervalle $\hat{\mu} \pm 1.96\,\widehat{\mathrm{SE}}$ sind **völlig sinnlos**.

Diagnose & Abhilfe

- **Diagnose:** Plot der $g(X_i)$ vs. i (suchen nach Spikes), Histogramm der $g(X_i)$ (suchen nach extremen Tails).
- **Abhilfe:** Problem ändern! (Trunkierung, Winsorisierung), Transformation (Log), oder IS mit einer *noch schwereren* Tail-Verteilung q.

Checkliste für transparentes Reporting

Ein MC-Ergebnis ohne Unsicherheitsmaß ist wertlos. Ein Ergebnis ohne Setup-Details ist nicht reproduzierbar. **Minimalanforderungen für jedes Ergebnis:**

- **Was (Die Schätzung):**
  - Punktschätzer $\hat{\mu}$ (z.B. Mittelwert, Quantil).
  - Unsicherheitsmaß: Standardfehler $\widehat{\mathrm{SE}}$ *und* ein $(1-\alpha)$%-Konfidenzintervall (z.B. 95%-CI).
- **Wie (Das Setup):**
  - Stichprobengröße *N*.
  - Bei Abhängigkeit: Anzahl Batches *B* und Länge *m* (für SE-BM).
  - Bei IS: Verwendete Dichte $q(x)$ und die resultierende ESS.
- **Womit (Die Reproduzierbarkeit):**
  - Verwendeter RNG (z.B. NumPy PCG64).
  - Verwendeter (Haupt-)Seed (z.B. Seed=42).
  - Kurzer Vermerk zur Seeding-Strategie (z.B. "Replikationen via `SeedSequence`").

Visuelle Kommunikation der Unsicherheit

- **Histogramm / Dichte-Plot:** Zeigen Sie die Verteilung der *Ergebnisgröße* $g(X_i)$, nicht nur deren Mittelwert. Das vermittelt die zugrundeliegende Variabilität.
- **Fehlerbalken (Error Bars):** Wenn Sie mehrere Szenarien vergleichen (z.B. $\hat{\mu}_A$ vs. $\hat{\mu}_B$), plotten Sie *immer* $\hat{\mu} \pm 1.96\,\widehat{\mathrm{SE}}$. Nur so ist sichtbar, ob sich die Ergebnisse signifikant unterscheiden.
- **Konvergenz-Plot:** (Optional, für Diagnose) Plot des laufenden Mittels $\hat{\mu}_k$ (mit CIs) gegen $k = 1 \ldots N$. Stabilisiert sich der Schätzer?
- **IS-Diagnostik:** (Essentiell bei IS) Histogramm der $\log(w_i)$. Zeigen Sie, dass die Gewichte "gesund" sind.
- **Batch-Means:** Konfidenzintervall der Blockmittel

Mini-Template für ein Ergebnis (Berichtstext)

Ziel. Schätzung des 99%-Quantils ($Q_{.99}$) der Projektkosten.
Methode. Standard-MC mit $N = 500{,}000$ Replikationen. Jede Replikation simuliert einen vollständigen Projektpfad.
RNG & Seed. NumPy `default_rng(PCG64)`, Haupt-Seed=20250925. Unabhängige Streams pro Replikation via `SeedSequence.spawn()`.
Schätzung. Der Punktschätzer für das 99%-Quantil ist $\hat{Q}_{.99} = 1.42$ Mio. EUR.
Unsicherheit. Ein 95%-Konfidenzintervall für $Q_{.99}$, geschätzt via Bootstrap (1000 Resamples), ist [1.39 Mio. EUR, 1.45 Mio. EUR].
Diagnostik. Die Verteilung der Projektkosten ($g(X_i)$) zeigt Rechtsschiefe, aber keine extremen Ausreißer (Varianz scheint endlich). Der SE der Schätzung ist stabil.
Fazit. Wir sind zu 95% zuversichtlich, dass das 99%-Quantil der Kosten zwischen 1.39 und 1.45 Mio. EUR liegt.

Mini-Demo

Mini-Demo

Konvergenzrate $\propto 1/\sqrt{N}$ visualisieren

- **Ziel:** Empirisch prüfen, ob der Fehler unserer Schätzung wie vom CLT vorhergesagt skaliert.
- **Theorie:** $\mathrm{SE}(\hat{\mu}_N) \propto \frac{1}{\sqrt{N}}$. Der absolute Fehler $|\hat{\mu}_N - \mu|$ sollte sich ähnlich verhalten.
- **Log-Log-Transformation:**

$$\log(\text{Fehler}) \approx \log(C \cdot N^{-1/2}) = \log(C) - \frac{1}{2}\log(N)$$

- **Hypothese:** Ein Plot von $\log(\text{Fehler})$ (y-Achse) gegen $\log(N)$ (x-Achse) sollte eine Gerade mit der **Steigung -0.5** ergeben.

Code-Skizze: Gitter über $N \times R$ i

Wir simulieren $\mu = \mathbb{E}[\exp(-X^2)]$ mit $X \sim \mathcal{U}(0,1)$. (Wahrer Wert ≈ 0.7468).

```
import numpy as np
import pandas as pd

TRUTH = 0.746824 # scipy.special.erf(1) * np.sqrt(np.pi) / 2
SEED = 42
base_seed_seq = np.random.SeedSequence(SEED)

def run_grid(
    ns=(10**2, 10**3, 10**4, 10**5), R=50,
    base_seed_seq=base_seed_seq):
    rows = []

    child_seeds = base_ss.spawn(len(ns) * R)
    k = 0
```

Code-Skizze: Gitter über $N \times R$ ii

```
15      for N in ns:
16          for r in range(R):
17              rng = np.random.default_rng(child_seeds[k])
18              k += 1
19              x = rng.random(N)
20              gx = np.exp(-x**2)
21              mu = gx.mean()
22              se = gx.std(ddof=1)/np.sqrt(N)
23              error = np.abs(mu - TRUTH) # in Wirklichkeit nie berechenbar
24              rows.append(
25                  {"N": N, "rep": r, "mu": mu, "se": se, "error": error}
26              )
27      return pd.DataFrame(rows)
28
29
```

Code-Skizze: Gitter über $N \times R$ iii

```
# Analyse (im Notebook)
# df = run_grid()
# agg = df.groupby('N')['error'].mean().reset_index()
# agg['log_N'] = np.log10(agg['N'])
# agg['log_error'] = np.log10(agg['error'])
# ... (dann plotte log_error vs log_N und fitte eine Linie)
```

Mini-Demo: Erwartetes Ergebnis

Ein Plot der aggregierten Ergebnisse (`agg` aus dem Code) würde Folgendes zeigen:

- **Linearer Plot:** Der Fehler fällt, aber die *Rate* ist schwer zu erkennen.
- **Log-Log Plot:** Die Punkte liegen auf einer klaren Geraden. Die gefittete Linie hat eine Steigung sehr nahe an -0.5. $\rightarrow$ **CLT empirisch bestätigt!**

Takeaway

- Solche Diagnose-Plots sind extrem nützlich, um Vertrauen in die eigene Implementierung zu gewinnen.
- Weicht die Steigung stark von -0.5 ab, hat man evtl. ein Problem:
  - **Steigung > -0.5 (flacher):** Evtl. unendliche Varianz (schwere Ränder).
  - **Steigung < -0.5 (steiler):** Glück gehabt! Man nutzt evtl. unbewusst eine Varianzreduktionstechnik (z.B. Quasi-MC).

Ausblick

Zusammenfassung & Takeaways

- **Kern (LLN/CLT):** MC-Integration ist universell, unverzerrt und konsistent. Der Fehler skaliert *zuverlässig* mit $\propto 1/\sqrt{N}$, *vorausgesetzt* $\sigma^2 < \infty$.
- **Unsicherheit ist Pflicht:** Ein MC-Schätzer $\hat{\mu}$ *muss* immer von $\widehat{\mathrm{SE}}$ oder einem CI begleitet werden.
- **Effizienz (IS):** Importance Sampling kann die Varianz (σ^2) drastisch senken (Faktor 100+), birgt aber die Gefahr der Gewichts-Degeneration. $\rightarrow$ **Immer ESS prüfen!**
- **Fallstricke (Dependenz):** Autokorrelation in den Daten führt zu *unterschätzten* Standardfehlern. $\rightarrow$ **Immer Batch-Means (BM) nutzen**, wenn i.i.d. nicht garantiert ist.
- **Reporting:** Reproduzierbarkeit (Seed, N) und Transparenz (Methode, $\widehat{\mathrm{SE}}$) sind entscheidend für Vertrauen.

Mini-Studie (Ausblick): Coupon Collector's Problem

- **Szenario:** Ein Sammelalbum hat $n = 50$ verschiedene Sticker. In jeder Packung ("Booster") ist genau 1 Sticker, rein zufällig (mit Zurücklegen) aus den n Typen gezogen.
- **Fragestellung:** Wie viele Packungen T muss man *im Mittel* kaufen, um das Album zu vervollständigen? ($\mathbb{E}[T]$?) Wie ist die Verteilung von T?
- **Analytische Lösung (bekannt):** $\mathbb{E}[T] = n \sum_{k=1}^{n} \frac{1}{k} \approx n \log(n)$.
- **Simulations-Ansatz:**
  - $g(X)$ ist hier kein einfaches $f(x)$, sondern der Output eines *ganzen Prozesses*.
  - $X^{(i)}$ ist eine *Sequenz* von Zufallszahlen.
  - $g(X^{(i)}) = T_i$ = Die (zufällige) Zeit, bis in Sequenz i alle 50 Typen gesehen wurden.
- **Ziel:** Schätze $\hat{\mu}_N = \frac{1}{N} \sum_{i=1}^{N} T_i$ (für N Replikationen) und $\widehat{\mathrm{SE}}$.

Ausblick: Coupon Collector i

Simulationslogik (1 Replikation)

Wie simuliert man T_i genau einmal?

```python
import numpy as np

def simulate_one_coupon_run(n=50, rng):
    """
    Simuliert das Sammeln von 'n' Coupons und gibt die
    benötigte Anzahl an Ziehungen (T) zurück.
    """
    collected = set()
    n_collected = 0
    t = 0

    while n_collected < n:
        # Ziehe einen neuen Coupon (Zahl von 0 bis n-1)
```

Ausblick: Coupon Collector ii

Simulationslogik (1 Replikation)

```
14            new_coupon = rng.integers(low=0, high=n, size=1)[0]
15            t += 1
16
17            # if new_coupon not in collected:
18            collected.add(new_coupon)
19            n_collected = len(collected)
20
21        return t
```

Ausblick: Coupon Collector i

Gesamte MC-Simulation

```
N_REPS = 10_000
seed = 42
rng = np.random.default_rng(seed)
results_T = []
for _ in range(N_REPS):
     T_i = simulate_one_coupon_run(n=50, rng=rng)
     results_T.append(T_i)

mu_hat = np.mean(results_T)
se_hat = np.std(results_T, ddof=1) / np.sqrt(N_REPS)
```

Nächste Schritte: Varianzreduktion durch intelligente nicht-i.i.d-Ziehung.