3. tétel

Kaszás Bálint

2019. június 20.

Kivonat

Véletlen számok generálása, numerikus integrálás, Newton-típusú formulák, Gauss-formulák. Monte-Carlo módszer, Markov-lánc Monte-Carlo, hierarchikus bayes-i hálózatok.

1. Véletlen számok generálása

A természettudományos problémák megoldása során gyakran szükség van, például véletlenszerű természeti jelenségek szimulálásakor, véletlen számok generálására. Nyilvánvaló nehézséget jelent, hogy generálást determinisztikus programmal kell végeznünk.
Tekintsük az $r_1, r_2, r_3, ...$ számok sorozatát. Ezt véletlenszerűnek (*randomnak*) mondjuk, hogyha a számok között nincs korreláció. Determinisztikus módszerekkel nem tudunk korrelációmentes sorozatot előállítani, a véletlenszámok között mindenképpen marad korreláció. Ezért, a számítógéppel generált véletlen számok sorozata igazából csupán *pszeudorandom*. Ez azt is jelenti, hogy amennyiben ismerjük r_m-t és az őt megelőző számokat a sorozatban, kellő energiabefektetéssel meg tudjuk mondani r_{m+1}-et is. Kifinomultabb generátorokkat használva a korreláció természetesen csökkenthető. Alternatívaként rendelkezésre állnak olyan adatbázisok is, amelyek *valódi véletlen számokat* tartalmaznak, például valami véletlenszerű jelenség mérési adatait. Ezekből azonban nem tudunk elég gyorsan véletlen számokat lehívni.
Tekintsük az egyik legegyszerűbb pszeudorandom számokat generáló algoritmust, a *lineáris kongruencia generátort*. Az r_i számokat $0 \leq r_i \leq M-1$ intervallumon generáljuk, egy a és egy c állandók választásával.

$$r_i \equiv (r_{i-1}a + c) \bmod M \tag{1}$$

A hátránya, hogy amint megismétlődik egy korábbi random szám, az egész sorozat ismétlődni fog. Ebben az egyszerű példában az r_i sorozat M lépés után ismétlődik, ez a generátor *ciklushossza*. A lineáris kongruencia hátránya továbbá, hogy több dimenzióban nem korrelálatlanok a generált számok, azaz bizonyos síkok mentén helyezkednek el a többdimenziós térben.
A lineáris kongruencia algoritmus akkor működik 'jól', ha a ciklushossz kellően nagy, és az a, c paramétereket is elég nagyra választjuk. A gyakorlatban alkalmazott generátorokban tipikusan $M = 2^{32}$ [5], vagy $M = 2^{64}$ [3].
Megemlítjük, hogy léteznek a lineáris kongruenciánál kifinomultabb algoritmusok is, amik többek között több dimenziós véletlen számokat is elő tudnak állítani. A legnépszerűbb ezek közül a *Mersenne-twister* [1], amely nevét onnan kapta, hogy a ciklushosszát egy Mersenne-prímnek választják.

2. Numerikus integrálás

A numerikus integrálás alapfeladata [4], hogy valamilyen tartományon adott, határozott integrál értékét numerikus módszerekkel kiszámítsuk. Erre azért lehet szükség, mert az integrandusnak gyakran nem ismerjük primitív függvényét.
Általánosan minden módszer leírható úgy, hogy az integrálási tartományban bizonyos helyeken kiértékeljük az integrandust, majd ezeket megfelelő súlyokkal összeadjuk.

2.1. Newton-típusú formulák

A Newton-típusú formulák, vagy Newton-Cotes formulák az integrálási tartományt egyenlő darabokra osztják fel,

$$\int_a^b f(x)\mathrm{d}x \approx \sum_{i=0}^{n} w_i f(x_i), \tag{2}$$

ahol $a = x_0 < x_1 < ... < x_i < ... < x_n = b$, egyenletesen elhelyezve, és w_i az i. integrálási ponthoz tartozó súly, amelyeket különböző alakban vehetünk fel.
Minden Newton típusú formula azon alapul, hogy az integrált egy köztes, $[x_i, x_{i+1}]$ intervallumon közelítjük, majd a kis intervallumok járulékát összeadjuk.
A legegyszerűbb ezek közül a *trapéz-szabály*, amely x_i és x_{i+1} között lineárisan interpolál:

$$\int_{x_i}^{x_{i+1}} f(x)\mathrm{d}x \approx (x_{i+1} - x_i)\frac{f(x_{i+1} + f(x_i))}{2}. \tag{3}$$

Ezt a kis intervallumokra egymás után alkalmazva és a Δx jelölést használva az integrálási pontok távolságára

$$\int_a^b f(x)\mathrm{d}x = \Delta x\frac{f(x_0) + f(x_n)}{2} + \sum_{i=1}^{n-1} f(x_i)\Delta x. \tag{4}$$

A trapéz szabálynál pontosabb formulákat is használhatunk, ezek az elemi intervallumok feletti interpoláció alakjában különböznek. A *Simpson-szabály* szerint például másodfokú polinommal interpolálunk, és ezért szükség lesz az intervallum közepén kiértékelt függvényértékre is. Megtartva a $\Delta x = x_{i+1} - x_i$ jelölést:

$$\int_{x_i}^{x_{i+1}} f(x)\mathrm{d}x \approx \frac{\Delta x}{3}\left(f(x_i) + 4f(x_i + \Delta x/2) + f(x_{i+1})\right). \tag{5}$$

2.2. Gauss-formulák

A Gauss-formulák ehhez képest nem ragaszkodnak az egyenközű függvény-kiértékeléshez. Ha az $f(x)$ integrandusból kiemelünk egy súly-faktort

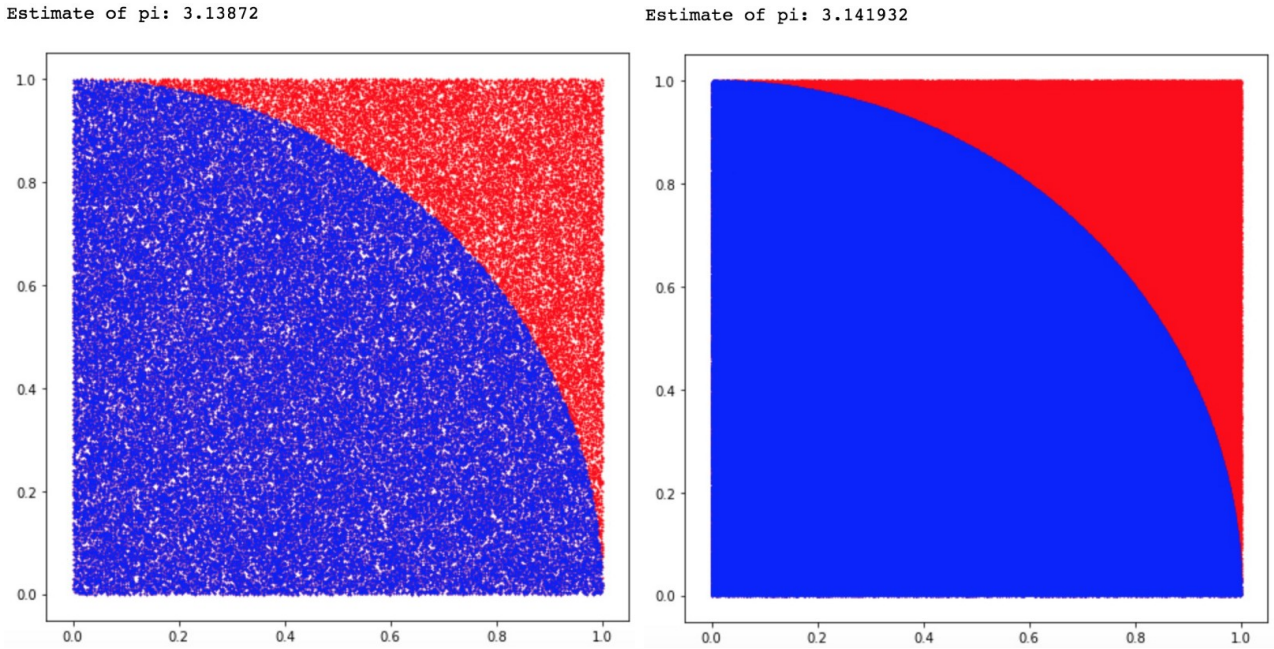
$$\int_a^b f(x)\mathrm{d}x = \int_a^b W(x)g(x)\mathrm{d}x \approx \sum_{i=0}^{n} w_i g(x_i). \tag{6}$$

Az n darab x_i pontok és a w_i súlyok pedig már nem egyenletesek. Úgy választjuk őket, hogy akkor legyen egzakt a módszer, amikor $g(x)$ egy $2n-1$-ed fokú polinom. Mivel a Gauss-formulák a $[-1, 1]$ intervallumra vannak megadva, előtte az integrálunkat is át kell transzformálni erre az intervallumra.
A Gauss-formulák előnye, hogy ugyanannyi függvénykiértékeléssel sokkal pontosabb eredményre vezetnek a Newton-formuláknál. Az integrálási pontok és a súlyok levezetését mellőzzük, ezek rendelkezésre állnak a legtöbb numerikus könyvtárban.

3. Monte-Carlo módszer

A fent vázolt módszerek egyik hátránya hogy több dimenzióra nem, vagy csak rosszul általánosíthatóak. Erre kínál megoldást a *Monte-Carlo* módszer, amellyel sok véletlenszám generálásával tudunk tetszőleges dimenziós integrált kiszámítani. A numerikus integráláson túl a

Monte-Carlo típusú módszereket használják optimalizálási feladatokra is. A közös ezekben, hogy *véletlen számok segítségével oldanak meg egy determinisztikus problémát*. Tekintsük most a numerikus integrálás esetét. A legegyszerűbb példaként az egységsugarú kör területének kiszámítását tárgyaljuk. Az egyszerűség kedvéért vegyük csak az egyik negyedét a körnek, amit az egységnégyzetbe tudunk beleírni.
Hogyha az egységnégyzet belsejében *egyenletes eloszlással* pontokat generálunk, a negyedkör belsejébe eső pontok számának, és az összes generált pont számának aránya a körcikk területét közelíti. A generált pontok számának növelésével tetszőleges pontosság elérhető. A módszert a 1. Ábra mutatja. Ez a területszámítási (vagy térfogat, magasabb dimenzió esetén) módszer



1. ábra. A körcikk területének kiszámítása Monte-Carlo módszerrel. Kék pontok jelölik a körcikk belsejébe esett pontokat. A bal panelen 10^5 pont van elosztva az egységnégyzetben, a jobb panelen pedig 10^6. Felettük pedig π kiszámolt közelítő értéke látható, vagyis a körcikk területének négyszerese. Forrás: [2].

egyszerűen működik tetszőleges alakzatokra is, csupán véletlen pontokról kell eldönteni, hogy bele esnek-e a kívánt tartományba.
A határozott integrál kiszámítására való alkalmazáshoz a következő szemléletes tulajdonságot használjuk ki.

$$I = \int_a^b f(x)\mathrm{d}x = (b-a)\langle f \rangle \tag{7}$$

Vagyis a függvény *átlagát* kell kiszámítanunk az intervallumon. Természetesen ez igaz több dimenzióra is, ilyenkor az integrál értéke a tartomány térfogatának és a függvény átlagának szorzata.
A függvény átlagát pedig már számolhatjuk a Monte-Carlo módszerrel, azaz

$$\langle f \rangle \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i), \tag{8}$$

ahol $x_i \in [a, b]$ számokat egyenletes eloszlással húzzuk az integrálási tartományból. A minták számának növelésével az N számból számolt átlag a függvény átlagértékéhez tart. A relatív hiba pedig $1/\sqrt{N}$ szerint csökken.
Az átlagérték kiszámítása lassan változó függvények esetén tényleg egy hatékony módját adja a keresett integrál közelítésének. Azonban mivel egyenletes eloszlású pontokkal dolgozunk, egy gyorsabban változó függvény esetén már nem ad megbízható eredményt, hiszen az egyenletes mintavételezés könnyen kihagyhatja azokat a részeit a függvénynek, ahonnan nagy járulék jönne az átlagba. Erre a problémára, a Monte-Carlo 'megjavítására' két alapvető módszer van:

3.1. Szórás csökkentés

Tegyük fel, hogy az integrálandó $f(x)$ függvényhez találunk egy $g(x)$-et, amellyel $f(x) - g(x)$ szórása kicsi, és ismerjük $g(x)$ integrálját.

$$\int_a^b g(x)\mathrm{d}x = J.$$

Így $f(x) - g(x)$ integrálját már számolhatjuk Monte-Carló módszerrel, és a keresett I integrál

$$I = \int_a^b f(x)\mathrm{d}x = \int_a^b f(x) - g(x)\mathrm{d}x + J.$$

3.2. Importance sampling

A másik megoldás, hogy az egyenletes mintavételezés helyett úgy választjuk a pontokat, hogy azok főleg a függvény nagy járulékot adó részeiről legyenek. Ehhez kifejezzük I integrált egy $w(x)$ súlyfaktorral

$$I = \int_a^b f(x)\mathrm{d}x = \int_a^b w(x)\frac{f(x)}{w(x)}\mathrm{d}x. \tag{9}$$

Hogyha pedig $w(x)$-szerint húzunk x_i mintákat, vagyis ezt *használjuk valószínűségi eloszlásnak* (az egyenletes helyett), az integrált megkaphatjuk

$$I = \left\langle \frac{f}{w} \right\rangle \approx \frac{1}{N}\sum_{i=1}^{N} \frac{f(x_i)}{w(x_i)}.$$

4. Markov-lánc Monte-Carlo módszer

Az importance sampling kapcsán felmerül az igény arra, hogy tetszőleges eloszlásból generáljunk véletlen számokat. Ez bonyolult feladat is lehet, hogyha a direkt mintavételezés nem opció, vagy nagyon számításigényes (különösen magas dimenziós eloszlások esetén). Erre nyújtanak megoldást a Markov-lánc Monte-Carlo módszerek (MCMC). Nevüket onnan kapták, hogy egy megfelelő határeloszlással rendelkező Markov-Lánc szimulálásával kapjuk a keresett mintát.

4.1. Metropolis-Hastings algoritmus

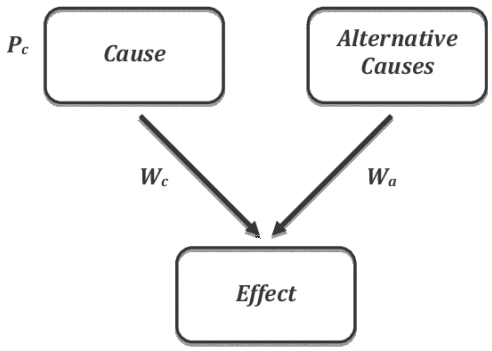
Egy példa az MCMC algoritmusokra a Metropolis-Hastings algoritmus, amely szerepet kap a termodinamikai szimulációkban is (lásd 4. tétel).

Az algoritmussal egymás után generálunk véletlen pontokat, méghozzá úgy, hogy minél több pontot generálunk, az eloszlásuk annál jobban közelíti a keresett valószínűségi eloszlást. A pontokat iteratívan kapjuk, a következő pont mindig csak az előzőtől függ (így válik Markov-lánccá). Az aktuális pontból kiindulva választunk egy lehetséges új pontot. Ezt a lehetséges új pontot pedig egy megadott valószínűséggel *fogadjuk el*, egyébként elutasítjuk.
Az algoritmus a következő, amennyiben $f(x)$ eloszlással szeretnénk mintákat generálni.

- Választunk egy kezdeti x_0 pontot és egy $g(x)$ eloszlást, amelyből könnyű mintavételezni. Ezt fogjuk arra használni, hogy új lehetséges mintákat húzzunk.
- Az n. iterációs lépésben:
- Válasszunk $g(x)$ szerint egy x' pontot
- Számoljuk ki az $\alpha = f(x')/f(x_n)$ hányadost
- x'-t α valószínűséggel fogadjuk el, azaz húzzunk egy u számot egyenletesen $[0, 1]$ között.
- $u \leq \alpha$ esetén elfogadjuk x'-t, azaz $x_{n+1} = x'$.
- $u > \alpha$ esetén pedig elutasítjuk, $x_{n+1} = x_n$.

5. Hierarchikus bayes-i hálózatok

A bayes-i hálózat egy olyan statisztikai modell, amelyben a változók egy irányított gráfba rendeződtek, ezzel kifejezve az ok-okozati viszonyt, illusztrációként lásd a 2. ábrát. Tegyük



2. ábra. Általános bayes-i hálózat, irányított gráfba rendezve.

fel, hogy a mért x adatainkhoz egy paramétert rendelünk, és annak a valószínűségét keressük (posterior), hogy a paraméter értéke Θ, a mért x adatainkat feltéve. Bayes-tétele alapján ez a valószínűség

$$p(\Theta|x) \sim p(x|\Theta)p(\Theta), \tag{10}$$

ahol $p(x|\Theta)$ a likelihood (a megfigyel empirikus eloszlása x-nek), és $p(\Theta)$ Θ prior eloszlása.
Egy hierarchikus bayes-i hálózatban azonban Θ is függhet további paraméterektől (további, őt okozó változóktól), például φ-től. Ekkor a Θ prior eloszlását $p(\Theta|\varphi)$ váltja le, vagyis

$$p(\Theta, \varphi|x) \sim p(x|\Theta)p(\Theta|\varphi)p(\varphi). \tag{11}$$

A folyamatot addig ismételjük, amíg a felállított statisztikus modell, a bayes-i hálózat minden csúcsát bele nem számoltuk a posterior valószínűségbe. Ezt a posterior eloszlást, vagyis a paraméter eloszlását a gyakorlatban Markov-Lánc Monte-Carlo módszerrel számoljuk ki. A paraméterekre feltett prior eloszlásokból és a likelihood-okból mintákat húzva, előállítjuk a keresett szorzatot.

Hivatkozások

[1] Random Number Generation in C++11. Standard C++ Foundation. https://isocpp.org/files/papers/n3551.pdf.

[2] An overview of Monte-Carlo methods. https://towardsdatascience.com/an-overview-of-monte-carlo-methods-675384eb1694.

[3] Donald E. Knuth. *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1997.

[4] Rubin H. Landau, Jose Paez, and Cristian C. Bordeianu. *A Survey of Computational Physics: Introductory Computational Science*. Princeton University Press, Princeton, NJ, USA, har/cdr edition, 2008.

[5] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1988.