UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

SPECIALIZAREA INFORMATICĂ

Lucrare de licență

IMPORTANCE SAMPLING

Absolvent

Sabo Vlad-Andrei

Coordonator științific

Conf.dr. Iulian Cîmpean

București, iunie 2022

Importance Sampling
Lucrare de finalizare de studii

Sabo Vlad-Andrei

Iunie 2022

Cuprins

1 Introducere **5**
1.1 Scopul și motivația lucrării 5
1.2 Structura Lucrării 6

2 Preliminarii și concepte de baza **7**
2.1 Concepte Matematice: 7
Elemente de baza de probabilitati: 7
Concepte de Teoria Codurilor 9
2.2 Concepte Tehnice 10
Python 10
NumPy 10

3 Importance sampling **11**
3.1 Metoda propriu-zisă 11
3.2 Metoda Minimizării Variației 17
3.3 Metoda Cross-Entropiei 20

4 Aplicaţie la probleme de numărare **25**
4.1 Sudoku: definiții, interpretare matematică 25
4.2 Abordarea problemei de numărare în termeni probabiliști și algoritmici 26

5 Cod **31**

6 Concluzii **35**

7 Bibliografie **36**

Rezumat

Lucrarea de față este dedicată studiului unei metode clasice de simulare de distribuții, anume importance sampling. Prezentăm fundamentul matematic al acestui procedeu cât și metodele derivate din acesta, împreună cu o aplicație practică a metodei în probleme de numărare, în particular la problema numărului de Sudoku.

1 Introducere

1.1 Scopul și motivația lucrării

Simularea este un procedeu ce constă în imitarea unor evenimente reale pe o durată de timp, cu scopul de a afla mai multe informații despre evenimentele în cauză. Procedeul poate fi utilizat pe o gamă variată de domenii, cum ar fi educație, finanțe, medicină sau inteligență artificială, și cu o multitudine de scopuri. De exemplu, să luăm situația unui investitor. Acesta poate simula evoluția economiei mondiale pe o durată de timp pentru a afla ce companie va deveni profitabilă, ce industrie va da faliment sau când va veni următoarea criză economică. Reiese în mod trivial utilitatea, am putea afirma chiar necesitatea, simulării în societate. Întrebarea este, cum putem simula evenimente?

Există mai multe metode de simulare a evenimentelor, una dintre cele mai puternice fiind metoda Monte Carlo. Metoda a fost inventată de Stanislaw Ulam și John von Neumann în timpul celui de-al doilea război mondial și este denumită după cartierul din Monaco faimos pentru cazinouri și jocuri de noroc, Monte Carlo. Tehnica de simulare se bazează pe crearea unui model probabilistic folosindu-se de o distribuție cunoscută, pentru a afla probabilitatea unui eveniment prin simularea repetata a modelului cu variabile aleatoare din distribuția aleasă. De-a lungul timpului pe baza acestei metode s-au dezvoltat și alte procedee de simulare, astfel putem vorbi despre o familie de metode Monte Carlo.

Din această familie face parte și importance sampling, o metodă de simulare introdusă în anul 1949 de către Gerald Goertzel, Herman Kahn și Theodore E. Harris. Metoda se bazează pe construirea modelului de simulare prin intermediul unei distribuții auxiliare, din care putem extrage date și concluzii relevante pentru un model construit cu distribuția principală. Procedeul este unul fundamental în tehnicile de reducere a variației, fiind folosit pentru simularea evenimentelor rare dar nu numai. Acesta poate fi folosit, de pildă, în rezolvarea unor probleme de numărare unde o abordare naivă sau prin intermediul elementelor de combinatorică necesită un grad ridicat de efort.

Scopul lucrării este de a prezenta procedeul de importance sampling și aplicațiile sale în estimarea probabilităților rare. Pentru a exemplifica utilitatea procedeului, vom aborda o problemă de numărare, anume estimarea numărului de soluții Sudoku unice. Dorim să arătăm că deși această problemă se poate rezolva în mod determinist prin metode combinatorice, prin intermediul metodei de importance sampling putem ajunge la o aproximare foarte apropiată de rezultatul concret.

Motivațiile alegerii temei includ importanța simulării în viața cotidiană și pasiunea autorului pentru rezolvarea de Sudoku.

1.2 Structura Lucrării

Lucrarea este împărțită în două. Prima parte a lucrării se preocupă cu partea teoretică a procedeului de simulare. Capitolul doi este capitolul de preliminarii și are ca scop pregătirea cititorului în vederea înțelegerii lucrării, fiind expuse mai multe concepte de baza în probabilități. Se introduce și o parte de teoria codurilor, fiind utilă în rezolvarea problemei cu Sudoku. În capitolul trei se explică pe larg fundamentul matematic din spatele metodei de importance sampling și se prezintă adaptări mai noi ale metodei.

A doua parte a lucrării se preocupă cu aplicația practică. În capitolul patru se analizează un puzzle Sudoku în termeni matematici și se prezintă strategia de rezolvare a problemei folosind metode Monte Carlo și rezultatele rulării algoritmilor, atât pe propriul PC cât și pe alte unități mai performante. Capitolul cinci include codul de implementare al algoritmilor din capitolul precedent.

2 Preliminarii și concepte de baza

În această secțiune menționăm concepte, definiții și rezultate necesare înțelegerii și aplicării procedeului de importance sampling și a problemei de numărare, cât și tehnologiile utilizate în realizarea aplicației practice. Ca surse bibliografice menționăm [9], [3].

2.1 Concepte Matematice:

Elemente de baza de probabilitati:

Definiţia 2.1. *Spunem că* $\mathbb{P} : \mathcal{F} \to [0,1]$ *este o probabilitate pe spațiul măsurabil* $(\Omega, \mathcal{F})$ *dacă:*

1. $\mathbb{P}(\emptyset) = 0$
2. $\mathbb{P}(\Omega) = 1$
3. *Dacă* $(A_n)_n \subset \mathcal{F}$ *și* $A_n \cap A_m = \emptyset,\ \forall n \neq m$ *atunci* $\mathbb{P}(\cup_{n=1} A_n) = \sum_{n=1} \mathbb{P}(A_n)$

Tripletul $(\Omega, \mathcal{F}, \mathbb{P})$ *se numește spațiu de probabilitate.*

Definiţia 2.2. *Fie* $(\Omega, \mathcal{F})$ *si* $(\mathbf{R}, \mathcal{B})$ *două spații măsurabile. Spunem că aplicația* $X : \Omega \to \mathbf{R}$ *este o variabilă aleatoare (v.a.) dacă* $X^{-1}(B) \in \mathcal{F}$ *pentru orice* $B \in \mathcal{B}$*.*

Definiţia 2.3. *Analog, aplicația* $X : \Omega \to \mathbf{R}^n$ *este un vector aleator, unde* $X = (X_1, X_2, ..., X_n)$ *iar* $X_1, X_2, ..., X_n$ *sunt variabile aleatoare. Ne vom referi în lucrare la vectori și ca "eșantioane".*

Definiţia 2.4. *Fie* $(\Omega, \mathcal{F})$ *si* $(\mathbf{R}, \mathcal{B})$ *două spații măsurabile și* $X : \Omega \to \mathbf{R}$ *o variabilă aleatoare. Probabilitatea* $\mathbb{P} \circ X^{-1} : \mathcal{B} \to [0,1]$ *definită prin*

$$\mathbb{P} \circ X^{-1}(B) = \mathbb{P}(X^{-1}(B)),\ \forall B \in \mathcal{B}(\mathbb{R}^n) \tag{1}$$

se numește repartiția sau distribuția lui X.

Definiţia 2.5. *Fie* $(\Omega, \mathcal{F})$ *și* $(\mathbf{R}, \mathcal{B})$ *două spații măsurabile și* $X : \Omega \to \mathbf{R}$ *o variabilă aleatoare. Dacă repartiția lui X este absolut continuă în raport cu masura Lebesgue (notată cu* λ*), atunci conform teoremei Radon-Nykodim există o funcție non-negativă f astfel încât*

$$\mathbb{P} \circ X^{-1}(B) = \int_B f(x)dx, \quad \forall B \in \mathcal{B}(\mathbb{R}^n) \tag{2}$$

Funcția f se numește densitatea variabilei X.

Definiţia 2.6. *Fie $X : \Omega \to \{x_1, x_2, ..., x_n\}$ o variabilă aleatoare cu $\mathbb{P}(X = x_i) = p_i$. Definim media lui X ca*

$$\mathbb{E}[X] = \sum_{i=1}^{n} p_i x_i \tag{3}$$

Teorema 2.7. *(Teorema de Transport)*
Fie $X : \Omega \to \mathbb{R}$ cu o densitate de probabilitate p și $f : S \to \mathbb{R}$ o funcție măsurabilă astfel încât $f(X) : \Omega \to \mathbb{R}$ e integrabilă. Atunci avem

$$\mathbb{E}[f(X)] = \int_R f(x)p(x)dx \tag{4}$$

În particular, dacă X e integrabilă, avem:

$$\mathbb{E}[X] = \int_R xp(x)dx \tag{5}$$

Definiţia 2.8. *Înțelegem prin metode Monte Carlo o familie de algoritmi computaționali care se bazează pe simularea de experimente aleatoare pentru a estima rezultate deterministe.*

Să presupunem că vrem să estimăm

$$\int_{\mathbb{R}^d} H(x)f(x)dx \tag{6}$$

sau

$$\sum_{x_i \in \mathbf{R}} H(x_i)f(x_i) \tag{7}$$

, unde $\mathbf{R}$ este cel mult numărabil.

În ambele cazuri, presupunem că f este o densitate de probabilitate.

Atunci metoda Monte Carlo standard va avea următoarea formă:

$$\int_{\mathbb{R}^d} H(x)f(x)dx = \mathbb{E}[H(X)] \simeq \frac{1}{n}\sum_{i=1}^{n} H(X_i) \tag{8}$$

sau

$$\sum_{i \geq 1} H(x_i)f(x_i) = \mathbb{E}[H(X)] \simeq \frac{1}{n}\sum_{i=1}^{n} H(X_i) \tag{9}$$

unde $X \sim f$ și $x_1, \ldots, x_n, \ldots$ sunt independente și identic distribuite din f.

În această lucrare vom considera doar variabila aleatoare $X : \Omega \to \mathbf{R}$, unde $\mathbf{R} = \mathbb{R}^d$ sau o mulțime cel mult numărabilă, dacă nu se specifică altfel. De asemenea vom nota cu $\int Hf$ atât cazul $\int_{\mathbb{R}^d} H(x)f(x)dx$, cât și cazul $\sum_{i \geq 1} H(x_i)f(x_i)$ pentru $|R| \leq n$

Definiţia 2.9. *(Conform [9]) Fie g și h două densități peste același spațiu măsurabil. Definim cross-entropia Kullback-Leibler (cross-entropia pe scurt) ca fiind:*

i) Dacă $R = \mathbb{R}^d$:

$$\mathcal{D}(g,h) = \mathbb{E}_g\big[ln\frac{g(X)}{h(X)}\big] = \int_{\mathbb{R}^n} g(x)ln\ g(x)dx - \int_{\mathbb{R}^n} g(x)ln\ h(x)dx \tag{10}$$

ii) Dacă $R = \{x_1, \ldots, x_n, \ldots\}$

$$\mathcal{D}(g,h) = \sum g(x_i)\ln g(x_i) - \sum g(x_i)\ln h(x_i) \tag{11}$$

De observat că deși se poate privi ca o distanță (deoarece $\mathcal{D}(g,h) \geq 0$) termenul este impropriu folosit deoarece se vede ușor că $\mathcal{D}$ nu este simetrică, de fapt $\mathcal{D}(g,h) \neq \mathcal{D}(h,g)$ pentru orice caz în afară de $g = h$.

Concepte de Teoria Codurilor Pe lângă conceptele probabiliste, vom face referire la două concepte de bază în teoria codurilor necesare pentru partea practica a aplicației, pentru mai multe detalii în acest domeniu consultați [1] și [6]

Definiţia 2.10. *Fie A o mulțime finită numită alfabetul codului și având $q = card(A) = |A|$ elemente. O submulțime nevidă, $C \subset A^n$ se numește cod de lungime n peste alfabetul A. Elementele lui A^n se numesc cuvinte, elementele lui C se numesc cuvintele codului. Pentru $q = 2$ codul se numește binar, iar pentru $q = 3$ codul se numește ternar.*

Dacă $|C| = 1$ atunci C este trivial. În general, alfabetul A va fi identificat cu mulțimea de numere $\{0, 1, ..., q-1\}$ și va fi înzestrat cu structura de grup abelian a lui $(\mathbb{Z}/q\mathbb{Z}, +)$ sau cu structura de corp dacă q este o putere a unui număr prim. Deci un cod binar este un cod peste $\mathbb{F}_2$ iar un cod ternar este un cod peste $\mathbb{F}_3$.

Un cod $C \subset A^n$ peste un alfabet cu q simboluri și cu M cuvinte se spune că este cod de tip $(n, M)_q$, adică C este un cod de lungime n și mărime M . Vom adăuga în curând un al treilea parametru pentru a desemna tipul codului.

Definiţia 2.11. *Fie A un alfabet, i.e. o mulțime finită nevidă, și A^n mulțimea cuvintelor de lungime n. Distanța Hamming dintre două cuvinte x și y este dată de aplicația $d(x,\ y) = |\{i|x_i \neq y_i\}|,\ \forall\ x, y \in A^n$.*

Definim o aplicație de forma $d : A^n \times A^n \to \mathbb{R}$ dată prin

$$d(x,y) = \big|\{i|x_i \neq y_i\}\big| \tag{12}$$

ca fiind distanța Hamming dintre două cuvinte $x = (x_1, ..., x_n),\ y = (y_1, ..., y_n)$ *din alfabetul* A^n*. Cu alte cuvinte, distanța Hamming* $d(x,\ y)$ *dintre* x *și* y *este numărul de poziții unde* x *și* y *sunt diferite. De exemplu* $d(1101101, 0110110) = 5$*.*

Definiţia 2.12. *Definim cuvântul nul , cuvântul din* A^n *care are* 0 *pe toate pozițiile.*

Definiţia 2.13. *Definim distanța dintre un cuvânt* $x = (x_1, ..., x_n)$ *dintr-un cod* $C \subset A^n$ *de lungime* n *și cuvântul nul din cod ca fiind ponderea Hamming a lui x.*

2.2 Concepte Tehnice

Pentru realizarea aplicației de simulare, am folosit

- Limbajul Python
- Libraria NumPy

Python Este un limbaj de programare dinamic și versatil, care pune accentul pe lizibilitatea și simplitatea codului. Am ales Python pentru realizarea simulării, în detrimentul altor limbaje, datorită documentației bogate și a ușurinței de utilizare.

NumPy Conform [4], este o librărie de Python făcută special pentru cercetare computațională. Include obiecte de tip vector, derivate uni-dimensionale sau multi-dimensionale ale vectorilor (precum matricile) și o multitudine de funcții și metode eficiente pentru utilizarea obiectelor. Câteva dintre tipurile de funcții includ operații matematice, sortări, modificarea formei obiectelor sau funcții de baza în algebra liniară, statistică sau simularea variabilelor aleatoare.

3 Importance sampling

În această secțiune prezentăm sistematic metoda de importance sampling, incluzând trei abordări diferite ale metodei

3.1 Metoda propriu-zisă

În această secțiune vom explica ce este și cum funcționează procedeul de importance sampling. Informațiile din această secțiune se bazează în mod predominant pe [9] și [10], însă fiind o descriere generală a procedeului se pot găsi și în alte surse din bibliografie.

Importance samplingul este o metodă Monte Carlo care se bazează pe schimbarea distribuției inițiale cu una auxiliară pentru a facilita extragerea de informații despre un set de variabile aleatoare, de exemplu media. Acest artificiu se aplică în cazul în care varianța distribuției inițiale este mare, sau în cazul în care distribuția inițială este dificil de simulat în mod direct.

Acest procedeu este posibil datorită următoarei estimări:

Propoziţia 3.1. *Fie X cu densitate f și $X_1, ..., X_n$ un șir de variabile aleatoare independente și identic distribuite din g. Atunci*

$$\mathbb{E}_f[H(X)] = \lim_n \frac{1}{n} \sum_{i=1}^{n} H(X_i) \frac{f(X_i)}{g(X_i)} \tag{13}$$

Atunci g este o densitate auxiliară strict pozitivă. H este o funcție oarecare cu valori reale, iar E_f reprezintă media raportată la densitatea f.

Demonstraţie. Fie X o variabilă aleatoare. Vom nota cu E_f atunci când considerăm $X \sim f$, și cu E_g atunci când considerăm $X \sim g$. Fie $X_1, ..., X_n$ un șir de variabile aleatoare independente și identic distribuite cu densitate g. Atunci, conform legii numerelor mari avem ca

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} H(X_i) \frac{f(X_i)}{g(X_i)} = \mathbb{E}_g \left[H(X) \frac{f(X)}{g(X)} \right] \tag{14}$$

Din teorema 2.7 obținem

$$\int H(x) \frac{f(x)}{g(x)} g(x) dx = \int H(x) f(x) = \mathbb{E}_g \left[H(X) \right] \tag{15}$$

□

Demonstraţie. Fie X un vector de variabile aleatoare, f densitatea vectorului de variabile aleatoare

X și H o funcție cu valori reale. Notăm cu l media lui H(X) în raport cu densitatea f. Avem că

$$l = \mathbb{E}_f[H(X)] = \int H(x)f(x)dx \approx \frac{1}{n}\sum_{i=1}^{n} H(x_i)f(x_i) \tag{16}$$

Acum vom lua o altă densitate g cu proprietatea că g domină H·f. Această densitate se numește densitatea de importance sampling, cu ajutorul căreia îl putem scrie pe l astfel:

$$l = \int H(x)\frac{f(x)}{g(x)}g(x)dx = \mathbb{E}_g\left[H(X)\frac{f(X)}{g(X)}\right] \tag{17}$$

Obținem un estimator al lui l în funcție de g, anume

$$\hat{l} = \frac{1}{n}\sum_{i=1}^{n} H(X_i)\frac{f(X_i)}{g(X_i)} \approx l \tag{18}$$

□

În cele ce urmează vom nota cu

$$l = \mathbb{E}_f\left[H(X)\right] \tag{19}$$

și cu

$$\hat{l} = \hat{l}_n = \frac{1}{n}\sum_{i=1}^{n} H(X_i)\frac{f(X_i)}{g(X_i)} \tag{20}$$

unde $(X_i)_i$ sunt ca în propoziţia 3.1

Remarca 3.2. *$\hat{l}$ se numește estimatorul de importance sampling și este nedeplasat în raport cu densitatea g.*

Demonstraţie.

$$\begin{aligned}
\mathbb{E}_g[\hat{l}] &= \mathbb{E}_g\left[\tfrac{1}{n}\textstyle\sum_{i=1}^{n} H(X_i)\frac{f(X_i)}{g(X_i)}\right] \\
&= \tfrac{1}{n}\textstyle\sum_{i=1}^{n} \mathbb{E}_g\left[H(X_i)\frac{f(X_i)}{g(X_i)}\right] \\
&= \tfrac{1}{n} n \textstyle\int H(x)\frac{f(x)}{g(x)}g(x)dx \\
&= \textstyle\int H(x)f(x)dx \\
&= l
\end{aligned} \tag{21}$$

□

Remarca 3.3. *Raportul densităților W se numește raportul de verosimilitate.*

$$W(x) = \frac{f(x)}{g(x)}\ ,\ x \in \mathbb{R} \tag{22}$$

De aici provine o altă denumire a estimatorului de importance sampling, anume estimatorul raportului de verosimilitate

Remarca 3.4. *Pentru cazul particular în care densitățile coincid, $g = f$, avem că W(x)=1 iar estimatorul de verosimilitate se reduce la estimatorul Monte Carlo standard.*

Raporturile de verosimilitate nu trebuiesc știute cu exactitate, ci pot fi evaluate ca $W(X) = cw(X)$ pentru o funcție oarecare cunoscută $w(\cdot)$ și c o constantă. Din moment ce $\mathbb{E}_g\big[W(X)\big] = 1$, putem scrie $l = \mathbb{E}_g\big[H(X)W(X)\big]$ ca

$$l = \frac{\mathbb{E}_g\big[H(X)W(X)\big]}{\mathbb{E}_g\big[W(X)\big]} \tag{23}$$

Asta sugerează conform [10], ca alternativă la estimatorul raportului de verosimilitate , existența următorului estimator (*weighted sample estimator*)

$$\hat{l}_w = \frac{\sum_{k=1}^{n} H(X_k)w_k}{\sum_{k=1}^{n} w_k} \tag{24}$$

În acest caz, după cum se relatează în [9], fiecare $\{w_k\}$, cu $w_k = w(X_k)$, este interpretat ca o pondere pentru elementele $\{X_k\}$ din vector. Șirul $\{(X_k, w_k)\}$ definește un eșantion de variabile aleatoare din g(x), variabilele având o pondere proprie. Estimatorul de weighted sampling este ușor deplasat, însă deplasarea tinde la 0 cu cât n este mai mare. Într-un mod foarte general, putem considera weighted sample-ul $\{(X_k, w_k)\}$ ca pe o reprezentare a densității f(x), în sensul în care pentru orice $H(\cdot)$ avem:

$$l = \mathbb{E}_f\big[H(X)\big] \approx \hat{l}_w \tag{25}$$

Propoziţia 3.5. *Conform [2], estimatorul de importance sampling are următoarea formulă de variantă:*

$$Var(\hat{l}) = \frac{1}{n}\left(\int \frac{H^2(x)f^2(x)}{g(x)}dx - l^2\right) \tag{26}$$

Demonstraţie.

$$\begin{aligned} Var(\hat{l}) &= \tfrac{1}{n} Var(H(X_i)\cdot \tfrac{f(X_i)}{g(X_i)}) \\ &= \tfrac{1}{n}\Big\{ E_g\big[H^2(X_i)\cdot \tfrac{f^2(X_i)}{g^2(X_i)}\big] - E_g\big[H(X_i)\cdot \tfrac{f(X_i)}{g(X_i)}\big]^2 \Big\} \\ &= \tfrac{1}{n}\Big\{ E_g\big[H^2(X_i)\cdot \tfrac{f^2(X_i)}{g^2(X_i)}\big] - l^2 \Big\} \\ &= \tfrac{1}{n}\big(\int \tfrac{H^2(x)f^2(x)}{g(x)}dx - l^2\big) \end{aligned} \tag{27}$$

□

După cum se poate observa din propoziţia 3.5, varianța estimatorului $\hat{l}$ este strâns legată de alegerea densității de probabilitate g. Astfel, dorim să găsim o densitate g^* pentru care varianța este minimă, densitate numită și densitatea optimă de importance sampling.

Propoziţia 3.6. *Densitatea g^* pentru care varianța estimatorului $\hat{l}$ este minimă este dată de:*

$$g^*(x) = \frac{|H(x)|f(x)}{\int |H(x)|f(x)dx},\ x \in \mathbb{R} \tag{28}$$

Demonstraţie. Notăm soluția

$$\min_g Var_g\big(H(X)\frac{f(x)}{g(x)}\big) \tag{29}$$

cu g^* și vrem să demonstrăm

$$\min_g Var_g\big(H(X)\frac{f(x)}{g(x)}\big) \Rightarrow g^*(x) = \frac{|H(x)|f(x)}{\int |H(x)|f(x)dx} \tag{30}$$

Avem că:

$$Var_g\big(H(X)\frac{f(x)}{g(x)}\big) \geq Var_{g^*}\big(H(X)\frac{f(x)}{g^*(x)}\big) \tag{31}$$

pentru o funcție $g(\cdot)$ oarecare, și știm că

$$\begin{aligned} Var_g\big(H(X)\tfrac{f(X)}{g(X)}\big) &= \mathbb{E}_g\Big[\big(H(X)\tfrac{f(x)}{g(x)}\big)^2\Big] - \mathbb{E}_g\Big[H(X)\tfrac{f(x)}{g(x)}\Big]^2 \\ &= \int \tfrac{H^2(x)f^2(x)}{g^2(x)}g(x)dx - l^2 \end{aligned} \tag{32}$$

Înlocuind în (32) pe $g(x)$ cu $g^*(x)$, pe care îl presupunem ca fiind $\frac{|H(x)|f(x)}{\int |H(x)|f(x)}$, obținem:

$$\begin{aligned} Var_{g^*}\big(H(X)\tfrac{f(X)}{g^*(X)}\big) &= \int \tfrac{H^2(x)f^2(x)}{g^{*2}(x)}g^*(x)dx - l^2 \\ &= \int H^2(x)f^2(x)(\tfrac{(|H(x)|f(x))^2}{(\int |H(x)|f(x)dx)^2})^{-1}g^*(x)dx - l^2 \\ &= \int H^2(x)f^2(x)\tfrac{(\int |H(x)|f(x)dx)^2}{H^2(x)f^2(x)}g^*(x)dx - l^2 \\ &= \big(\int H(x)f(x)\big)^2 dx \int g^*(x)dx - l^2 \\ &= \big(\int H(x)f(x)\big)^2 dx \cdot 1 - l^2 \\ &= \big(\int H(x)f(x)\big)^2 dx - l^2 \end{aligned} \tag{33}$$

Revenind la inecuația (31)

$$\begin{aligned} \int \tfrac{H^2(x)f^2(x)}{g^2(x)}g(x)dx - l^2 &\geq (\int H(x)f(x)dx)^2 - l^2 \\ \int \tfrac{H^2(x)f^2(x)}{g^2(x)}g(x)dx &\geq (\int H(x)f(x)dx)^2 \end{aligned} \tag{34}$$

Este, așadar, suficient să demonstrăm următoarea inegalitate:

$$\big(\int H(x)f(x)dx\big)^2 \leq \int \frac{H^2(x)f^2(x)}{g^2(x)}g(x)dx \tag{35}$$

De aici putem aplica inegalitatea Cauchy–Bunyakovsky–Schwarz pentru funcții integrabile, folosindu-ne de faptul că $g(x)dx = \mu$ este o probabilitate și privind cele două componente ale inegalității ca două integrale Lebesgue de forma:

$$\big(\int \Phi d\mu\big)^2 \leq \big(\int \Phi^2 d\mu\big) \tag{36}$$

unde $\Phi = \frac{H(x)f(x)}{g(x)}$
Prin urmare, obținem că afirmația (34) este adevărată și deci (35) este adevărată. □

Remarca 3.7. *În cazul particular în care $H(x) \geq 0$ obținem că:*

$$g^*(x) = \frac{H(x)f(x)}{l},\ x \in \mathbb{R} \tag{37}$$

Iar varianța estimatorului $\hat{l}$ în acest caz este egală cu zero:

$$\begin{aligned} Var_{g^*} &= \tfrac{1}{n}\Big\{E_{g^*}\big[H^2(X_i)\cdot\tfrac{f^2(X_i)}{g^{*2}(X_i)}\big] - l^2\Big\} \\ &= \tfrac{1}{n}\Big[\big(\textstyle\int (H(x)f(x)\big)^2 - l^2)\Big] \\ &= \tfrac{1}{n}(l^2 - l^2) \\ &= 0 \end{aligned} \tag{38}$$

Exemplul 3.8. *((Exemplu extras din [9] (Cap. 5.6.2, exem. 5.8, pp. 133))*
Fie $X \sim Exp(u^{-1})$ și $H(X) = I_{\{X\geq\gamma\}}$ pentru un γ pozitiv, și fie f densitatea probabilității X. Avem următoarea estimare

$$l = \mathbb{E}_f\big[H(X)\big] = \int_\gamma^\infty u^{-1}e^{-xu^{-1}}dx = e^{-\gamma u^{-1}} \tag{39}$$

Avem, conform (3.7)

$$g^*(x) = H(x)f(x)l^{-1} = I_{\{X\geq\gamma\}}u^{-1}e^{-xu^{-1}}e^{\gamma u^{-1}} = I_{\{X\geq\gamma\}}u^{-1}e^{-(x-\gamma)u^{-1}} \tag{40}$$

Astfel, distribuția optimă pentru importance sampling a lui X este o distribuție exponențială șiftată. De remarcat că, după cum am ales și în (3.1) *, $H \cdot f$ este dominat de g^*, dar funcția f în sine nu este dominată de g^*. Cum g^* e optim, obținem că estimatorul $\hat{l}$ este constant. Mai specific, pentru n=1 obținem:*

$$\hat{l} = H(X)W(X) = \frac{H(X)f(X)}{H(X)f(x)l^{-1}} = l \tag{41}$$

Remarca 3.9. *Este important de menționat că, după cum se poate citi în [10],[9] și [5], deși estimatorul obținut la* (3.2) *este nedeplasat pentru orice densitate g care domină $H \cdot f$ nu toate densitățile sunt propice estimării. Unul dintre factorii cruciali în alegerea densităților pentru importance sampling este că estimatorul $\hat{l}$ să aibă varianța finită. Această afirmație este echivalentă cu:*

$$\mathbb{E}_g\Big[H^2(X)\frac{f^2(X)}{g^2(X)}\Big] = \mathbb{E}_f\Big[H^2(X)\frac{f(X)}{g(X)}\Big] < \infty \tag{42}$$

În concluzie, g nu ar trebui să aibă "coada mai subțire" ca f și că ar fi de preferat ca raportul de verosimilitate ,$\frac{f}{g}$, sa fie mărginit.

Remarca 3.10. *(Degenerarea estimatorului de importance sampling)*
Conform [9], estimatorul $\hat{l}$ obținut la 3.1 poate suferi de o formă de degenerare, deoarece distribuția raportului de verosimilitate W(X) sub densitatea de importance sampling g se poate distorsiona în funcție de "dimensiunea" lui n. Pe scurt, W(X) poate lua valori aproape de 0 cu probabilități mari, dar poate lua de asemenea valori foarte mari cu o probabilitate mică dar semnificativă. Consecința evidentă este creșterea varianței distribuției lui W(X) pentru un n foarte mare.

Exemplul 3.11. *(Exemplu bazat pe [9] (Cap. 5.6.2, rem. 5.6.1, pp. 133-134) și pe [5])*
Pentru a exemplifica remarca de mai sus, vom presupune că variabilele $X_1, ..., X_n$ sunt independente și identic distribuite atât sub densitatea f cât și sub densitatea g. Prin urmare, dacă avem $X := (X_1, ..., X_n)$ un vector de variabile aleatoare, atunci $X \sim f$ cu $f(x) = f_1(x_1) = ... = f_n(x_n)$ și respectiv $X \sim g$ cu $g(x) = g_1(x_1) = ... = g_n(x_n)$. Presupunând că pentru fiecare X_i, densitățile sunt f_1, respectiv g_1 îl putem scrie pe W(X) ca

$$\begin{aligned} W(X) &= \textstyle\prod_1^n \frac{f_1(X_i)}{g_1(X_i)} \\ &= e^{\prod_{i=1}^n \frac{f_1(X_i)}{g_1(X_i)}} \\ &= e^{\sum_{i=1}^n ln\frac{f_1(X_i)}{g_1(X_i)}} \\ &= Exp(\textstyle\sum_{i=1}^n ln\frac{f_1(X_i)}{g_1(X_i)}) \end{aligned} \tag{43}$$

Dacă folosim legea numerelor mari pentru a aproxima $\sum_{i=1}^n ln\frac{f_1(X_i)}{g_1(X_i)}$ cu $n\mathbb{E}_{g_1}[ln\frac{f_1(X_i)}{g_1(X_i)}]$, atunci:

$$W(X) \approx Exp\Big\{ - n\mathbb{E}_{g_1}\Big[ln\big(\frac{g_1(X_i)}{f_1(X_i)}\big)\Big]\Big\} \tag{44}$$

Putem observa ușor cu ajutorul inegalității Jensen că

$$\mathbb{E}_{g_1}\Big[ln\big(\frac{g_1(X_i)}{f_1(X_i)}\big)\Big] = -\mathbb{E}_{g_1}\Big[ln\big(\frac{f_1(X_i)}{g_1(X_i)}\big)\Big] \geq -ln\mathbb{E}_{g_1}\Big[\frac{g_1(X_i)}{f_1(X_i)}\Big] = -ln1 = 0 \tag{45}$$

Prin urmare obținem că:

$$\lim_{n\to\infty} W(X) = 0 \tag{46}$$

Pe de alta parte $\mathbb{E}_g[W(X)] = 1$. *De aici se poate observa degenerarea distribuției lui W(X), cauzată de "mărimea" lui n.*

Din cauza acestei degenerări ne dorim ca g^* să aibă o formă similară cu f.

În fine, având în vedere aceste criterii de alegere a densității optime, merită menționat că pentru a o determina folosind formula (3.7) este necesar să știm l, care este chiar valoarea pe care vrem să o estimăm. Deși se poate utiliza o simulare "pilot" pentru a determina g^* sau o densitate cât mai apropiată, metoda nu este întotdeauna eficientă. Ca soluție pentru acest impas, există metode precum minimizarea directă a variației sau metoda cross-entropiei, care au ca rol estimarea unei densități cât mai apropiate de cea optima și care fac obiectul secțiunilor următoare.

3.2 Metoda Minimizării Variației

Această secțiune se bazează integral pe [9] și [10].

Remarca 3.12. *Atunci când f aparține unei familii de distribuții parametrice, este convenabil ca și distribuția de "importance sampling" să aparțină aceleiași familii de distribuții.*

Presupunem că $f(\cdot) = f(\cdot; u)$ aparține familiei de distribuții

$$\mathscr{F} = \{f(\cdot; v), v \in \mathscr{V}\}. \tag{47}$$

Atunci găsirea unei distribuții optime se reduce la rezolvarea unei probleme de minimizare parametrică

$$\min_{v\in\mathscr{V}} Var_v\big(H(X), W(X; u, v)\big) \tag{48}$$

unde W(X;u,v) = $\frac{W(X;u)}{W(X;v)}$. Ne vom referi la v ca și vectorul parametric de referință .

Pe distribuția $f(\cdot, v)$ media $l = \mathbb{E}_v\big[H(X)W(X; u, v)\big]$ este constantă, prin urmare soluția problemei de mai sus este:

$$\min_{v\in\mathscr{V}} V(v) \tag{49}$$

unde

$$V(v) = \mathbb{E}_v\big[H^2(X)W^2(X; u, v)\big] = \mathbb{E}_u\big[H^2(X)W(X; u, v)\big] \tag{50}$$

Problemele echivalente de la (48) și (49) se categorizează ca problema minimizării variației, iar vectorul parametric v^* se va numi în acest context vectorul parametric optim de referința pentru minimizarea variației. Vectorul u este numit vectorul nominal.

O variantă probabilist aleatoare a (48) și (49) este

$$\min_{v\in\mathscr{V}} \widehat{V}(v) \tag{51}$$

unde

$$\widehat{V}(v) = \frac{1}{n}\sum_{k=1}^{n}\big[H^2(X_k)W(X_k;u,v)\big] \tag{52}$$

iar eșantionul $X_1, ..., X_n$ este din f(x;u). De observat că funcția $\widehat{V}(v)$ devine determinstă atunci când eșantionul este cunoscut.

Deoarece, în general, atât $V(v)$ cât și $\widehat{V}(v)$ sunt convexe și diferențiabile, iar media și diferențierea sunt interschimbabile, soluțiile pentru (48) - (49) și (51) - (52) se pot obține prin rezolvarea în funcție de v a unui sistem format din următoarele ecuații:

$$\mathbb{E}_u\big[H^2(X)\nabla W(X;u,v)\big] = 0 \tag{53}$$

respectiv

$$\frac{1}{n}\sum_{k=1}^{n}\big[H(X_k)\nabla W(X_k;u,v)\big] = 0 \tag{54}$$

unde

$$\nabla W(X;u,v) = \nabla\frac{f(X;u)}{f(X,v)} = [\nabla lnf(X;v)]W(X;u,v) \tag{55}$$

Exemplul 3.13. *(Exemplu bazat pe [9] (Cap. 5.6.2, exem. 5.9, pp. 135))*
Îl vom estima pe $l = \mathbb{E}[X]$, *unde* $X \sim Exp(u^{-1})$. *Dacă alegem* $f(x;v) = v^{-1}e^{-xv^{-1}}$ *ca și densitatea de importance sampling, cu* $x \geq 0$, *și* $H(\cdot)$ *ca fiind funcția identitate,* (49) *se reduce la:*

$$\min_v V(v) = \min_v \int_0^\infty x^2\frac{f(x;u)^2}{f(x;v)}dx = \min_v \frac{v}{u^2}\int_0^\infty x^2 e^{-(2u^{-1}-v^{-1})x}dx \tag{56}$$

Calculând primitiva

$$\begin{aligned}
\int x^2e^{-(2u^{-1}-v^{-1})x}dx &= \tfrac{x^2e^{x(1/v-2/u)}}{1/v-2/u} - \tfrac{2}{1/v-2/u}\int xe^{-(2u^{-1}-v^{-1})x} \\
&= \tfrac{x^2e^{x(1/v-2/u)}}{1/v-2/u} - \tfrac{2xe^{x(1/v-2/u)}}{(1/v-2/u)^2} + \tfrac{2}{(1/v-2/u)^2}\int e^{-(2u^{-1}-v^{-1})x} \\
&= \tfrac{x^2e^{x(1/v-2/u)}}{1/v-2/u} - \tfrac{2xe^{x(1/v-2/u)}}{(1/v-2/u)^2} + \tfrac{2e^{x(1/v-2/u)}}{(1/v-2/u)^3} \\
&= e^{x(1/v-2/u)}\big(\tfrac{x^2}{1/v-2/u} - \tfrac{2x}{(1/v-2/u)^2} + \tfrac{2}{(1/v-2/u)^3}\big) \\
&= e^{x(1/v-2/u)}\big(\tfrac{uvx^2}{u-2v} - \tfrac{2u^2v^2x}{(u-2v)^2} + \tfrac{2u^3v^3}{(u-2v)^3}\big)
\end{aligned} \tag{57}$$

se poate observa ușor ca se ajunge la următorul rezultat pentru $v \geq u/2$*:*

$$\min_v V(v) = \min_{v \geq u/2} \frac{2uv^4}{(2v-u)^3} \tag{58}$$

Pentru a obține minimul, derivăm rezultatul obținut după v și egalăm cu 0

$$\frac{\partial}{\partial v} \frac{2uv^4}{(2v-u)^3} = \frac{4uv^3(v-2u)}{(2v-u)^3} = 0 \tag{59}$$

Evident, soluția ecuației este $v = 2u$*, astfel* $v^* = 2u$*. Folosind variantele stocastice* (51) *și* (52)*, avem*

$$\min_v V_n(v) = \frac{v}{u^2} \frac{1}{n} \sum_{k=1}^{n} X_n^2 e^{-(2u^{-1}-v^{-1})X_n} \tag{60}$$

cu soluția $\hat{v^*}$*.*
Cunoaștem că

$$V_n(v) \xrightarrow[n \to \infty]{a.s.} V(v) \tag{61}$$

deci putem afirma că $V_n(v) \approx V(n)$*, concluzia fiind că* $\hat{v^*} \approx v^*$

Exemplul 3.14. *(Exemplu extras din [9] (Cap. 5.6.2, exem. 5.10, pp. 135))*
Considerăm, ca în (3.8)*, că* $l = \mathbb{P}(X \geq \gamma) = exp(-\gamma u^{-1})$*. În acest caz, folosind familia de distribuții* $\{f(x;v);\ v > 0\}$ *definite de* $f(x;v) = v^{-1}exp(-xv^{-1})$*, unde* $x \geq 0$*,* (49) *se reduce la*

$$\min_v V(v) = \min_v \frac{v}{u^2} \int_\gamma^\infty e^{-(2u^{-1}-v^{-1})x} dx = \min_{v \geq u/2} \frac{v^2}{u} \frac{e^{-\gamma(2u^{-1}-v^{-1})}}{(2v-u)} \tag{62}$$

Vrem deci să aflăm v pentru cazul

$$-\frac{e^{(-2/u+1/v)\gamma}(-2v(v-\gamma)+u(2v-\gamma))}{(u(u-2v)^2)} = 0 \tag{63}$$

vrem, deci

$$\begin{aligned} &-2v(v-\gamma)+u(2v-\gamma) = 0 \\ &2v^2-2v\gamma-2uv+u\gamma = 0 \\ &4v^4-4v\gamma-4uv+2u\gamma = 0 \\ &4v^4-4v\gamma-4uv+2u\gamma+u^2+\gamma^2 = u^2+\gamma^2 \\ &(2v-u-\gamma)^2 = u^2+\gamma^2 \end{aligned} \tag{64}$$

Reiese trivial mai departe că

$$v^* = \frac{1}{2}(\gamma + y + \sqrt{u^2+\gamma^2}) \tag{65}$$

Dându-l pe γ factor comun sub radical și folosind aproximarea $\sqrt{1+(u/\gamma)^2} = 1+\mathcal{O}((u/\gamma)^2)$ obținem

$$v^* = \gamma + \frac{u}{2} + \mathcal{O}((u/\gamma)^2) \tag{66}$$

unde $\mathcal{O}(x)$ este o funcție de x care îndeplinește condiția

$$\lim_{n\to 0} \frac{\mathcal{O}(x^2)}{x^2} = constant \tag{67}$$

Remarca 3.15. *Cu ajutorul unui parametru arbitrar de referință w, putem scrie* (49) *ca și*

$$\min_{v\in\mathscr{V}} V(v) = \min_{v\in\mathscr{V}} \mathbb{E}_w\big[H^2(X)W(X;u,v)W(X;u;w)\big] \tag{68}$$

prin înmulțirea și împărțirea cu o densitate de test f(X;w) a părții integrate. Mai departe, putem înlocui media din forma de mai sus cu varianta stocastică (probabilist aleatoare). Mai precis, varianta stocastică este:

$$\min_{v\in\mathscr{V}} V(v) = \min_{v\in\mathscr{V}} \frac{1}{n}\sum_{k=1}^{n}\big[H^2(X_k)W(X_k;u,v)W(X_k;u,w)\big] \tag{69}$$

unde $X_1, ..., X_n$ este un vector de variabile independente și identic distribuite din $f(\cdot;w)$ iar w este un parametru de test ales corespunzător. Rezolvarea acestei ecuații stocastice produce un estimator $\hat{v^*}$ al vectorului parametric optim. Procedeul poate fi repetat folosind $\hat{v^*}$ ca și parametru de testare pentru obținerea unui estimator mai precis.

După determinarea estimatorului, l este estimat cu ajutorul estimatorului pentru raportul de verosimilitate

$$\hat{l} = \frac{1}{N}\sum_{k=1}^{N} H(X_k)W(X_k;u,v) \tag{70}$$

unde $X_1, ..., X_n$ este un eșantion aleator din $f(\cdot;v)$,iar N-ul folosit în (70) *este, în mod normal, mai mare decât cel din* (69). (70) *se numește estimatorul standard al raportului de verosimilitate.*

3.3 Metoda Cross-Entropiei

Această secțiune se bazează pe informațiile din [9], [10] și [3]. Din punct de vedere al autorului, referința [3] este o alegere foarte bună pentru a aprofunda tema acestei secțiuni.

O altă metodă pentru alegerea densității optime de importance sampling se bazează pe cross-entropia Kullback-Leibler, pe scurt cross-entropia. Reamintim că aceasta reprezintă "distanța" dintre două densități g și h și se notează $\mathcal{D}(g,h)$ (impropriu spus distanță, deoarece $\mathcal{D}(g,h)$ diferit

de $\mathcal{D}(h, g)$). Fiind o "distanță", $\mathcal{D}(g, h) \geq 0$, cu egalitatea valabilă doar în cazul g=h. Ideea de bază a acestei metode este alegerea unei densități de importance sampling, în acest caz h, astfel încât cross-entropia dintre h și densitatea optimă g^*(28) să fie minimă. Practic, dorim o densitate h cât mai apropiată ca formă de g^* pe care o vom numi densitatea optimă cross-entropică și care va servi drept soluție pentru următoarea problemă de optimizare funcțională.

$$\min_h \mathcal{D}(g^*, h) \tag{71}$$

Propoziţia 3.16. *(Conform [9])*
Avem că densitatea optimă cross-entropică h, soluția pentru programul (71)*, este una și aceeași cu densitatea optimă pentru minimizarea variației* g^*

Demonstraţie. Vrem h astfel încât

$$\min_h \mathcal{D}(g^*, h) \tag{72}$$

Dar știm că $\mathcal{D}(g, h) \geq 0$, deci problema se reduce la

$$\begin{aligned} &\mathcal{D}(g^*, h) = 0 \\ &\textstyle\int g^*(x) ln\ g^*(x) - \int g^*(x) ln\ h(x) = 0 \\ &\textstyle\int g^*(x) ln\ g^*(x) = \int g^*(x) ln\ h(x) \end{aligned} \tag{73}$$

Reiese trivial ca $g^*(x) = h(x)$ □

După cum am procedat și la metoda minimizării variației (3.12), ne vom restrânge la o familie de densități parametrice $\{f(\cdot, v); v \in \mathscr{V}\}$ care conține și densitatea "nominală" $f(\cdot; u)$. Rămâne de rezolvat următoarea problemă de optimizare parametrică:

$$\min_v \mathcal{D}(g^*; f(\cdot; v)) \tag{74}$$

Propoziţia 3.17. *(Conform [3], [9])*
Din moment ce prima integrală din formula cross-entropiei nu depinde de v, minimizarea distanței dintre g^* *și* $f(\cdot; v)$ *este echivalentă cu maximizarea următoarei expresii în funcție de v*

$$D(v) = \int H(x) f(x; u) ln\ f(x; v)\ dx = \mathbb{E}_u\big[H(X) ln\ f(X; v)\big] \tag{75}$$

unde presupunem ca funcția H(x) nu este negativă.

Demonstraţie. Avem următoarea expresie

$$\min_v \mathcal{D}(g^*(X); f(X; v)) = \min_v \int g^*(X)\ ln\ g^* dx - \int g^*(X)\ ln\ f(X; v) dx \tag{76}$$

Cum primul termen al formulei de mai sus $\int g^*(X)\ ln\ g^* dx$ nu depinde de v, expresia devine

$$\min_v - \int g^*(X)\ ln\ f(X;v)dx = \max_v \int g^*(X)\ ln\ f(X;v)dx \tag{77}$$

Reamintim de formula pentru densitatea optimă de importance sampling pentru H(X) non-negativ (3.7), astfel în cazul de față avem $g^*(X) = H(X)f(X;u)l^{-1}$ și deci expresia devine

$$\max_v \int H(X)f(X;u)l^{-1}\ ln\ f(X;v)dx \tag{78}$$

Cum ne interesează doar punctul în care funcția are valoare minimă și nu valoarea minimă în sine, putem ignora constanta l și rămânem cu:

$$\max_v \int H(X)f(X;u)\ ln\ f(X;v)dx = \max_v \mathbb{E}_u\big[H(X)ln\ f(X;v)\big] = \max_v D(v) \tag{79}$$

Soluția problemei este, în mod evident, soluția pentru (74) □

Astfel, vectorul v^*, care în acest context se numește vectorul parametric optim de referință cross-entropic, este soluția pentru (79) (prin urmare pentru 74). Deoarece $D(v)$ este convexă și diferențiabilă, v^* se poate obține prin rezolvarea următoarei ecuații:

$$\mathbb{E}_u\big[H(X)\nabla ln\ f(X;v)\big] = 0 \tag{80}$$

Alternativa stocastică a (80) este următoarea, fiind similară cu cea de la minimizarea variației:

$$\frac{1}{n}\sum_{k=1}^{n} H(X_k)\nabla ln\ f(X_k;v) = 0 \tag{81}$$

În mod analog cu problema minimizării variației (48), numim (79) "problema cross-entropiei" iar vectorul soluție v^* se numește vectorul parametric optim de referința cross-entropic.

Remarca 3.18. *În mod similar cu* (3.15)*, putem scrie* (75) *folosind un parametru arbitrar* w *ca și:*

$$\max_v D(v) = \max_v \mathbb{E}_w\big[H(X)W(X;u,w)\ ln\ f(X;v)\big] \tag{82}$$

De asemenea, identic cu (69)*, îl putem estima pe* v^* *ca soluție la următoarea problemă*

$$\max_v \hat{D}(v) = \max_v \frac{1}{n}\sum_{k=1}^{n} H(X_k)W(X_k;u,w)\ ln\ f(X_k;v) \tag{83}$$

unde $X_1, ..., X_n$ reprezintă un eșantion aleator din $f(\cdot, w)$. Analog cu (69)*, procedeul se poate repeta pentru obținerea unui estimator mai bun pentru v^*.*

Soluția pentru (83)*, dat fiind că $\hat{D}(v)$ este și convexă și diferențiabilă în funcție de v, se poate obține rezolvând:*

$$\frac{1}{n}\sum_{k=1}^{n} H(X_k)W(X_k; u, w)\nabla ln\ f(X_k; v) = 0 \tag{84}$$

unde gradientul este în funcție de v.

Conform [9] pentru eșantioane de dimensiuni normale ($50 \leq n$) soluțiile obținute prin metoda cross-entropiei cu un parametru arbitrar (3.15) sunt similare cu cele obținute prin metoda minimizării de varianță cu un parametru arbitrar (3.18). Pentru eșantioane de dimensiuni mai mari însă, estimatorul de importance sampling obținut prin metoda cross-entropiei este inferior celui obținut prin minimizarea variației, având atât deplasarea cât și varianța mai mari. Acest fapt se datorează degenerării lui W, care este mai accentuată la metoda cross-entropiei.

Acestea fiind spuse, conform sursei [3], avantajul metodei cross-entropiei este că soluția se poate deduce în mod analitic. În particular, se întâmplă când distribuția vectorului de variabile aleatoare X face parte din familia distribuțiilor exponențiale. De exemplu, pentru o exponențială uni-dimensională parametrizată după medie, soluția optimă este mereu

$$v^* = \frac{\mathbb{E}_u\big[H(X)X\big]}{\mathbb{E}_u\big[H(X)\big]} = \frac{\mathbb{E}_w\big[W(X; u, w)H(X)X\big]}{\mathbb{E}_w\big[W(X; u, w)H(X)\big]} \tag{85}$$

iar corespondentul raportat la eșantion este

$$\hat{v} = \frac{\sum_{k=1}^{N} W(X_k; u, w)H(X_k)X_k}{\sum_{k=1}^{N} W(X_k; u, w)H(X_k)} \tag{86}$$

cu $X_1, ..., X_N$ un eșantion arbitrar din $f(\cdot; w)$ și w un parametru arbitrar. Există și o variantă pentru distribuții multi-dimensionale a (86), aceasta fiind

$$\hat{v}_i = \frac{\sum_{k=1}^{N} W(X_k; u, w)H(X_k)X_{k_i}}{\sum_{k=1}^{N} W(X_k; u, w)H(X_k)} \tag{87}$$

cu $i = 1, ..., n$ unde X_{k_i} este al i-lea element al vectorului X_k iar u și w sunt vectori parametrici.

Propoziţia 3.19. *Pentru cazul particular u=w,* (87) *se reduce la*

$$\hat{v}_i = \frac{\sum_{k=1}^{N} H(X_k)X_{k_i}}{\sum_{k=1}^{N} H(X_k)} \tag{88}$$

unde $X_k \sim f(x; u)$

Exemplul 3.20. *Bazat pe [9](Cap. 5.6.3, exem. 5.11, pp. 138)*
Fie ca în (3.13)*,* $l = \mathbb{E}[X]$ *cu* $X \sim Exp(u^{-1})$ *si* $f(x; v) = v^{-1}e^{-xv^{-1}}$*. Pentru a rezolva folosind metoda cross-entropiei, înlocuim în* (80) *și obținem:*

$$E_u[H(X)\nabla ln\ f(X; v)] = \frac{1}{uv^2}\int_0^\infty xe^{-xu^{-1}}(x - v) = 0 \tag{89}$$

Rezolvând integrala obținem:

$$\int_0^\infty xe^{-xu^{-1}}(x - v) = -e^{-x/u}u(2u^2 - uv + 2ux - vx + x^2)\Big|_0^\infty = u^2(v - 2u) \tag{90}$$

Deci soluția problemei se rezumă la:

$$v - 2u = 0 \tag{91}$$

În mod evident, $v^* = 2u$*, aceeași soluție cu cea obținută prin metoda minimizării de varianță la* (3.13)*. În mod analog cu* (3.13)*, rezolvând varianța stocastică* (81) *obținem un rezultat* $\hat{v^*}$ *care converge către parametrul optim* $v^* = 2u$

4 Aplicaţie la probleme de numărare

În această secțiune vom analiza un puzzle Sudoku într-un mod matematic și vom încerca sa răspundem la întrebarea ”Câte Sudoku unice există?”. Această secțiune a fost realizată urmărind materialele [7] și [8].

4.1 Sudoku: definiții, interpretare matematică

Definiţia 4.1. *Sudoku este un tip de puzzle în care jucătorul trebuie sa completeze o matrice de* 9×9 *,împărțită la rândul ei în 9 matrici de* 3×3*, cu cifre de la 1 la 9 în așa manieră încât cifrele să nu se repete pe linii, coloane sau în interiorul matricilor de* 3×3*. Ne vom referi de aici înainte la aceste matrici de* 3×3 *ca și ”pătrate”, și la cea mai mică unitate a matricei ne vom referi ca fiind o ”celulă”.Notăm cu* b_{ij} *celula corespunzătoare indicilor matricei i și j.*

Definim liniile, coloanele și pătratele ca fiind elemente componente ale unui puzzle Sudoku, iar N mulțimea tuturor elementelor componente unui puzzle Sudoku; observăm ca $|N| = 27$*.*

Stabilim notația C = $\{1,2,3,4,5,6,7,8,9\}$.

Definiţia 4.2. *O matrice de* 9×9 *care respectă regulile de completare pe linie, coloană sau în interiorul pătratelor, adică conține o permutare a mulțimii C pe fiecare element în parte, se numește soluție Sudoku.*

Pentru a verifica dacă o matrice este soluție Sudoku, modalitatea evidentă este parcurgerea fiecărui element pentru verificarea condiției, însă este o modalitate ineficientă din punct de vedere computațional. Ne punem deci problema unui mod eficient de a verifica validitatea unei soluții.

La prima vedere, o metodă de validare este sumarea termenilor fiecărui element component în parte, rezultatul dorit fiind $\sum_{i=1}^{9} n_i = 45$, unde n_i este cifra din celula i a componentei. Totuși, această sumă nu asigură corectitudinea soluției deoarece rezultatul obținut prin simpla adunare a cifrelor dintr-o componentă nu este determinat în mod unic, ușor de văzut pentru $\sum_{i=1}^{9} n_i = 45$ cu $n = (5, 5, 5, 5, 5, 5, 5, 5, 5)$

O metodă de validare mai puțin evidentă este sumarea termenilor fiecărui element component în parte sub forma $\sum_{i=1}^{9} 2^{n_i - 1}$. Se poate calcula ușor ca suma unui element component rezolvat corect este $\sum_{i=1}^{9} 2^{n_i - 1} = 511$.

Ce conferă unicitatea lui 511 ca rezultat în cazul dat? Răspunsul este reprezentarea în cod binar a acestuia, vectorul (111111111), care are ponderea Hamming $w\big((111111111)\big) = 9$; vezi 2.13. Se verifică ușor că ponderea Hamming a unei sume de puteri a lui 2 poate fi maxim numărul de termeni ai sumei, și strict mai mică decât numărul de termeni în cazul elementelor duplicate. De aici putem extrage o nouă definiție pentru o soluție Sudoku:

Definiţia 4.3. *O matrice de* 9×9 *se numește soluție Sudoku dacă*

$$\sum_{n=1}^{9} 2^{n_i - 1} = 511, \ \forall n \in N \text{adică pentru orice componentă Sudoku}$$

4.2 Abordarea problemei de numărare în termeni probabiliști și algoritmici

Pentru a găsi numărul de soluții Sudoku existente prin intermediul metodelor de simulare Monte Carlo, avem nevoie de o abordare care să respecte următoarele constrângeri probabiliste:

1. Vrem un spațiu total al soluțiilor $\mathbf{E}$ finit al cărui cardinal să îl putem calcula ușor.
2. Dorim de asemenea ca $S \subset \mathbf{E}$, unde S este mulțimea tuturor soluțiilor Sudoku unice
3. Pentru a putea genera în mod aleator elemente $\omega \in \mathbf{E}$ vrem ca ω să fie ușor de generat din distribuția uniformă pe $\mathbf{E}$
4. Trebuie să putem verifica în mod eficient dacă $x \in S$ pentru orice $x \in \mathbf{E}$

Având aceste cerințe, este ușor de văzut cum putem traduce problema de numărare în termeni probabiliști. Vom alege $\mathbf{E}$ ca fiind mulțimea matricilor de 9×9 formate prin umplerea liniilor cu permutări ale mulțimii C. Cardinalul mulțimii este ușor de calculat $|\mathbf{E}| = (9!)^9 \approx 1.091 \cdot 10^{50}$ și avem certitudinea că $S \subset \mathbf{E}$.

Remarca 4.4. *De observat că* $|\mathbf{E}|$ *este un număr uriaș, iar o numărare naivă a soluțiilor Sudoku prin parcurgerea întregii mulțimi* $\mathbf{E}$ *nu este fezabilă.*

Înzestrăm Ω cu σ-algebra părților, $\mathcal{B}$. Am definit spațiul măsurabil $(\mathbf{E}, \mathcal{B})$, rămâne să alegem

$$\mathbb{P}(X = x) = \mathbb{P}_u(X = x) = u(x) = \frac{1}{|\mathbf{E}|}, \ \forall x \in \mathbf{E} \tag{92}$$

, ca fiind probabilitatea pe $\mathbf{E}$ dată de distribuția uniformă pentru o variabilă aleatoare $X : \Omega \to \mathbf{E}$. În particular, pentru acest experiment, putem alege $\Omega = \mathbf{E}$ definim $X : \mathbf{E} \to \mathbf{E}$ identitatea pe spațiul măsurabil. Partea de verificare eficientă este acoperită de definiţia 4.3.

Avem stabilit spațiul de probabilitate $(\Omega, \mathcal{F}, \mathbb{P})$ și variabila X, conform afirmațiilor de mai sus obținem:

$$\mathbb{P}(X \in S) = \sum_{x \in S} u(x) = \frac{|S|}{|\mathbf{E}|} \tag{93}$$

de unde putem deduce că:

$$|S| = |\mathbf{E}| \frac{|S|}{|\mathbf{E}|} \tag{94}$$

Așa cum am menționat, este iluzoriu să încercăm să calculăm exact $|S|$ vom încerca să găsim o estimare pentru $|S|$, folosind următorul algoritm:

1. Generăm un șir de variabile aleatoare independente și identic distribuite uniform $X_1, ...X_n$ pe $\mathbf{E}$

   i) Generăm o matrice de 9×9 linie cu linie

   ii) Fiecare linie se generează de la stânga la dreapta

   iii) Pentru fiecare celulă, se alege din mulțimea C o valoare în mod aleator în așa fel încât să nu coincidă cu celelalte valori de pe linie; cu alte cuvinte, pentru fiecare linie generăm o permutare aleatoare a mulțimii C

2. Folosind metoda Monte Carlo standard, vom obține un estimator pentru $\mathbb{P}(X \in S)$

$$\hat{l} = \frac{1}{n} \sum_{i=1}^{n} I_{(X_i \in S)} \tag{95}$$

3. Cu ajutorul lui $\hat{l}$, putem estima valoarea aproximativă a numărului de soluții Sudoku

$$\widehat{|S|} = |\Omega| \hat{l} \tag{96}$$

Remarca 4.5. *(Problema Evenimentelor Rare)*
Rulând simularea Monte Carlo standard pentru $n = 1000000$ *4.2 și* $n = 100000000$ *4.3 nu am obținut nicio soluție Sudoku. Conform [8], numai o matrice în* $1.6 * 10^{28}$ *de matrici din* Ω *reprezintă o soluție Sudoku și ar dura în medie* $1.8 * 10^{15} - 1.8 * 10^{18}$ *ani pentru a o obține folosind acest algoritm. Putem concluziona că metoda Monte Carlo simplă nu este eficientă în acest caz,* $(X \in S)$ *fiind un eveniment rar în spațiul nostru de probabilități.*

Deoarece $(X \in S)$ este un eveniment rar, se recomandă folosirea unor alte metode precum importance sampling, abordare pe care o vom aborda și noi în cele ce urmează.

Remarca 4.6. *Densitatea u, cât și densitatea de importance sampling g, sunt date de algoritmii folosiți pentru generarea matricilor de* 9×9. *În consecință, schimbarea de probabilitate aferentă procedeului de importance sampling implică modificarea algoritmului de generare într-unul care favorizează apariția matricilor de tip soluție Sudoku. Adică vrem un algoritm implementabil care să dea o densitate g cât mai apropiată de cea optimă astfel încât:*

$$\mathbb{P}_g(X \in S) \gg \mathbb{P}_u(X \in S) \tag{97}$$

Din punct de vedere teoretic, schimbând densitatea obținem:

$$\mathbb{P}_u(X \in S) = E_u[I_{X \in S}] = E_g\Big[I_{(X \in S)} \frac{u(X)}{g(X)}\Big] \tag{98}$$

iar estimatorul $\hat{l}$ devine:

$$\hat{l} = \frac{1}{n} \sum_{i=1}^{n} I_{(X_i \in S)} \frac{u(X)}{g(X)} \tag{99}$$

Estimarea lui $\widehat{|S|}$ se rezumă la următoarea formulă:

$$\begin{aligned} \widehat{|S|} &= \tfrac{|\Omega|}{n} \textstyle\sum_{i=1}^{n} I_{(X \in S)} \tfrac{u(X)}{g(X)} \\ &= \tfrac{|\Omega|}{n} \textstyle\sum_{i=1}^{n} I_{(X \in S)} \tfrac{|\Omega|^{-1}}{g(X)} \\ &= \tfrac{1}{n} \textstyle\sum_{i=1}^{n} I_{(X \in S)} \tfrac{1}{g(X)} \end{aligned} \tag{100}$$

Rămâne deci să decidem cum considerăm g. Ca principiu, g trebuie să conducă la un algoritm de generare care, în loc să aleagă aleator termeni dintr-o permutare a lui C (astfel ținând cont doar de unicitatea elementelor de pe o linie), tine cont de cifrele deja existente în matrice și verifică să nu existe repetiții nici pe coloanele sau în pătratele aferente celulelor. Practic, alegerea lui g încearcă să restrângă cazurile posibile date de generarea de la metoda Monte Carlo simplă, venind cu o lista de cazuri fezabile pentru o soluție Sudoku. În cazul în care nu există cazuri fezabile, algoritmul revine la cel de generare de la metoda Monte Carlo simpla (și se utilizează și pentru generarea primului rând).

Pe scurt, algoritmul urmează următorii pași de generare pentru o matrice:

1. Fiecare matrice se generează linie cu linie
2. Fiecare linie se generează de la stânga la dreapta, verificând valoarea potențială fiecărei celule
3. Pentru generarea valorii unei celule b_{ij}:
   - i) Se verifică valorile deja generate pe linia i, eliminându-se din mulțimea de valori fezabile
   - ii) Se verifică valorile deja generate pe coloana j, eliminându-se din mulțimea de valori fezabile
   - iii) Se verifică valorile deja generate în pătratul corespunzător celulei b_{ij}, eliminându-se din mulțimea de valori fezabile
   - iv) Se alege în mod aleator una dintre valorile fezabile rămase. Dacă mulțimea valorilor fezabile este vidă, se recurge la generarea de valori din algoritmul precedent în care se ține cont doar de elementele de pe linie. Notăm mulțimea valorilor fezabile asociată unei celule cu $F_{b_{ij}}$.

4. Dacă s-a ajuns în cazul în care mulțimea valorilor posibile este vidă, algoritmul de generare se reduce la cel de la metoda Monte Carlo simplă.

Pentru finalizarea algoritmului de aproximare pentru $\widehat{|S|}$ ne rămâne doar să deducem densitatea g. Dacă avem un șir de variabile aleatoare independente $X_1, ..., X_n$ sub o densitate g și considerăm $X = (X_1, ..., X_n)$ un vector de variabile aleatoare, atunci $X \sim g$ cu $g(x) = g(x_1) \cdot g(x_2) \cdot ... \cdot g(x_n)$. Datorită algoritmului de generare putem privi matricea ca pe un vector de variabile aleatoare independente. Ajungem la următoarea formulă.

$$g(x) = \begin{cases} \prod_{i,j} \frac{1}{|F_{b_{ij}}|}, & daca\ |F_{b_{ij}}| > 0 \forall i, j = \overline{1,9} \\ u(x), & altfel \end{cases} \tag{101}$$

Rulând algoritmul de importance sampling pentru $n = 1000000$ obținem aproximativ 3000 de soluții Sudoku.

Conform [7], media lui $\widehat{|A|}$ în 100 de experimente cu $n = 1000000$ este $6.662 \cdot 10^{21}$ ceea ce este în concordanta cu rezultatele obținute individual (pe un n mai mic din cauza limitărilor tehnice). De asemenea, conform [8], numărul total de Sudoku este confirmat la $6670903752021072936960 \approx 6.671 \cdot 10^{21}$ ceea ce indică corectitudinea algoritmului utilizat, cât și eficiența și acuratețea metodei de importance sampling comparativ cu metoda Monte Carlo simplă în vederea estimării evenimentelor rare.

```
Insert number of matrices to be generated:1000000
0
0.0
```

Figure 4.1: Rezultatul simulării folosind metoda Monte Carlo simplă pentru $n = 1000000$

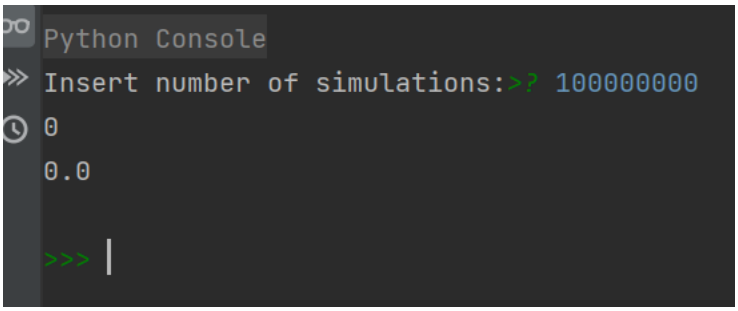

Figure 4.2: Rezultatul simulării folosind metoda Monte Carlo simplă pentru $n = 100000000$

```
Insert number of matrices to be generated:1000000
3100
6.543847295063285e+21
```

Figure 4.3: Rezultatul simulării folosind importance sampling

```
Insert number of simulations:10

Insert number of matrices to be generated:1000000
6.63603800560018e+21
```

Figure 4.4: Media a 10 simulări folosind importance sampling

5 Cod

În aceasta secțiune am adăugat codul folosit pentru simulările folosind metoda de importance sampling, cât și codul pentru metoda Monte Carlo simplă.

Codul pentru verificarea unei soluții Sudoku

```
import numpy as np
import random
#%%
def check_line(line):
    sum = 0
    for n in line:
        sum = sum + 2**(n-1)
    if sum != 511:
        return 0
    return 1

def check_col(col):
    sum = 0
    for n in col:
        sum = sum + 2**(n-1)
    if sum != 511:
        return 0
    return 1

def check_box(line,col,sud):
    sum = 0
    for i in range(line,line+3):
        for j in range(col,col+3):
            sum = sum + 2**(sud[i][j]-1)

    if sum != 511:
        return 0
    return 1

def check(sudoku):
    line_check = 1
    col_check = 1
    box_check = 1
    for i in range(9):
        line_check = line_check*check_line(sudoku[i])
        col_check = col_check*check_col(sudoku[:,i])
        if(i%3==0):
            for j in range(0,9,3):
```

```
                    box_check = box_check*check_box(i,j,sudoku)
    return (line_check*col_check*box_check)
```

Codul pentru metoda Monte Carlo simplă

```
control_group = [1, 2, 3, 4, 5, 6, 7, 8, 9]
number_of_matrices = int(input("Insert number of matrices to be generated:"))
number_of_sudoku = 0

for iteration in range(number_of_matrices):

    sudoku_matrix = [[], [], [], [], [], [], [], [], []]

    for line in range(9):
        numbers_list = control_group.copy()
        for column in range(9):
            new_element = random.choice(numbers_list)
            sudoku_matrix[line].append(new_element)
            numbers_list.remove(new_element)


    matrix = np.array(sudoku_matrix).reshape(9, 9)
    matrix = np.array(sudoku_matrix).reshape(9, 9)
    number_of_sudoku = number_of_sudoku + check(matrix)


print(number_of_sudoku)
print((1.091*(10**50))*(number_of_sudoku/number_of_matrices))
```

Codul pentru funcția care generează mulțimea numerelor fezabile a unei celule

```
def feasible_numbers(constructed_matrix, current_column, current_line, numbers
    , box_line, box_column):
    for position in range(9):
        if len(constructed_matrix[position]) - 1 >= current_column and
    constructed_matrix[position][current_column] in numbers:
            numbers.remove(constructed_matrix[position][current_column])
        if len(constructed_matrix[current_line]) - 1 >= position and
    constructed_matrix[current_line][position] in numbers:
            numbers.remove(constructed_matrix[current_line][position])

    for i in range(box_line, box_line + 3):
        for j in range(box_column, box_column + 3):
            if len(constructed_matrix[i]) - 1 >= j and constructed_matrix[i][j
    ] in numbers:
                numbers.remove(constructed_matrix[i][j])
```

```
    return numbers
```

Codul pentru importance sampling

```
control_group = [1, 2, 3, 4, 5, 6, 7, 8, 9]
number_of_matrices = int(input("Insert number of matrices to be generated:"))
number_of_sudoku = 0
likelihood_ratio_sum = 0

for iteration in range(number_of_matrices):

    likelihood_ratio = 1
    sudoku_matrix = [[], [], [], [], [], [], [], [], []]

    box_current_line = 0
    box_current_column = 0

    for line in range(9):
        if line % 3 == 0:
            box_current_line = line
        for column in range(9):
            if column % 3 == 0:
                box_current_column = column
            if len(sudoku_matrix[line]) - 1 < column:
                feasible_numbers_list = feasible_numbers(sudoku_matrix, column
    , line, control_group.copy(), box_current_line, box_current_column)
                if len(feasible_numbers_list) > 0:
                    current_number = random.choice(feasible_numbers_list)
                    sudoku_matrix[line].append(current_number)
                    likelihood_ratio = likelihood_ratio * len(
    feasible_numbers_list)
                else:
                    sudoku_matrix[line].append(random.choice(control_group))
    matrix = np.array(sudoku_matrix).reshape(9, 9)
    number_of_sudoku = number_of_sudoku + check(matrix)
    likelihood_ratio_sum = check(matrix)* likelihood_ratio +
    likelihood_ratio_sum

print(number_of_sudoku)
print(likelihood_ratio_sum / number_of_matrices)
```

Codul pentru figura 4.4

```
def iteratie(number_of_matrices):
    control_group = [1, 2, 3, 4, 5, 6, 7, 8, 9]
    number_of_sudoku = 0
```

```
    likelihood_ratio_sum = 0

    for iteration in range(number_of_matrices):

        likelihood_ratio = 1
        sudoku_matrix = [[], [], [], [], [], [], [], [], []]

        box_current_line = 0
        box_current_column = 0

        for line in range(9):
            if line % 3 == 0:
                box_current_line = line
            for column in range(9):
                if column % 3 == 0:
                    box_current_column = column
                if len(sudoku_matrix[line]) - 1 < column:
                    feasible_numbers_list = feasible_numbers(sudoku_matrix,
    column, line, control_group.copy(), box_current_line, box_current_column)
                    if len(feasible_numbers_list) > 0:
                        current_number = random.choice(feasible_numbers_list)
                        sudoku_matrix[line].append(current_number)
                        likelihood_ratio = likelihood_ratio * len(
    feasible_numbers_list)
                    else:
                        sudoku_matrix[line].append(random.choice(control_group
    ))
        matrix = np.array(sudoku_matrix).reshape(9, 9)
        number_of_sudoku = number_of_sudoku + check(matrix)
        likelihood_ratio_sum = check(matrix)* likelihood_ratio +
    likelihood_ratio_sum
    return (likelihood_ratio_sum / number_of_matrices)
#%%

estimate = 0
number_of_simulations = int(input("Insert number of simulations:"))
number_of_matrices = int(input("Insert number of matrices to be generated:"))

for i in range(number_of_simulations):
    estimate = estimate + iteratie(number_of_matrices)

print(estimate/number_of_simulations)
```

6 Concluzii

Importance sampling este și va rămâne o metodă fundamentală de simulare, datorită simplității, eficienței și reducerii mari de variantă de care este capabilă, dupa cum s-a putut observa in lucrare. Implementarea procedeului în probleme de numărare poate prezenta dificultăți din cauza densității optime de importance sampling.

În vederea implementării, găsirea unei densități propice și implementarea acesteia în cod reprezintă principalele obstacole. Viitoare inovații în domeniu pot apărea pentru a facilita simularea, metoda Monte Carlo fiind o apariție recentă în matematică supusă unei cercetări constante . Dovada acestui fapt o face chiar metoda cross-entropiei, fiind considerată relativ recentă în domeniul metodelor Monte Carlo.

Bibliografie

[1] C.L. Gherghe, D. Popescu, *Criptografie, coduri, algoritmi*, ed.1 Bucuresti: Editura Universitatii din Bucuresti, 2005

[2] Y-C. Chen(2019), STAT 535 Statistical Machine Learning: Lecture 9 - Monte Carlo Simulations and Sampling[PDF], Available: http://faculty.washington.edu/yenchic/19A_stat535/Lec9_MC.pdf

[3] P-T. De Boer, D.P. Kroese, S. Mannor, R.Y. Rubinstein(2005), "A Tutorial on the Cross-Entropy Method", *Annals of Operations Research*, vol. 134, no. 1, pp. 19-67, Feb. 2005, DOI: https://doi.org/10.1007/s10479-005-5724-z, Available: https://people.smp.uq.edu.au/DirkKroese/ps/aortut.pdf

[4] C.R. Harris, K.J. Millman, S.J. van der Walt et al. **Array programming with NumPy**. Nature 585, 357–362, 2020. DOI: 10.1038/s41586-020-2649-2, Available:https://www.nature.com/articles/s41586-020-2649-2

[5] M.Haugh(2017), IEOR E4703 Monte-Carlo Simulation: Further Variance Reduction Methods [PDF], Available: https://martin-haugh.github.io/files/MonteCarlo/MCS_Var_Red_Advanced.pdf

[6] S.Ling, C. Xing, *Coding Theory: A First Course*, ed. 1 Cambridge: Cambridge University Press, 2004

[7] A. Ridder, 2013. Counting the number of Sudoku's by importance sampling simulation[PDF]. Available: https://personal.vu.nl/a.a.n.ridder/papers/sudoku13.pdf

[8] A. Ridder, 2013, Importance Sampling for Sudoku's[PDF]. Available: https://personal.vu.nl/a.a.n.ridder/lezingen/sudokusim_rm13.pdf

[9] R.Y. Rubinstein, D.P. Kroese, *Simulation and the Monte Carlo Method*, 2nd ed. Hoboken, NJ: John Wiley & Sons, 2008

[10] N.Shimkin(2015), Monte Carlo Methods - Lecture 4: Importance Sampling [PDF], Available: https://webee.technion.ac.il/shimkin/MC15/MC15lect4-ImportanceSampling.pdf