3. tétel

Kaszás Bálint

2019. június 19.

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 $M = 2^{32}$ [3], vagy $M = 2^{64}$ [2].
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 [**?**], hogy egy 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

4. Markov-lánc Monte-Carlo módszer

5. Hierarchikus bayes-i hálózatok

Hivatkozások

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

[2] 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.

[3] 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.