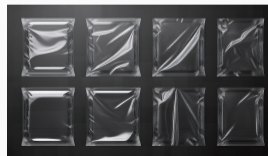


Varianzreduktion & Experimentdesign

Effizienz-Hacks, saubere Vergleiche & robustes Design

Wintersemester 2025/26,
Benedikt Mangold



Unser Plan

1. Warum Varianzreduktion?
2. Antithetische Variablen
3. Control Variates
4. Stratified Sampling & Latin Hypercube Sampling (LHS)
5. Common Random Numbers (CRN)
6. Faktorielles Design & Replikationspläne
7. Robustheit & Reproduzierbarkeit
8. Ausblick

- **Art Owen: *Monte Carlo theory, methods and examples* – Kapitel “Variance reduction”**
(frei verfügbar). <https://artowen.su.domains/mc/Ch-var-basic.pdf> bzw. Buchübersicht
- **SciPy QMC/LHS** – `scipy.stats.qmc` (Quasi-Monte-Carlo, LatinHypercube).
<https://docs.scipy.org/doc/scipy/reference/stats.qmc.html>
<https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.qmc.LatinHypercube.html>
- **Control Variates – Tutorial (PyApprox)** – zweistufige Modelle & Effizienz.
https://sandialabs.github.io/pyapprox/auto_tutorials/multi_fidelity/plot_control_variate_monte_carlo.html
- **Common Random Numbers (CRN) – Tutorialartikel (PMC).**
<https://pmc.ncbi.nlm.nih.gov/articles/PMC2761656/>

Warum Varianzreduktion?

Roter Faden & Lernziele

Frage des Kapitels. Wie erreichen wir die gleiche Monte-Carlo-Präzision deutlich schneller – oder mehr Präzision im gleichen Rechenbudget – ohne bloß die Stichprobengröße zu erhöhen?

Nach diesem Kapitel können Sie:

- erklären, warum die reine Erhöhung von N ineffizient ist und weshalb Varianzreduktion der zentrale Hebel ist;
- die Wirkprinzipien von Antithetik, Control Variates, Stratified Sampling und Latin Hypercube Sampling (LHS) verständlich wiedergeben;
- Common Random Numbers (CRN) korrekt für Szenarienvergleiche einsetzen und die Varianz der Differenzschätzung herleiten
- einfache faktoriellen Designs planen, den Replikationsbedarf begründen und die Effizienz Ihrer Simulationen transparent reporten
- robuste, reproduzierbare Experimente aufsetzen (Seeds, Streams, Diagnostik) und typische Fallstricke vermeiden.

Motivation: Der $1/\sqrt{N}$ -Flaschenhals

- MC-Standardfehler (aus Kap. 2): $SE(\hat{\mu}_N) = \frac{\sigma}{\sqrt{N}}$
- “ $\frac{1}{\sqrt{N}}$ -Flaschenhals”: Präzision steigt nur wie $1/\sqrt{N}$
 - **Halbierung** des Standardfehlers \Rightarrow **Vervierfachung** von N
 - **Zehntelung** des Standardfehlers \Rightarrow **Verhundertfachung** von N
- Oft rechentechnisch zu teuer oder unmöglich!

Alternative: Den Zähler angreifen

Statt N (Nenner) brutal zu erhöhen, *senken* wir $\sigma = \sqrt{\text{Var}[g(X)]}$ (Zähler).

- Ziel: Finde einen *anderen* Schätzer $\hat{\mu}_{\text{neu}}$ für μ , der
 1. immer noch unverzerrt ist

$$\mathbb{E}[\hat{\mu}_{\text{neu}}] = \mu$$

2. aber eine *kleinere Varianz* hat

$$\text{Var}(\hat{\mu}_{\text{neu}}) < \text{Var}(\hat{\mu}_{\text{naiv}}).$$

- Schneller Weg, um gegebenes Fehlerziel innerhalb Zeit-/ Budgetrahmens zu erreichen
 \Rightarrow je nach Budget oft auch der einzige Weg.

Effizienzmetrik: Varianz \times Zeit

- Ein faires Maß für die **Gesamteffizienz** eines MC-Schätzers ist der Aufwand, der für eine bestimmte Präzision (z.B. SE^2) nötig ist.
- Ziel: Minimiere das Produkt aus Varianz und Rechenzeit.

$$\text{Effizienz} = \text{Var}(\hat{\mu}) \cdot T_{\text{total}}$$

- Sei t die (durchschnittliche) Rechenzeit pro Sample $g(X_i)$ und $\sigma^2 = \text{Var}(g(X_i))$.
- Für ein naives i.i.d.-MC-Verfahren mit N Samples:

$$\text{Effizienz}_{\text{naiv}} = \text{Var}(\hat{\mu}_N) \cdot T_{\text{total}} = \left(\frac{\sigma^2}{N} \right) \cdot (N \cdot t) = \sigma^2 \cdot t$$

- **Kernidee:** Ein neues Verfahren (z.B. "CV" für Control Variate) ist besser, wenn

$$\sigma_{\text{CV}}^2 \cdot t_{\text{CV}} < \sigma_{\text{naiv}}^2 \cdot t_{\text{naiv}}$$

- Eine leichte Erhöhung der Zeit pro Ziehung ($t_{\text{CV}} > t_{\text{naiv}}$) ist akzeptabel, wenn der Varianzgewinn ($\sigma_{\text{CV}}^2 \ll \sigma_{\text{naiv}}^2$) dies überkompensiert.

Beispiel

Effizienzgewinn greifbar machen

Szenario 1: Naives MC

- $\sigma_{\text{naiv}}^2 = 1.0$
- $t_{\text{naiv}} = 10 \text{ ms / Sample}$
- Effizienz = $1.0 \times 10 = \mathbf{10.0}$
- Ziel: SE = 0.01
- Nötiges N :
$$\frac{\sigma^2}{N} \leq \text{SE}^2 \Rightarrow N \geq \frac{1.0}{0.01^2} = 10\,000$$
- **Gesamtzeit:** $10\,000 \times 10 \text{ ms} = 100 \text{ s}$

Szenario 2: Methode X

- $\sigma_X^2 = 0.2$ (80% Varianzreduktion)
- $t_X = 12 \text{ ms / Sample}$ (20% teurer)
- Effizienz = $0.2 \times 12 = \mathbf{2.4}$
- Ziel: SE = 0.01
- Nötiges N :
$$N \geq \frac{0.2}{0.01^2} = 2\,000$$
- **Gesamtzeit:** $2\,000 \times 12 \text{ ms} = 24 \text{ s}$

Ergebnis

Methode X ist $\approx 4.2 \times$ schneller ($100/24$) für das *gleiche* Präzisionsziel, obwohl jede einzelne Ziehung *teurer* ist.

Wahlhilfe: Welche Technik passt zu meinem Problem?

Heuristische Entscheidungshilfe

1. Für *Indikator-Funktionen* / *seltene Ereignisse*: Nutzen Sie in erster Linie **Importance Sampling** (vgl. Kap. 2; Tail-Matching/Mean-Shift).
2. Wenn Ihre Zielgröße $g(X)$ *glatt und (überwiegend) monoton* von den Inputs X abhängt, wirkt **Antithetik** oft sofort und ist fast kostenlos.
3. Wenn Sie eine *Hilfsgröße* $h(X)$ mit bekanntem Erwartungswert μ_h finden, die stark mit $g(X)$ korreliert, sind **Control Variates** extrem effektiv.
4. Wenn sich die Eingangsdomäne natürlich in *homogene Regionen (Strata)* teilen lässt (oder viele Eingänge vorhanden sind), helfen **Stratified Sampling** bzw. **LHS** beim "Raumausfüllen".
5. Wenn Sie *Differenzen zwischen Szenarien* ($\mu_A - \mu_B$) schätzen, steigern **Common Random Numbers (CRN)** die Präzision, indem sie gemeinsame Zufallseinflüsse koppeln.

Diese Techniken schließen sich nicht aus; in der Praxis werden sie gezielt kombiniert (z.B. Stratified + Antithetic), um den größten Effizienzgewinn zu erzielen.

Verschiedene Methoden zur Varianzreduktion:

- ☐ Antithetische Variablen
- ☐ Control Variates
- ☐ Stratified Sampling & Latin Hypercube
- ☐ Common Random Numbers
- ☐ Faktorielles Design & Reproduktionspläne
- ☐ Robustheit & Reproduzierbarkeit

Antithetische Variablen

Prinzip

- **Idee:** Statt N unabhängige Ziehungen U_i zu verwenden, nutzen wir nur $N/2$ Ziehungen U_i und bilden *Paare* $(U_i, 1 - U_i)$.
- Beide, U_i und $1 - U_i$, sind perfekt $\mathcal{U}(0, 1)$ -verteilt.
- Wir bilden $N/2$ gepaarte Schätzer:

$$y_i^{\text{anti}} = \frac{1}{2} (g(F^{-1}(U_i)) + g(F^{-1}(1 - U_i))) = \frac{1}{2} (g(X_i) + g(X_{\bar{i}}))$$

- Der finale Schätzer ist der Mittelwert dieser $N/2$ Paare:

$$\hat{\mu}_{\text{anti}} = \frac{1}{N/2} \sum_{i=1}^{N/2} y_i^{\text{anti}} = \frac{1}{N} \sum_{i=1}^{N/2} (g(X_i) + g(X_{\bar{i}}))$$

Intuition

- Wenn g **monoton** ist (z.B. steigend¹), dann
 - U_i klein $\Rightarrow X_i$ klein $\Rightarrow g(X_i)$ klein
 - $1 - U_i$ groß $\Rightarrow X_{\bar{i}}$ groß $\Rightarrow g(X_{\bar{i}})$ groß
- Ein "zu kleiner" Wert wird durch einen "zu großen" Wert im Paar ausbalanciert. Der Mittelwert des Paares (Y_i^{anti}) ist sehr stabil (hat eine geringe Varianz).

¹Z.B. die Quantilsfunktion

Antithetik: Visuelle Intuition

Fall 1: Monotone Funktion (Gut)

- $g(x) = e^x$
- U_i (klein) erzeugt $g(X_i)$ (klein)
- $1 - U_i$ (groß) erzeugt $g(X_{\bar{i}})$ (groß)
- Der Mittelwert $\frac{1}{2}(g(X_i) + g(X_{\bar{i}}))$ liegt immer nahe am wahren μ .
- \Rightarrow **Stark negative Korrelation**, hohe Varianzreduktion.

Fall 2: Nicht-Monoton (Schlecht)

- $g(x) = \cos(5\pi x)$
- U_i (klein) erzeugt $g(X_i)$ (z.B. groß)
- $1 - U_i$ (groß) erzeugt $g(X_{\bar{i}})$ (z.B. auch groß)
- Der Mittelwert schwankt stark.
- \Rightarrow **Nahe-Null-Korrelation**, kein Gewinn.

Antithetik: Formale Wirkung

Varianz des Paares $Y^{\text{anti}} = \frac{1}{2}(g_1 + g_2)$, mit $g_1 = g(X_i)$ und $g_2 = g(X_a)$

- Aus Statistik bekannt: $\text{Var}(A + B) = \text{Var}(A) + \text{Var}(B) + 2\text{Cov}(A, B)$. Damit gilt

$$\text{Var}(Y^{\text{anti}}) = \text{Var}\left(\frac{1}{2}(g_1 + g_2)\right) = \frac{1}{4}(\text{Var}(g_1) + \text{Var}(g_2) + 2\text{Cov}(g_1, g_2))$$

- Da X_i und X_a (also g_1 und g_2) identisch verteilt sind, gilt $\text{Var}(g_1) = \text{Var}(g_2) = \sigma^2$.

$$\text{Var}(Y^{\text{anti}}) = \frac{1}{4}(2\sigma^2 + 2\text{Cov}(g_1, g_2)) = \frac{1}{2}(\sigma^2 + \text{Cov}(g_1, g_2))$$

- **Vergleich:** Ein naives MC-Paar $Y^{\text{naiv}} = \frac{1}{2}(g_1 + g_3)$ mit g_3 unabhängig hätte $\text{Cov} = 0$ und damit $\text{Var}(Y^{\text{naiv}}) = \frac{1}{2}\sigma^2$.

Antithetik: Formale Wirkung & Grenzen

Fazit

- Antithetik *gewinnt* (d.h. $\text{Var}(Y^{\text{anti}}) < \text{Var}(Y^{\text{naiv}})$), genau dann wenn

$$\text{Cov}(g(F^{-1}(U)), g(F^{-1}(1 - U))) < 0$$

- Dies ist für alle monotonen Funktionen $g \circ F^{-1}$ garantiert.

Antithetik: Implementationsmuster

- Ziehe $U_1, \dots, U_{N/2}$ und bilde Paare $(U_i, 1 - U_i)$
- Bei Bedarf: Transformation mit inverser Verteilungsfunktion² $X = F^{-1}(U)$
⇒ Auf diese Weise lassen sich Antithetik-Muster auf Normal-, Exponential- und viele weitere Verteilungen übertragen
- Effizient in Vektor-APIs (NumPy/SciPy) durch Aneinanderhängen der Arrays und vektorweise Berechnung des mittleren Paardurchschnittes bildet.

²Variablentransformation, vgl. Kapitel 2

Antithetik: Gewinn vs. Naiv i

- Schätze $\mu = \mathbb{E}[e^Z]$ wobei $Z \sim \mathcal{N}(0, 1)$. (Wahrer Wert: $e^{0.5} \approx 1.6487$).
- Die Funktion $g(Z) = e^Z$ ist monoton steigend.

```
1  import numpy as np
2  from scipy.stats import norm
3
4  TRUTH = np.exp(0.5)
5  SEED = 42
6  rng = np.random.default_rng(SEED)
7
8  def g_of_z(z):
9      # Monotone Funktion
10     return np.exp(z)
11
12
```

Antithetik: Gewinn vs. Naiv ii

```
13 def mc_naive(N=200_000, rng=rng):
14     z = rng.standard_normal(N)
15     gz = g_of_z(z)
16     mu = gz.mean()
17     se = gz.std(ddof=1) / np.sqrt(N)
18     return mu, se
19
20 def mc_antithetic(N=200_000, rng=rng):
21     # Wir brauchen nur N/2 Basis-Ziehungen
22     M = N // 2
23     u = rng.random(M)
24
25     # Antithetische Paare auf U(0,1)-Ebene
26     ua = 1.0 - u
27
```

Antithetik: Gewinn vs. Naiv iii

```
28     # Transformation auf Normalverteilung (monoton via PPF)
29     z = norm.ppf(u)
30     za = norm.ppf(ua) # za = -z
31
32     # Auswertung
33     gz = g_of_z(z)
34     gza = g_of_z(za)
35
36     # Bilde den Mittelwert der Paare
37     paired_samples = 0.5 * (gz + gza)
38
39     mu = paired_samples.mean()
40     # SE wird von den N/2 Paarmittelwerten berechnet
41     se = paired_samples.std(ddof=1) / np.sqrt(M)
42     return mu, se
```

Antithetik: Gewinn vs. Naiv iv

```
43
44 mu_n, se_n = mc_naive()
45 mu_a, se_a = mc_antithetic()
46 print(f"Wahrheit: {TRUTH:.6f}")
47 print(f"Naiv: mu={mu_n:.6f}, SE={se_n:.4e}")
48 print(f"Anti: mu={mu_a:.6f}, SE={se_a:.4e}")
49 print(f"      (Gewinnfaktor: {se_n/se_a:.2f}x)")
```

Antithetik: Praxis-Hinweise

- Wirksam bei (nahezu) monotonem Integranden.
- Ebenfalls gut, wenn wenige starke³ Zufallsquellen die Ausgabe treiben.

Bsp. für mehrere starke Zufallsquellen: $g_1(X) = \frac{1}{10} \sum_{i=1}^{10} X_i$

Bsp. für wenige starke Zufallsquellen: $g_2(X) = 3X_1 + \frac{1}{10} \sum_{i=2}^{10} X_i$

- Diagnostik: SE_{anti} vs. SE_{naiv} vergleichen; Gewinn in % ausweisen.
- Optional: $\text{Corr}(g(U), g(1-U)) < 0$ prüfen.
- Bei nicht-monotonen/oszillierenden Reaktionen: meist kein Gewinn, selten Schaden.

³hoher Varianzbeitrag

Verschiedene Methoden zur Varianzreduktion:

- ☒ Antithetische Variablen
- ☐ **Control Variates**
- ☐ Stratified Sampling & Latin Hypercube
- ☐ Common Random Numbers
- ☐ Faktorielles Design & Reproduktionspläne
- ☐ Robustheit & Reproduzierbarkeit

Control Variates

Control Variates

Idee: Wissen nutzen, um Rauschen zu reduzieren

- **Ausgangslage:** Wir wollen $\mu_g = \mathbb{E}[g(X)]$ schätzen. $g(X)$ hat eine hohe Varianz σ_g^2 .
- **Zusatzwissen:** Nehmen wir an, es gibt eine andere Funktion $h(X)$,
 1. die **stark** mit $g(X)$ korreliert (z.B. $\rho_{gh} \approx 0.9$).
 2. deren Erwartungswert $\mu_h = \mathbb{E}[h(X)]$ wir **exakt kennen** (z.B. analytisch).
- **Beispiel (aus Kap 2):** $X \sim \mathcal{U}(0, 1)$
 - $g(X) = \sqrt{X}$. $\mu_g = \int_0^1 \sqrt{x} dx = 2/3$. (Tun wir so, als wüssten wir das nicht).
 - $h(X) = X$. Wir wissen exakt: $\mu_h = \mathbb{E}[X] = 0.5$.
 - \sqrt{X} und X sind offensichtlich stark korreliert.
- **Idee:** Wenn unsere Stichprobe zufällig "zu hoch" ausfällt (d.h. $\bar{h}_N > \mu_h$), dann wird \bar{g}_N wahrscheinlich auch "zu hoch" sein. Wir können diesen Fehler *korrigieren*.

Control Variates: Herleitung des Schätzers

- Wir definieren einen neuen, *korrigierten* Schätzer $Z(\beta)$ für einen beliebigen Koeffizienten β :

$$Z(\beta) = g(X) - \beta \cdot (h(X) - \mu_h)$$

- Dieser Schätzer ist **immer unverzerrt**, egal wie wir β wählen:

$$\mathbb{E}[Z(\beta)] = \mathbb{E}[g(X)] - \beta \cdot (\mathbb{E}[h(X)] - \mu_h) = \mu_g - \beta \cdot (\mu_h - \mu_h) = \mu_g$$

- Unser neuer MC-Schätzer ist $\hat{\mu}_{CV} = \frac{1}{N} \sum Z_i(\beta)$.

Control Variates: Wahl von β

- **Ziel:** Wähle β so, dass $\text{Var}(Z(\beta))$ minimal wird.

$$\text{Var}(Z(\beta)) = \text{Var}(g(X) - \beta h(X)) = \text{Var}(g) + \beta^2 \text{Var}(h) - 2\beta \text{Cov}(g, h)$$

- Das ist eine quadratische Funktion in β . Wir finden das Minimum per Ableitung:

$$\frac{d}{d\beta} \text{Var}(Z(\beta)) = 2\beta \text{Var}(h) - 2\text{Cov}(g, h) \stackrel{!}{=} 0$$

- Der optimale Koeffizient β^* ist:

$$\beta^* = \frac{\text{Cov}(g(X), h(X))}{\text{Var}(h(X))}$$

- Das ist exakt der Koeffizient einer linearen Regression von g auf h !

Control Variates: Wie hoch ist der optimale Varianzgewinn?

Übung

Setzen Sie das optimale β^* wieder in die Varianzformel ein um den Varianzgewinn zu quantifizieren:

Control Variates: Beispiel für Varianzgewinn

Übung

Berechnen Sie den Varianzgewinn für $\rho_{gh} = 0.9$:

Control Variates: Bestimmung von β in der Praxis

In der Praxis kennen wir $\text{Cov}(g, h)$ und $\text{Var}(h)$ (und $\text{Var}(g)$) nicht.

Lösung:

- Wir schätzen sie aus denselben N Samples, die wir sowieso ziehen!

$$\hat{\beta} = \frac{\widehat{\text{Cov}}(g, h)}{\widehat{\text{Var}}(h)} = \frac{\sum (g_i - \bar{g})(h_i - \bar{h})}{\sum (h_i - \bar{h})^2}$$

- Der finale Schätzer ist dann:

$$\hat{\mu}_{\text{CV}} = \bar{g} - \hat{\beta}(\bar{h} - \mu_h)$$

Dieser Schätzer ist *fast* unverzerrt. Ein winziger Bias entsteht durch die Schätzung von $\hat{\beta}$, der aber mit $1/N$ verschwindet und fast immer vernachlässigbar ist.

Control Variates: Was, wenn mehr als eine Control Variate bekannt ist?

- Hat man k Control Variates $\mathbf{h} = (h_1, \dots, h_k)^\top$ mit bekanntem Vektor $\mu_{\mathbf{h}}$, wird

$$\hat{\mu}_{CV} = \bar{g} - \hat{\beta}^\top (\bar{\mathbf{h}} - \mu_{\mathbf{h}}).$$

- $\hat{\beta}$ ist hier einfach der Vektor der Koeffizienten aus einer **Multiplen Linearen Regression** (OLS) von g auf \mathbf{h} .
- Diagnostik: Bericht von geschätztem β , Korrelationen und beobachtetem Varianzgewinn.

Control Variates: Python Demo i

- Schätze $\mu_g = \mathbb{E}[\sqrt{X}]$ für $X \sim \mathcal{U}(0, 1)$ (wahrer Wert $2/3$).
- Wir nutzen $h(X) = X$ als Control Variate mit bekanntem $\mu_h = 0.5$.

```
1 import numpy as np
2
3 TRUTH = 2/3
4 MU_H = 0.5 # E[X] für  $X \sim U(0,1)$ 
5 SEED = 42
6 rng = np.random.default_rng(SEED)
7
8 def mc_naive_sqrt(N=50_000, rng=rng):
9     X = rng.random(N)
10    g = np.sqrt(X)
11    mu_g_naiv = g.mean()
12    se_g_naiv = g.std(ddof=1) / np.sqrt(N)
```

Control Variates: Python Demo ii

```
13     return mu_g_naiv, se_g_naiv
14
15 def mc_control_variate_sqrt(N=50_000, rng=rng):
16     X = rng.random(N)
17     g = np.sqrt(X) # Zielgröße  $g(X)$ 
18     h = X          # Control Variate  $h(X)$ 
19
20     # Schätze beta via OLS (oder Kovarianz/Varianz)
21     # np.cov gibt die 2x2 Kovarianzmatrix zurück
22     cov_matrix = np.cov(g, h)
23     beta_hat = cov_matrix[0, 1] / cov_matrix[1, 1]
24
25     # Berechne den CV-Schätzer
26     mu_cv = g.mean() - beta_hat * (h.mean() - MU_H)
27
```

Control Variates: Python Demo iii

```
28     # SE-Schätzung ist komplexer, da beta geschätzt wurde.
29     # Einfachste (und robuste) Methode: SE der korrigierten Werte
30     z = g - beta_hat * (h - MU_H)
31     se_cv = z.std(ddof=1) / np.sqrt(N)
32
33     return mu_cv, se_cv, beta_hat
34
35 mu_n, se_n = mc_naive_sqrt()
36 mu_c, se_c, beta = mc_control_variate_sqrt()
37
38 print(f"Wahrheit: {TRUTH:.6f}")
39 print(f"Naiv: mu={mu_n:.6f}, SE={se_n:.4e}")
40 print(f"CV:    mu={mu_c:.6f}, SE={se_c:.4e} (beta={beta:.3f})")
41 print(f"Gewinnfaktor: {se_n/se_c:.2f}x")
```

Control Variantes in der Praxis (1/3)

Wie finde ich gute Control Variates?

Grundregel

Gesucht wird eine Hilfsgröße $h(X)$, die zwei Bedingungen erfüllt:

1. $\mu_h = \mathbb{E}[h(X)]$ ist **exakt bekannt** (analytisch, nicht geschätzt!).
2. $h(X)$ ist **stark korreliert** mit unserer Zielgröße $g(X)$ (d.h. ρ_{gh} ist nahe $+1$ oder -1).

Trick 1: Inputs als CVs nutzen

Wenn $g(X_1, \dots, X_d)$ die Zielgröße ist und die $\mathbb{E}[X_i]$ bekannt sind (z.B. 0 für $\mathcal{N}(0, 1)$), sind die Inputs selbst oft exzellente CVs.

- **Ziel:** $\mu_g = \mathbb{E}[e^{X_1+X_2}]$, mit $X_1, X_2 \sim \mathcal{N}(0, 1)$.
- **CVs:** Wähle $h_1(X) = X_1$ (mit $\mu_{h1} = 0$) und $h_2(X) = X_2$ (mit $\mu_{h2} = 0$).
- **Intuition:** Wenn X_1 zufällig hoch (> 0) ist, wird $e^{X_1+X_2}$ tendenziell auch hoch sein. Die CV korrigiert diesen Zufallstreffer.

Control Variantes in der Praxis (1/3)

Wie finde ich gute Control Variates?

Trick 2: Vereinfachte Modelle / Approximationen

- **Ziel (Finanzen):** Preis einer *Asiatischen Option* $g(X)$ (komplex, Pfad-abhängig, keine Formel).
- **CV (Finanzen):** Preis einer *Europäischen Option* $h(X)$ (einfach, Black-Scholes-Formel μ_h ist bekannt).
- **Intuition:** Beide Preise hängen stark vom selben unterliegenden Aktienkurs X ab ($\rho_{gh} \approx 0.9+$). Die CV fängt den Großteil der Marktbewegung ein.
- **Allgemein:** Nutze eine Linearisierung (z.B. Taylor-Approximation 1. Ordnung) als $h(X)$, falls deren Erwartungswert berechenbar ist.

Control Variantes in der Praxis (2/3)

Fallstricke & Warnsignale

Fallstrick 1: Die Bias-Falle (Der häufigste Fehler!)

Der CV-Schätzer ist $\hat{\mu}_{CV} = \bar{g} - \hat{\beta}(\bar{h} - \mu_h)$.

- Wenn der Wert für μ_h **falsch** ist (z.B. 0.51 statt $\mu_h = 0.50$), wird der Schätzer **verzerrt (biased)**!

$$\mathbb{E}[\hat{\mu}_{CV}] = \mathbb{E}[\bar{g}] - \mathbb{E}[\hat{\beta}(\bar{h} - \mu_h^{\text{falsch}})] \approx \mu_g - \beta^*(\mu_h - \mu_h^{\text{falsch}}) \neq \mu_g$$

- Ein kleiner Bias ist oft schlimmer als eine große Varianz, da er auch mit $N \rightarrow \infty$ nicht verschwindet.
- **Regel:** μ_h muss analytisch exakt sein. Es darf *niemals* aus einer anderen (Pilot-)Simulation geschätzt werden!

Control Variantes in der Praxis (2/3)

Fallstricke & Warnsignale

Fallstrick 2: "Estimation Risk" (Qualität > Quantität)

- **Problem:** Man ist versucht, viele (k) mögliche CVs $\mathbf{h} = (h_1, \dots, h_k)$ in die OLS-Regression zu werfen.
- **Aber:** Wir müssen den Koeffizienten-Vektor β aus denselben N Daten schätzen.
- Wenn k groß ist (z.B. $k = 20$) und N klein (z.B. $N = 100$), wird $\hat{\beta}$ sehr ungenau geschätzt (hohe Varianz).
- Diese Unsicherheit in $\hat{\beta}$ ("Estimation Risk") kann den Varianzgewinn der CVs zunichte machen oder die Varianz **sogar erhöhen!**
- **Tipp:** Lieber 1-2 CVs mit hoher Korrelation ($|\rho| > 0.7$) als 10 CVs mit schwacher Korrelation ($|\rho| \approx 0.1$).

Control Variantes in der Praxis (3/3)

Diagnose & Reporting

Der Schätzer $\hat{\mu}_{CV}$ ist kompliziert, da $\hat{\beta}$ selbst eine Zufallsvariable ist, die von \bar{g} und \bar{h} abhängt.

Problem: Wie schätzt man das $SE(\hat{\mu}_{CV})$?

Ansatz 1 (Einfach, oft OK):

- Berechne die korrigierten Werte $Z_i = g_i - \hat{\beta}(h_i - \mu_h)$ und nutze deren Standardfehler:

$$\widehat{SE}_{\text{einfach}} = \frac{s_Z}{\sqrt{N}}$$

Dies ignoriert die Unsicherheit von $\hat{\beta}$ und ist oft leicht "zu optimistisch" (zu klein).

Control Variantes in der Praxis (3/3)

Diagnose & Reporting

Der Schätzer $\hat{\mu}_{CV}$ ist kompliziert, da $\hat{\beta}$ selbst eine Zufallsvariable ist, die von \bar{g} und \bar{h} abhängt.

Problem: Wie schätzt man das $SE(\hat{\mu}_{CV})$?

Ansatz 2 (Robust, empfohlen): Replikation.

- Führe die *gesamte* Simulation (inkl. der Schätzung von $\hat{\beta}$) R -mal (z.B. $R = 20$) mit unabhängigen Seeds durch.
 - Du erhältst R Schätzwerte: $\hat{\mu}_{CV}^{(1)}, \dots, \hat{\mu}_{CV}^{(R)}$.
 - Der finale Schätzer ist $\bar{\mu}_{CV} = \frac{1}{R} \sum \hat{\mu}_{CV}^{(r)}$.
 - Der (glaubwürdige) Standardfehler dieses Schätzers ist:

$$\widehat{SE}_{\text{robust}} = \frac{s_{\mu, CV}}{\sqrt{R}} = \sqrt{\frac{1}{R(R-1)} \sum_{r=1}^R (\hat{\mu}_{CV}^{(r)} - \bar{\mu}_{CV})^2}$$

Control Variantes in der Praxis (3/3)

Diagnose & Reporting

Checkliste für das Reporting

Um zu beweisen, dass die CV-Methode funktioniert hat, berichten Sie immer:

- ✓ Die Baseline: \widehat{SE}_{naiv} (aus denselben Daten oder Pilotlauf).
- ✓ Das Ergebnis: \widehat{SE}_{CV} (robust geschätzt!).
- ✓ Den "Beweis": Die geschätzte Korrelation $\hat{\rho}_{gh}$ (z.B. $\hat{\rho} = 0.87$).
- ✓ (Optional): Den Koeffizienten $\hat{\beta}$ (z.B. $\hat{\beta} = 1.23$).
- ✓ Den Gewinn: Entweder als Faktor ($\widehat{SE}_{naiv}/\widehat{SE}_{CV}$) oder als Effizienz-Metrik ($SE^2 \times \text{Zeit}$).

Verschiedene Methoden zur Varianzreduktion:

- ☒ Antithetische Variablen
- ☒ Control Variates
- ☐ Stratified Sampling & Latin Hypercube
- ☐ Common Random Numbers
- ☐ Faktorielles Design & Reproduktionspläne
- ☐ Robustheit & Reproduzierbarkeit

Stratified Sampling & Latin Hypercube Sampling (LHS)

Stratified Sampling: Grundidee

- Teile die Domäne (den Input-Raum) in L disjunkte Regionen ("Strata"⁴) S_1, \dots, S_L , die den gesamten Raum abdecken.
- Sei $w_\ell = \mathbb{P}(X \in S_\ell)$ das (bekannte) Gewicht des Stratums (z.B. die Fläche/Volumen). Es gilt $\sum w_\ell = 1$.
- Der Gesamterwartungswert ist die Summe der gewichteten Erwartungswerte in jedem Stratum:

$$\mu = \mathbb{E}[g(X)] = \sum_{\ell=1}^L \mathbb{E}[g(X)|X \in S_\ell] \cdot \mathbb{P}(X \in S_\ell) = \sum_{\ell=1}^L \mu_\ell \cdot w_\ell$$

⁴Lateinisch Stratum = Schicht (Gruppe) aus Statistik 1 im ersten Semester

Stratified Sampling: Wie hilft uns das?

MC-Strategie:

- Statt N Punkte *irgendwo* zu ziehen, weisen wir jedem Stratum ℓ eine feste Anzahl n_ℓ Samples zu ($\sum n_\ell = N$).
- Wir schätzen μ_ℓ mit $\bar{g}_\ell = \frac{1}{n_\ell} \sum_{i=1}^{n_\ell} g(X_{\ell,i})$, wobei $X_{\ell,i} \sim F(x|X \in S_\ell)$, das heißt jede Schicht hat eine eigene Verteilung
- Der stratifizierte Schätzer ist dann

$$\hat{\mu}_{\text{strat}} = \sum_{\ell=1}^L w_\ell \cdot \bar{g}_\ell$$

- **Einfluss auf die Monte-Carlo Varianz:** Da die \bar{g}_ℓ unabhängig sind:

$$\text{Var}(\hat{\mu}_{\text{strat}}) = \sum_{\ell=1}^L w_\ell^2 \cdot \text{Var}(\bar{g}_\ell) = \sum_{\ell=1}^L w_\ell^2 \cdot \frac{\sigma_\ell^2}{n_\ell}$$

wobei $\sigma_\ell^2 = \text{Var}(g(X)|X \in S_\ell)$ die Varianz *innerhalb* des Stratums ℓ ist.

Stratified Sampling: Allokation

Wie ist das Gesamtbudget N auf die Strata zu verteilen (d.h. wie wählen wir die n_ℓ)?

1. Proportionale Allokation

- **Idee:** Ziehe proportional zur Größe des Stratums.
- **Formel:** $n_\ell = N \cdot w_\ell$
- **Vorteil:** Einfach. Garantiert besser (oder gleich gut) als naives MC.
- **Varianz:**

$$\sum_{\ell=1}^L w_\ell \cdot \sigma_\ell^2$$

2. Neyman-Allokation (Optimal)

- **Idee:** Investiere mehr Samples in Strata, die *groß* (w_ℓ) und *unsicher* (σ_ℓ) sind.
- **Formel:** $n_\ell \propto N \cdot w_\ell \sigma_\ell$
- **Vorteil:** Minimal mögliche Varianz für ein gegebenes N .
- **Nachteil:** Benötigt Schätzungen der σ_ℓ (z.B. aus einem Pilotlauf).
- **Varianz:**

$$\frac{1}{N} \left(\sum_{\ell=1}^L w_\ell \sigma_\ell \right)^2$$

Stratified Sampling: Allokation

Warum funktioniert das?

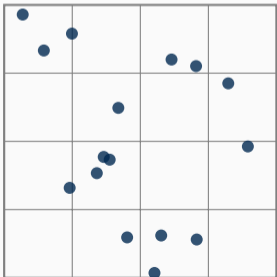
Die Gesamtvarianz σ_{naiv}^2 zerfällt in Varianz *innerhalb* der Strata und Varianz *zwischen* den Strata. Stratified Sampling eliminiert die "Zwischen-Strata-Varianz" komplett. Der Gewinn ist am größten, wenn die μ_ℓ (Strata-Mittelwerte) sehr unterschiedlich sind.

Die Varianz sinkt, wenn die innerhalb-Strata-Variabilität kleiner ist als die Gesamtvariabilität.

Visualisierung: Naiv vs. Stratified

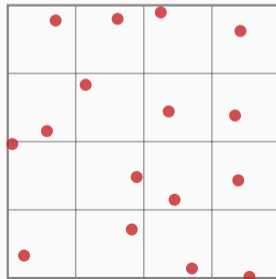
Naives MC (N=16)

- Punkte können klumpen.
- Einige Regionen (z.B. links unten) werden vielleicht gar nicht getroffen.
- Die Abdeckung ist rein zufällig.



Stratified MC (N=16, L=16)

- Domäne in 16 (4x4) Strata geteilt.
- Proportionale Allokation:
 $n_\ell = N \cdot w_\ell = 16 \cdot (1/16) = 1.$
- Wir ziehen **genau einen** Sample pro Zelle.
- Perfekte, garantierte Abdeckung.



Stratified Sampling: Problem

Anzahl der "Zellen" (Strata) explodiert mit der Anzahl der Dimensionen d ("**Fluch der Dimensionen**" (Curse of Dimensionality)).

Beispiel

- Wir wollen $d = 5$ Input-Faktoren untersuchen (z.B. $X_1, \dots, X_5 \sim \mathcal{U}(0, 1)$).
- Wir entscheiden uns, jede Achse in nur $L = 3$ Bereiche zu unterteilen (niedrig, mittel, hoch).
- Anzahl der resultierenden Strata (Zellen): $L^d = 3^5 = \mathbf{243}$ Zellen.
- Wenn unser Rechenbudget nur $N = 1000$ Simulationen erlaubt, haben wir im Schnitt nur ≈ 4 Samples pro Zelle.
- Bei $L = 2$ ($2^5 = 32$) ginge es, aber bei $d = 10$ ($2^{10} = 1024$) scheitert es wieder.

Schlussfolgerung

Wir brauchen eine Methode, die den Raum gut abdeckt, *ohne* dass die Anzahl der "Zellen" mit d explodiert.

Latin Hypercube Sampling: Idee

Die Idee: Nur die 1D-Ränder stratifizieren

LHS garantiert, dass die *Projektion* (der "Schatten") der Stichprobe auf *jede einzelne Achse* perfekt stratifiziert ist. Es stratifiziert nicht das d -dimensionale Volumen.

Ziel: Erzeuge N Stichprobenpunkte $\mathbf{X}_i \in [0, 1]^d$.

Latin Hypercube Sampling: Algorithmus (Pseudo-Code)

1. **Basis-Gitter erstellen:** Teile $[0, 1]$ in N gleiche Intervalle: $[\frac{0}{N}, \frac{1}{N}), \dots, [\frac{N-1}{N}, \frac{N}{N}]$.

2. **Samples pro Dimension:**

- Für Dimension 1: Ziehe *genau einen* Punkt U_i aus jedem der N Intervalle:

$$x_k = \frac{k - 1 + U_k}{N}, \quad U_k \sim \mathcal{U}(0, 1), \quad k = 1, \dots, N.$$

- Für Dimension 2: Ziehe ebenfalls *genau einen* Punkt V_i aus jedem der N Intervalle.
- ...
- Für Dimension d : Ziehe W_i ...

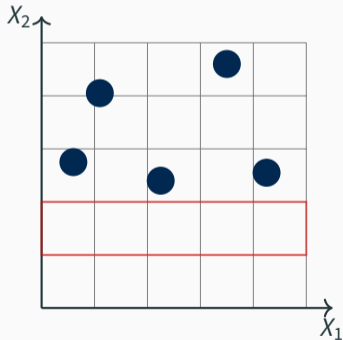
3. **Zusammenbau (der entscheidende Schritt):**

- Erstelle die $N \times d$ Matrix der Samples.
- **WICHTIG:** Behalte die Reihenfolge der Samples für Dimension 1 bei (z.B. sortiert).
- **Permutiere (mische)** die Reihenfolge der Samples für Dimension 2, 3, ..., d zufällig.

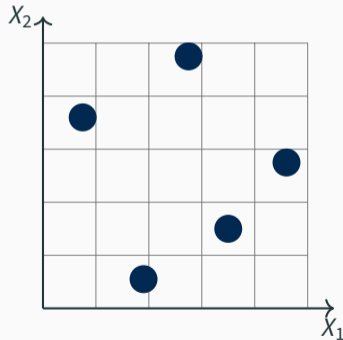
4. **Ergebnis:** Man erhält N Punkte \mathbf{X}_i . Der i -te Punkt ist z.B. $(U_i, V_{\pi_2(i)}, \dots, W_{\pi_d(i)})$, wobei π_j eine zufällige Permutation von $\{1, \dots, N\}$ ist.

Latin Hypercube Sampling: Visualisierung ($N = 5$)

Naives Monte Carlo ($N = 5$)



Latin Hypercube ($N = 5$)



Latin Hypercube Sampling: Visualisierung ($N = 5$)

Naives Monte Carlo ($N = 5$)

- Punkte können klumpen.
- Manche "Zeilen" oder "Spalten" (Projektionen) werden gar nicht getroffen (hier: 2. Zeile von unten leer).
- Reine Zufallsabdeckung.

Latin Hypercube ($N = 5$)

- Das 5x5-Gitter wird als Referenz gezeigt.
- Es gibt *genau einen* Punkt in jeder Zeile und *genau einen* Punkt in jeder Spalte.
- Dies wird als "**N-Rooks**"-Problem bezeichnet.
- Die 1D-Ränder sind perfekt abgedeckt.

Fazit: LHS als robuster Standard

LHS ist oft der robustere Default für Computer-Experimente mit vielen Input-Dimensionen ($d > 2$), da es eine gute Raumabdeckung mit flexiblem N sicherstellt.

Latin Hypercube Sampling: Gewinn

Der Gewinn durch LHS hängt stark von der Struktur der Funktion $g(X)$ ab:

Wie stark ist die Varianzreduktion?

- **Best Case (Additive Funktionen):** Wenn die Funktion (nahezu) additiv ist, d.h. $g(X) \approx \sum_{j=1}^d g_j(X_j)$, ist der Gewinn **enorm**.
- **Worst Case (Starke Interaktionen):** Wenn die Funktion von Interaktionen dominiert wird (z.B. $g(X) = X_1 \cdot X_2 \cdot \dots \cdot X_d$), ist der Gewinn durch LHS gering.
- **Typischer Fall (Glatte Funktionen):** Für die meisten "glatten" (smooth) Funktionen in der Praxis ist die Varianz des LHS-Schätzers *nie schlechter* als die des naiven MC-Schätzers.

$$\text{Var}(\hat{\mu}_{\text{LHS}}) \leq \text{Var}(\hat{\mu}_{\text{naiv}})$$

Ein Gewinn ist also (fast) garantiert und kostet kaum zusätzlichen Rechenaufwand.

Latin Hypercube Sampling: Best-Case Beispiel

Best Case: Additive Funktionen

Ziel: Schätze $\mu = \mathbb{E}[g(X_1, X_2)]$ mit

$$g(X_1, X_2) = X_1 + X_2$$

(wobei $X_1, X_2 \sim \mathcal{U}(0, 1)$, der wahre Wert ist $\mu = 0.5 + 0.5 = 1.0$).

- **Warum LHS hier gewinnt:**
- LHS garantiert, dass der Mittelwert der N X_1 -Samples (\bar{X}_1) extrem nahe an 0.5 liegt (Fehler $\propto 1/N$, nicht $1/\sqrt{N}$).
- Dasselbe gilt für den Mittelwert \bar{X}_2 .
- Der LHS-Schätzer ist $\hat{\mu}_{\text{LHS}} = \frac{1}{N} \sum g(X_i) = \bar{X}_1 + \bar{X}_2$.
- Da \bar{X}_1 und \bar{X}_2 beide "fast perfekte" Mittelwerte haben, ist ihre Summe auch "fast perfekt".
- Die Varianz von $\hat{\mu}_{\text{LHS}}$ ist hier extrem klein, viel kleiner als $\text{Var}(\hat{\mu}_{\text{naiv}})$.

Latin Hypercube Sampling: Worst-Case Beispiel

Worst Case: Starke Interaktionen

Ziel: Schätze $\mu = \mathbb{E}[g(X_1, X_2)]$ mit

$$g(X_1, X_2) = X_1 \cdot X_2$$

(wobei $X_1, X_2 \sim \mathcal{U}(0, 1)$, der wahre Wert ist $\mu = 0.5 \cdot 0.5 = 0.25$).

- **Warum LHS hier (kaum) hilft:**
- LHS stratifiziert zwar die Ränder (die X_1 - und X_2 -Achsen) perfekt.
- Aber durch die zufällige *Permutation* (das Mischen der Spalten) kontrolliert es die *Kombinationen* nicht.
- Ein X_1 aus dem Intervall $[0, 0.1]$ (sehr klein) wird mit gleicher Wahrscheinlichkeit mit einem X_2 aus $[0, 0.1]$ (Produkt ≈ 0.01) oder einem X_2 aus $[0.9, 1.0]$ (Produkt ≈ 0.1) gepaart.
- Die Varianz des *Produkts* hängt stark von diesen zufälligen Kombinationen ab und wird durch die Rand-Stratifizierung kaum reduziert.

Python: LHS mit `scipy.stats.qmc`

`scipy.stats.qmc` (Quasi-Monte Carlo) ist das Standard-Tool hierfür.

```
1  import numpy as np
2  from scipy.stats import qmc, norm
3  SEED = 42
4  base_seed_seq = np.random.SeedSequence(SEED)
5
6  def target_2d(z1, z2):
7      """Eine glatte Testfunktion in 2D"""
8      return np.exp(-0.5 * (z1**2 + z2**2)) + z1 * z2
9
10 def compare_lhs_vs_random_2d(R=30, m=2000, base_seed_seq=base_seed_seq):
11
12     total_N = R * m
13
14     # 1 Seed für naive, R Seeds für die R LHS-Läufe
```

Python: LHS mit `scipy.stats.qmc` II

```
15 child_seeds = base_ss.spawn(R + 1)
16
17 # --- 1. Naiv (i.i.d. Normal) ---
18 # Nutzt das *gesamte* Budget N = R * m für einen fairen Vergleich
19 rng_naive = np.random.default_rng(child_seeds[0])
20 zr = rng_naive.standard_normal(size=(total_N, 2)) # Shape (N, 2)
21 gr = target_2d(zr[:, 0], zr[:, 1])
22 mu_r = gr.mean()
23 # SE für naives MC ist korrekt
24 se_r = gr.std(ddof=1) / np.sqrt(total_N)
25
26 # --- 2. LHS in 2D (mit Replikation für korrekte SE-Schätzung) ---
27 lhs_means = [] # Liste zum Sammeln der R Mittelwerte
28
29 for r in range(R):
```

Python: LHS mit `scipy.stats.qmc` iii

```
30      # Nimm den nächsten unabhängigen Seed für diese Replikation
31      current_seed = child_seeds[r + 1]
32
33      # WICHTIG: Sampler *in* der Schleife mit neuem Seed erstellen!
34      sampler = qmc.LatinHypercube(
35          d=2,
36          seed=current_seed.generate_state(1)[0]
37      )
38
39
40      # Ziehe m Punkte
41      U = sampler.random(n=m) # Shape (m, 2)
42
43      # Transformation
44      z1 = norm.ppf(U) # Shape (m, 2)
```

Python: LHS mit `scipy.stats.qmc` iv

```
45
46     # Auswertung
47     gl = target_2d(zl[:, 0], zl[:, 1])
48
49     # Berechne Mittelwert für diesen *einen* Lauf
50     mu_l_run = gl.mean()
51     lhs_means.append(mu_l_run)
52
53     # --- Auswertung der R Replikationen ---
54     lhs_means = np.array(lhs_means)
55
56     # Finaler Schätzer ist der Mittelwert der Mittelwerte
57     mu_l = lhs_means.mean()
58
59     # Der KORREKTE Standardfehler wird von den R Mittelwerten berechnet
```

Python: LHS mit `scipy.stats.qmc` V

```
60     # (da diese R Werte nun i.i.d. sind!)
61     se_l = lhs_means.std(ddof=1) / np.sqrt(R)
62
63     return (mu_r, se_r), (mu_l, se_l)
64
65     # --- Ausführung ---
66
67     R_REPS = 50           # 50 unabhängige Läufe
68     M_SAMPLES = 1000     # 1000 LHS-Samples pro Lauf
69     N_TOTAL = R_REPS * M_SAMPLES
70
71     (r_mu, r_se), (l_mu, l_se) = compare_lhs_vs_random_2d(
72         R=R_REPS,
73         m=M_SAMPLES,
74         base_seed_seq=base_seed_seq
```

Python: LHS mit `scipy.stats.qmc` vi

```
75 )
76
77 print(
78     f"Vergleich Total Samples N = {N_TOTAL} (R={R_REPS}, m={M_SAMPLES})"
79 )
80 print(f"-----")
81 print(f"Naiv (i.i.d.): mu={r_mu:.6f}, SE={r_se:.4e}")
82 print(f"LHS (Replik.): mu={l_mu:.6f}, SE={l_se:.4e}")
83 print(f"(Gewinnfaktor: {r_se/l_se:.2f}x)")
```

Entscheidungshilfe: Stratified Sampling vs. LHS

Die Wahl hängt stark von der Struktur Ihres Problems ab

- **Stratified Sampling** ist die erste Wahl, wenn...
 - ... der Input-Raum in *natürliche, klar definierte Gruppen* zerfällt (z.B. Regionen, Kundensegmente, Szenarien wie "hohe vs. niedrige Volatilität").
 - ... Sie wissen, dass die Varianz *zwischen* diesen Gruppen hoch ist (d.h. die Mittelwerte μ_ℓ der Gruppen stark unterschiedlich sind).
 - ... die Dimensionalität d klein genug ist, sodass L^d handhabbar bleibt.
- **Latin Hypercube Sampling (LHS)** ist ein robuster Default, wenn...
 - ... *viele Input-Faktoren* (d ist groß) beteiligt sind und Stratified Sampling am "Fluch der Dimensionen" scheitert.
 - ... es keine offensichtliche "natürliche" Gruppierung im Input-Raum gibt.
 - ... die Zielfunktion $g(X)$ (vermutlich) glatt und von den Rändern (1D-Projektionen) dominiert wird.

Kombination von Techniken

Tipp: Techniken sind kombinierbar!

Varianzreduktionstechniken schließen sich nicht gegenseitig aus. Die stärksten Effekte erzielt man oft durch Kombination:

- **LHS + Antithetik:** Führen Sie ein LHS-Design mit $N/2$ Punkten durch. Erzeugen Sie dann ein zweites, antithetisches LHS-Design ($1 - U$) und paaren Sie die Ergebnisse.
- **Stratified + Antithetik:** Wenden Sie innerhalb jedes Stratum ℓ antithetische Ziehungen an, um die Varianz σ_ℓ^2 innerhalb des Stratum zu senken.
- **LHS + Control Variates:** Nutzen Sie ein (effizientes) LHS-Design, um $g(X)$ und $h(X)$ zu evaluieren. Die CV korrigiert dann die verbleibende, durch Interaktionen getriebene Varianz.

Wichtig: Erfolg immer messen!

Egal welche Methode(n) Sie wählen:

Wichtig ist in allen Fällen eine klare Diagnose der Varianzreduktion gegenüber einer naiven Baseline.

(Berichten Sie immer $\widehat{SE}_{\text{neu}}$ vs. $\widehat{SE}_{\text{naiv}}$ und/oder die Effizienz-Metrik $SE^2 \times \text{Zeit.}$)

Verschiedene Methoden zur Varianzreduktion:

- ☒ Antithetische Variablen
- ☒ Control Variates
- ☒ Stratified Sampling & Latin Hypercube
- ☐ Common Random Numbers
- ☐ Faktorielles Design & Reproduktionspläne
- ☐ Robustheit & Reproduzierbarkeit

Common Random Numbers (CRN)

CRN: Das Problem – Vergleich von Szenarien

- Bisheriges Ziel: Schätze *einen* Wert, $\mu = \mathbb{E}[g(X)]$.
- **Neues, häufiges Ziel:** Vergleiche zwei Szenarien (z.B. "alte Policy" A vs. "neue Policy" B).
- Schätzung von $\Delta = \mu_A - \mu_B$
- **Naiver Ansatz:**
 1. Simulation von A mit N Samples (Seed 1) $\rightarrow \hat{\mu}_A$.
 2. Simulation von B mit N Samples (Seed 2) $\rightarrow \hat{\mu}_B$.
 3. Schätzung von $\hat{\Delta} = \hat{\mu}_A - \hat{\mu}_B$.
- **Problem:** Da $\hat{\mu}_A$ und $\hat{\mu}_B$ unabhängig sind:

$$\text{Var}(\hat{\Delta}_{\text{indep}}) = \text{Var}(\hat{\mu}_A) + \text{Var}(\hat{\mu}_B) = \frac{\sigma_A^2}{N} + \frac{\sigma_B^2}{N}$$

- Die Varianzen addieren sich! Wenn $\hat{\mu}_A$ zufällig "zu hoch" und $\hat{\mu}_B$ zufällig "zu niedrig" ausfällt, ist der geschätzte Unterschied $\hat{\Delta}$ riesig, obwohl der wahre Unterschied Δ klein sein mag.

CRN: Die Lösung – Gekoppelte Simulation

- **Idee:** Was A "zu hoch" macht, ist oft derselbe Zufallseinfluss (z.B. "hohe Nachfrage"), der auch B "zu hoch" macht.
- **Common Random Numbers (CRN):** Exakt *derselbe* Stream von Zufallszahlen $\{U_i\}$ für *beide* Simulationen.
- **Ablauf:**
 1. Ziehung U_1, \dots, U_N (Seed 1).
 2. Berechnung von $g_A(U_1), \dots, g_A(U_N) \rightarrow \hat{\mu}_A$.
 3. Berechnung von $g_B(U_1), \dots, g_B(U_N) \rightarrow \hat{\mu}_B$.
 4. Differenzen *pro Sample*: $D_i = g_A(U_i) - g_B(U_i)$.
 5. Schätzung von $\hat{\Delta}_{\text{CRN}} = \bar{D} = \frac{1}{N} \sum D_i$. (Was $\hat{\mu}_A - \hat{\mu}_B$ entspricht).
- **Der Varianz-Trick:**

$$\text{Var}(\hat{\Delta}_{\text{CRN}}) = \text{Var}(\hat{\mu}_A - \hat{\mu}_B) = \text{Var}(\hat{\mu}_A) + \text{Var}(\hat{\mu}_B) - 2 \cdot \text{Cov}(\hat{\mu}_A, \hat{\mu}_B)$$

- Da wir dieselben U_i verwenden, werden $\hat{\mu}_A$ und $\hat{\mu}_B$ **positiv korreliert** sein.
- ⇒ Der Kovarianz-Term wird *positiv* und *subtrahiert*, was die Gesamtvarianz drastisch senkt!

CRN: Korrekte Umsetzung

- **Synchronisation ist entscheidend!** Der i -te Zufallszahl muss in beiden Modellen für denselben Zweck verwendet werden.
 - **Gut:** 'rng.random()' für Ankunftszeit in A *und* Ankunftszeit in B.
 - **Schlecht:** 'rng.random()' für Ankunftszeit in A, aber für Servicezeit in B. Das bricht die Korrelation.
- **Best Practice:** Verwende separate, aber synchronisierte RNG-Streams für jeden stochastischen Prozess (z.B. einen Stream für Ankünfte, einen für Servicezeiten).
- **Diagnostik:** Immer die Korrelation $\rho(\hat{\mu}_A, \hat{\mu}_B)$ (oder $\rho(g_A, g_B)$) in einem Pilotlauf prüfen. Ist sie nicht stark positiv, ist das CRN-Setup fehlerhaft oder ungeeignet⁵.

⁵z. B. bei einer negative Korrelation

Warteschlangen-Beispiel

Vergleiche μ_A (1 Server) vs. μ_B (2 Server).

- **CRN:** Simuliere beide Systeme mit derselben Sequenz von Kunden-Ankunftszeiten und Service-Anforderungen.
- **Ergebnis:** Wenn zufällig ein "Kunden-Sturm" kommt (hohe Ankunftsrate), steigt die Wartezeit in *beiden* Systemen. Die *Differenz* der Wartezeiten bleibt jedoch relativ stabil.
- **Ohne CRN:** System A bekommt unter Umständen einen "Kunden-Sturm", System B zufällig eine "Flaute". Die geschätzte Differenz ist riesig und sagt nichts über den wahren Systemunterschied aus.

Python: CRN-Demo für zwei Szenarien i

Schätze $\Delta = \mathbb{E}[g_A(U)] - \mathbb{E}[g_B(U)]$

```
1 import numpy as np
2
3 SEED = 42
4 CRN_SEED = SEED + 1
5 rng_crn = np.random.default_rng(CRN_SEED)
6
7
8 def simulate_A(U):
9     # Szenario A: z.B. einfacher Prozess
10    return np.exp(U)
11
12 def simulate_B(U):
13     # Szenario B: leicht modifizierter Prozess (z.B. teurer)
14    return np.exp(U) + 0.1 * U**2 # Fügt einen kleinen Extra-Term hinzu
```

Python: CRN-Demo für zwei Szenarien ii

```
15
16 def compare_deltas(N=50_000, SEED=SEED, rng_crn=rng_crn):
17
18     # --- 1. Mit CRN ---
19     # *Ein* Set von Zufallszahlen
20     U_crn = rng_crn.random(N)
21     gA_crn = simulate_A(U_crn)
22     gB_crn = simulate_B(U_crn)
23
24     # Bilde Differenzen *pro Sample*
25     D_crn = gA_crn - gB_crn
26     mu_delta_crn = D_crn.mean()
27     se_delta_crn = D_crn.std(ddof=1) / np.sqrt(N)
28     corr_crn = np.corrcoef(gA_crn, gB_crn)[0, 1]
29
```

Python: CRN-Demo für zwei Szenarien iii

```
30      # --- 2. Ohne CRN (Unabhängig) ---
31      rng_A = np.random.default_rng(SEED)
32      rng_B = np.random.default_rng(SEED + 1)
33      U_A = rng_A.random(N)
34      U_B = rng_B.random(N)
35
36      gA_ind = simulate_A(U_A)
37      gB_ind = simulate_B(U_B)
38
39      # SE der Differenz unabhängiger Mittelwerte
40      se_A_ind = gA_ind.std(ddof=1) / np.sqrt(N)
41      se_B_ind = gB_ind.std(ddof=1) / np.sqrt(N)
42      se_delta_ind = np.sqrt(se_A_ind**2 + se_B_ind**2)
43
44
```

Python: CRN-Demo für zwei Szenarien iv

```
45 print(f"CRN Korrelation: {corr_crn:.4f}")
46 print(f"Delta (CRN): {mu_delta_crn:.6f}, SE={se_delta_crn:.4e}")
47 print(f"Delta (Indep): {gA_ind.mean() - gB_ind.mean():.6f}")
48 print(f"    => SE={se_delta_ind:.4e}")
49 print(f"    => Gewinnfaktor: {se_delta_ind/se_delta_crn:.2f}x")
50
51 compare_deltas()
```

Verschiedene Methoden zur Varianzreduktion:

- ☒ Antithetische Variablen
- ☒ Control Variates
- ☒ Stratified Sampling & Latin Hypercube
- ☒ Common Random Numbers
- ☐ Faktorielles Design & Reproduktionspläne
- ☐ Robustheit & Reproduzierbarkeit

Faktorielles Design & Replikationspläne

Planung von Simulationsexperimenten

- Bisher:
 - Fokus auf *einem* Schätzer $\hat{\mu}$ oder
 - Differenz zweier Schätzer $\hat{\Delta}$ (Baseline vs. *Treatment*)
- **Realität:** Einfluss von vielen *kontrollierbaren* Faktoren (Design-Parameter) gleichzeitig verstehen – und wie diese miteinander interagieren:
 - Haupteffekt
 - Interaktionseffekte

Planung von Simulationsexperimenten

Beispiel: Warteschlangen-System

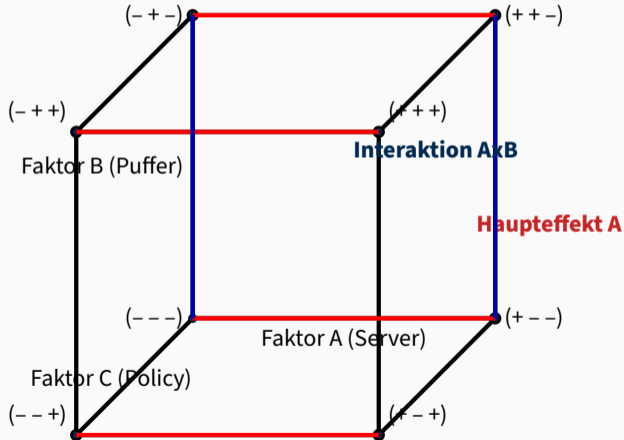
Wir wollen die mittlere Wartezeit $\mathbb{E}[W]$ minimieren. Wir können 3 Faktoren kontrollieren (jeweils 2 Stufen):

- **Faktor A (Server):** 2 Server ("niedrig") vs. 3 Server ("hoch")
- **Faktor B (Puffer):** 10 Plätze ("niedrig") vs. 50 Plätze ("hoch")
- **Faktor C (Policy):** "FIFO" ("niedrig") vs. "Priorität" ("hoch")

Fragen:

1. **Haupteffekt A:** Wie stark ändert sich $\mathbb{E}[W]$, wenn wir von 2 auf 3 Server wechseln (im Schnitt über B und C)?
2. **Interaktion AxB:** Ist der Effekt von mehr Servern (A) *anders*, wenn der Puffer (B) groß ist, als wenn er klein ist? (z.B. "Mehr Server nützen nichts, wenn der Puffer sowieso klein").

Planung von Simulationsexperimenten: Faktorielles Design



Faktorielles Design: Übung Alias-Struktur

Setting (3 Faktoren, $k = 3$):

- Faktor A: Ankunftsrate (niedrig/hoch)
- Faktor B: Service (langsam/schnell)
- Faktor C: Policy (aus/an)

Ein vollfaktorielles Design bräuchte $2^3 = 8$ Läufe. Wir haben nur Budget für 4 (ein 2^{3-1} Design). Wir wählen folgenden Plan und erhalten die Wartezeiten:

A (Ankunft)	B (Service)	C (Policy)	Wartezeit [min]
niedrig (-)	langsam (-)	aus (-)	12
hoch (+)	schnell (+)	aus (-)	6
niedrig (-)	schnell (+)	an (+)	8
hoch (+)	langsam (-)	an (+)	14

Faktorielles Design: Übung Alias-Struktur (1/3)

Übung

1. Berechnen Sie den (scheinbaren) **Haupteffekt von Faktor A** (Ankunftsrate). Der Haupteffekt ist definiert als: $\text{Effekt}_A = \text{Mittelwert}(A = +) - \text{Mittelwert}(A = -)$.
2. Interpretieren Sie das Ergebnis. Wirkt sich die Ankunftsrate aus?
3. Schauen Sie sich den Plan genau an. Fällt Ihnen eine "Vermischung" (Alias) auf? Z.B. zwischen A und der Interaktion BxC?

Konsequenz aus der Übung

Aliasing (Vermischung) bedeutet, dass wir den Effekt eines Faktors (z.B. A) nicht mehr sauber vom Effekt einer Interaktion (z.B. BxC) trennen können.

Faktorielles Design: Übung Alias-Struktur (2/3)

Übung

Lösung

Faktorielles Design: Übung Alias-Struktur (3/3)

Übung

Lösung

Wann ist Aliasing ein Problem?

Wann darf man Aliasing ignorieren?

- **Ziel: Black-Box-Optimierung.**
- Wenn Sie *nicht* verstehen wollen, *warum* eine Einstellung gut ist, sondern nur die **beste getestete Kombination** finden wollen.
- **Beispiel:** Wir testen 4 Einstellungen und finden, (+, -, +) bei 14 min ist die schlechteste. Wir wollen nur wissen, welche der 4 Kombinationen die *beste* Wartezeit (6 min) liefert.

Wann darf man Aliasing NICHT ignorieren?

- **Ziel: Systemverständnis (Ursache-Wirkung).**
- Sobald Sie die Frage stellen: "Hilft schnellerer Service (Faktor B) *an sich*, oder war es nur die Policy (Faktor C)?"
- In unserem Beispiel ist $\text{Effekt}_A = \text{Effekt}_{B \times C}$. Wir wissen nicht, ob die Ankunftsrate (A) wichtig ist oder die Interaktion von Service und Policy (BxC).

Fazit: Für das Verständnis von Zusammenhängen ist ein voll-faktorielles (oder ein "sauberes" teil-faktorielles) Design unerlässlich.

Experimentdesign: Blocking & Randomisierung

Problem: Störfaktoren (Nuisance Factors)

Was ist, wenn ein Faktor, den wir *nicht* kontrollieren können, unsere Ergebnisse beeinflusst? (z.B. Tageszeit, Server-Patch-Level, ...). Dieses "Hintergrundrauschen" kann unsere echten Faktor-Effekte überdecken.

Praxis-Tipp: Wann was?

- Ist ein klarer, nicht kontrollierbarer Störfaktor bekannt? ⇒ **Blocking** (und *innerhalb* der Blöcke randomisieren).
- Sind keine klaren Störfaktoren bekannt, aber es werden Zeit-/Systemeffekte befürchtet? ⇒ **Randomisieren** der Reihenfolge.

Experimentdesign: Blocking & Randomisierung

Ansatz 1: Blocking

- **Idee:** Gruppiere Läufe, die denselben bekannten Störfaktor teilen, in einen "Block".
- **Beispiel:** Störfaktor "Tageszeit".
 - Block 1 (Vormittag): Simuliere A vs. B.
 - Block 2 (Nachmittag): Simuliere A vs. B.
- **Ziel:** Vergleiche A und B *innerhalb* der Blöcke. Der (vielleicht große) Unterschied zwischen Vormittag und Nachmittag wird so herausgerechnet. Macht den Vergleich "fairer".

Ansatz 2: Randomisierung

- **Idee:** Verhindert systematische Verzerrungen durch *unbekannte* Störfaktoren (Trends, CPU-Last, Caching-Effekte, ...).
- **Beispiel:** Statt alle R Replikationen von Szenario A und dann alle R von B zu fahren, mischen wir die Reihenfolge:
- A, B, B, A, B, A, A, ...
- **Ziel:** Stellt sicher, dass ein unbekannter Trend (z.B. Server wird über Zeit langsamer) *beide* Szenarien (A und B) im Schnitt gleichmäßig trifft.

Faktorielles Design: Replikation

- **Replikationen (R) pro Zelle:** Jedes der 2^k Design-Settings (jede "Zelle") muss mehrfach (R -mal) mit *neuen, unabhängigen Seeds* laufen.
 - Dies ist nötig, um den *stochastischen Fehler* (die Varianz σ_{Zelle}^2) innerhalb jedes Settings zu schätzen.
 - Ohne Replikationen ($R = 1$) kann man den echten Faktor-Effekt nicht vom zufälligen Rauschen unterscheiden.
- **CRN für Differenzen:** Werden innerhalb einer Zelle zwei Politiken (z.B. A vs. B) miteinander verglichen, sollten diese R Läufe **mit CRN** (gekoppelte Seeds) erfolgen, um die Varianz der Differenz δ zu minimieren.

Minimal Detectable Effect: Planung

- **Frage:** Wie viele Replikationen R werden benötigt?
- **MDE:** Welcher Unterschied δ ist *praktisch relevant*? (z.B. "Eine Differenz von < 0.5 min Wartezeit spielt keine Rolle").
- **Pilot-Studie:** $R_0 = 10$ Läufe werden durchgeführt und die Standardabweichung der Differenz s_δ (mit CRN!) geschätzt.
- **Stichprobengröße (aus Kap. 2):** Benötigte Replikationen R für ein 95%-CI (mit $z \approx 1.96$), dessen halbe Breite $\varepsilon = \text{MDE}$ ist:

$$\varepsilon \geq 1.96 \cdot \frac{s_\delta}{\sqrt{R}} \quad \Rightarrow \quad R \geq \left(\frac{1.96 \cdot s_\delta}{\text{MDE}} \right)^2$$

- **Dokumentieren:** Seeds/Streams, R , MDE-Ziel, Pilotwerte, Einsatz von CRN.

Replikation & MDE: Beispielrechnung

- **Setting:** Betrachten die Zelle (Ankunft hoch, Service langsam) im vorherigen 2^k -Design.
- **Vergleich:** Policy A vs. Policy B.
- **MDE:** Ein Unterschied von $\text{MDE} = 0.5$ min ist die Nachweisgrenze.
- **Pilot (ohne CRN):** $R_0 = 20$ Läufe. $\hat{\mu}_A = 14.2$, $\hat{\mu}_B = 12.8$. $s_A = 2.1$, $s_B = 1.9$.
 - $s_\delta = \sqrt{s_A^2 + s_B^2} = \sqrt{2.1^2 + 1.9^2} \approx 2.83$ min.
 - Nötiges $R \geq (1.96 \cdot 2.83 / 0.5)^2 \approx \mathbf{123}$ Replikationen.
- **Pilot (mit CRN):** $R_0 = 20$ Läufe (gekoppelt).
 - $\hat{\mu}_A = 14.2$, $\hat{\mu}_B = 12.8$. (Mittelwerte sind gleich).
 - ABER: Die Differenzen $D_i = g_{A,i} - g_{B,i}$ werden direkt analysiert.
 - Wir finden $s_\delta = s_D = 0.8$ min (viel kleiner, da das Rauschen korreliert war).
 - Nötiges $R \geq (1.96 \cdot 0.8 / 0.5)^2 \approx \mathbf{10}$ Replikationen.
- **Fazit:** CRN reduziert den Replikationsaufwand (und damit die Rechenzeit) um >90%.

Budgetierung: Stichproben vs. Varianzreduktion

Effizienz = $\text{Var}(\hat{\mu}) \times \text{Zeit} \Rightarrow$ minimiere $\sigma^2 \cdot t$, wobei

- σ^2 : Varianz pro Ziehung (naiv vs. neu, e.g. Antithetik/CV/LHS/Strata/CRN).
- t : Zeit pro Ziehung (Struktur kann t leicht erhöhen).

Break-even: Reduktionsfaktor $r = \sigma_{\text{neu}}^2 / \sigma_{\text{naiv}}^2$, Zeitfaktor $c = t_{\text{neu}} / t_{\text{naiv}}$.

$$\frac{\text{Eff}_{\text{neu}}}{\text{Eff}_{\text{naiv}}} = r \cdot c \Rightarrow \text{lohnend, wenn } \mathbf{r \cdot c < 1}.$$

Beispiel: Antithetik

- Varianz halbiert: $r = 0.5$
- Code braucht 5% länger (Paarbildung): $c = 1.05$
- $r \cdot c = 0.5 \cdot 1.05 = 0.525 < 1$.
- **Lohnt sich!** Wir erreichen das Ziel in ca. 53% der naiven Zeit.

Verschiedene Methoden zur Varianzreduktion:

- ☒ Antithetische Variablen
- ☒ Control Variates
- ☒ Stratified Sampling & Latin Hypercube
- ☒ Common Random Numbers
- ☒ Faktorielles Design & Reproduktionspläne
- ☐ Robustheit & Reproduzierbarkeit

Robustheit & Reproduzierbarkeit

Checkliste: Design, Seeds & Robustheit

- **Baseline & Vorüberlegungen:**

- ☐ *Basismodell* korrekt implementiert? Falls nicht: Systematische Optimierung auf einen Fehler.
- ☐ Gibt es *Control Variates* mit bekanntem Erwartungswert? Falls ja, Schätzung von β und Bericht der beobachtete Varianzreduktion.
- ☐ Integrand *glatt oder (teilweise) monoton*? Falls ja, Test von Antithetik als ersten, sehr günstigen Schritt.
- ☐ Sind viele Input Variablen beteiligt? Start mit *LHS* als robustem Default, ggf. Kombination mit Antithetik und CV.

- **RNG/Seeds:**

- ☐ Welcher RNG-Algorithmus wurde verwendet? (z.B. `np.random.default_rng(PCG64)`)
- ☐ Was war der Haupt-Seed?
- ☐ **Seed-Strategie:** Wie wurden Seeds für Replikationen oder parallele Läufe erzeugt? (z.B. `SeedSequence.spawn()`)

Checkliste: Design, Seeds & Robustheit

- **CRN-Synchronisation:**

- ☐ Wurden *Common* Random Numbers (gleiche Seeds/Streams⁶) oder *Independent* Random Numbers (unterschiedliche Seeds) für den Vergleich von A vs. B verwendet?
- ☐ (Bei CRN): Ist sichergestellt, dass die Streams synchronisiert sind (z.B. Stream 1 *immer* für Ankünfte in A und B)?

- **Unabhängigkeit:**

- ☐ Sind die R Replikationen wirklich unabhängig (getrennte Seeds)?
- ☐ Wurden Abhängigkeiten aus Kap. 2 (z.B. Autokorrelation) durch Batch-Means behandelt?

- **Sensitivität:**

- ☐ Sind die Ergebnisse stabil gegenüber $\pm 10\%$ Parametervariation? Macht übermäßige Modellfragilität sichtbar.

⁶eindeutig identifizierte, reproduzierbare Folge

Checkliste: Diagnostik & Reporting

Jede Varianzreduktionstechnik braucht eine Diagnose zum Beweis, dass sie funktioniert.

- **Baseline:**

- ☐ Berichten Sie *immer* das \widehat{SE}_{naiv} als Referenz

- **Antithetik:**

- ☐ Berichten Sie \widehat{SE}_{anti} und den Gewinnfaktor $\widehat{SE}_{naiv}/\widehat{SE}_{anti}$
- ☐ (Optional): Schätzen Sie $\rho(g(X_i), g(X_a))$ (sollte negativ sein)

- **Control Variates (CV):**

- ☐ Berichten Sie \widehat{SE}_{cv} und den Gewinnfaktor
- ☐ Erstellen Sie Gewichtshistogramme, ESS-Schätzungen
- ☐ Berichten Sie das geschätzte $\hat{\beta}$ und Korrelation $\hat{\rho}_{gh}$. ($\hat{\rho}_{gh}$ nahe 0 zeigt eine ungeeignete CV)

Checkliste: Diagnostik & Reporting

- **LHS / Stratified:**

- ☐ Berichten Sie $\widehat{SE}_{\text{LHS}}$ vs. $\widehat{SE}_{\text{naiv}}$.⁷

- **Common Random Numbers (CRN):**

- ☐ Berichten Sie $\widehat{SE}_{\delta, \text{CRN}}$ vs. $\widehat{SE}_{\delta, \text{indep}}$.
- ☐ Berichten Sie die Korrelation $\hat{\rho}(g_A, g_B)$ (sollte stark positiv sein).

- **Form:**

- Alle Ergebnisse mit \widehat{SE} , CI, Rechenzeit und klaren Beschreibung der verwendeten Seeds, Streams und Designs reported?
- Repository aufgeräumt? Coding Standards eingehalten?

⁷Vorsicht: Die Formel s/\sqrt{N} ist für LHS/Stratified nicht exakt, oft wird Replikation der *gesamten* LHS-Simulation benötigt, um ein SE zu bekommen

Reporting: Mini-Template für Ergebnis-Tabellen

Eine gute Tabelle fasst die Effizienz aller Methoden zusammen. (Werte sind beispielhaft)

Table 1: Vergleich der Effizienz zur Schätzung von $\mu = \mathbb{E}[e^Z]$. (Budget: $N = 200\,000$, $R = 10$ Replikationen für SE-Schätzung der Schätzer).

Methode	N	$\hat{\mu}$	$\widehat{SE}(\hat{\mu})$	Zeit [s]	Effizienz ($\propto SE^2 \times \text{Zeit}$)
Naiv (Baseline)	200 000	1.6485	3.85e-03	0.50	1.00 (Ref)
Antithetik	200 000	1.6488	1.21e-03	0.52	0.10
Control Var. (h=Z)	200 000	1.6487	2.10e-03	0.55	0.33
LHS (1D)	200 000	1.6487	1.55e-03	0.58	0.19

- $\hat{\mu}$ ist der Mittelwert der 10 Replikations-Schätzer.
- $\widehat{SE}(\hat{\mu})$ ist der Standardfehler der 10 Replikations-Schätzer (s/\sqrt{R}).
- **Interpretation:** Antithetik ist hier (für e^Z) die mit Abstand effizienteste Methode. Sie liefert in ähnlicher Zeit (0.52s) eine $\approx 10\times$ bessere Effizienz (d.h. kleinere $SE^2 \times \text{Zeit}$).

Verschiedene Methoden zur Varianzreduktion:

- ✓ Antithetische Variablen
- ✓ Control Variates
- ✓ Stratified Sampling & Latin Hypercube
- ✓ Common Random Numbers
- ✓ Faktorielles Design & Reproduktionspläne
- ✓ Robustheit & Reproduzierbarkeit

Ausblick

Zusammenfassung & Takeaways

- Varianzreduktion ist der **zentrale Hebel** zur Effizienzsteigerung und adressiert den $\frac{1}{\sqrt{N}}$ -Flaschenhals, indem sie σ^2 senkt.
- **Antithetik** $((U, 1 - U))$ ist fast kostenlos und sehr effektiv bei *monotonen* Funktionen.
- **Control Variates** $(g - \beta(h - \mu_h))$ nutzen *Wissen* (μ_h) und *Korrelation* (ρ_{gh}) , um Rauschen zu eliminieren. Der Gewinn ist $(1 - \rho_{gh}^2)$.
- **LHS & Stratified Sampling** erzwingen eine gleichmäßige Abdeckung des Input-Raums und reduzieren die Varianz, die durch "Klumpenbildung" der Samples entsteht.
- **Common Random Numbers (CRN)** sind der Goldstandard für den *Vergleich* von Szenarien $(\mu_A - \mu_B)$, da sie die Varianz der *Differenz* durch positive Korrelation senken.
- Jede Technik muss **diagnostiziert** (Gewinnfaktor messen!) und **sauber dokumentiert** werden (Seeds, Streams, Design).

Ausblick: CV für das Coupon Collector's Problem

Erinnerung (Kap 2): Schätze $\mu = \mathbb{E}[T]$, wobei T die Anzahl gezogener Karten (Booster) bis *alle* n Typen gesammelt sind.

- **Problem:** T hat eine hohe Varianz, $\text{Var}(T) \approx n^2 \sum k^{-2} \approx n^2 \frac{\pi^2}{6}$. Für $n = 50$ ist $\sigma_T \approx 203$. Das naive $\text{SE}(\hat{\mu}_N) = \sigma_T / \sqrt{N}$ ist groß.
- **Control Variate** $h(X)$:
- Wähle eine feste Zahl m (z.B. $m = n$). Sei K_m die Anzahl *verschiedener* Typen, die wir nach genau m Ziehungen gesehen haben.

Ausblick: CV für das Coupon Collector's Problem

- **Control Variate** $h(X)$:
- Wähle eine feste Zahl m (z.B. $m = n$). Sei K_m die Anzahl *verschiedener* Typen, die wir nach genau m Ziehungen gesehen haben.
- K_m ist stark (negativ) mit T korreliert: Wenn wir nach $m = 50$ Zügen erst wenige Karten haben (K_m klein), wird die Gesamtzeit T wahrscheinlich sehr lang sein.
- Der Erwartungswert von K_m ist analytisch bekannt (klassische Besetzungsverteilung):

$$\mu_h = \mathbb{E}[K_m] = n \cdot \left(1 - \left(1 - \frac{1}{n}\right)^m\right)$$

- **CV-Schätzer:**

$$\hat{\mu}_{CV} = \bar{T} - \hat{\beta}(\bar{K}_m - \mu_h)$$

- **Ergebnis:** Diese Methode kann die Varianz um 90%–98% reduzieren und ist massiv effizienter als das naive Simulieren von T .