Monte Carlo módszerek

Monte Carlo módszerek

- Alapja a véletlen minták kiértékelése
  - Matematikai rendszerek
  - Fizikai szimuláció
- Sok szabadság fokú csatolt rendszerek
  - Folyadékok, sejt struktúrák, kapcsolt szilárd rendszerek
- Nagy bizonytalanságú rendszerek
  - Üzleti modellek, kockázat elemzés
- Nagy dimenziójú integrálok
  - Összetett peremfeltételek

Monte Carlo módszerek

- Története
  - 1930, Enrico Fermi
    - Neutron diffúzió
  - 1946, Stanislaw Ulam
    - Manhattan project
    - Nukleáris árnyékolás
      - Neutronok szabad úthossza különböző anyagokban
      - Energia változás szóródás közben
      - Analitikusan nem volt megoldható
    - Neumann János nevezte el
      - Monakói Monte Carlo kaszinó alapján

Monte Carlo módszerek

- Monte Carlo szimuláció
  - Valószínűség eloszlás mintavételezése
  - A minták alapján lehetséges kimenetek meghatározása
  - A lehetséges kimenetek valószínűségének számítása

Monte Carlo módszerek

- Numerikus integrálás
  - Kis dimenzió esetén jól működik
  - Nagy dimenzió esetén problémás
    - A függvény kiértékelések száma robbanásszerűen nő
    - A peremfeltételeket nem könnyű 1D integrálokra visszavezetni
- Monte Carlo integrálás
  - Véletlen mintapontokban az integrál kiértékelése
  - Az eredmények „átlagolása”
  - Konvergencia $1/\sqrt{N}$, a nagy számok törvénye alapján

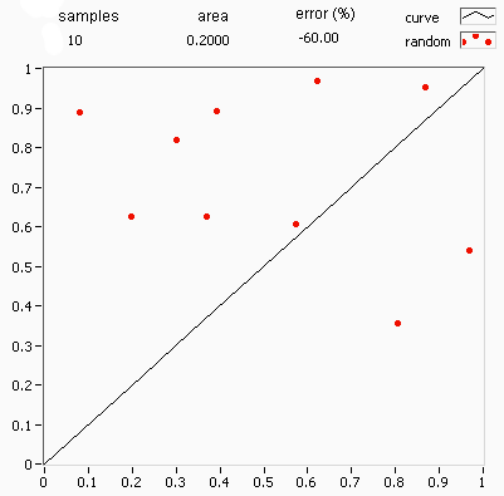
Monte Carlo módszerek

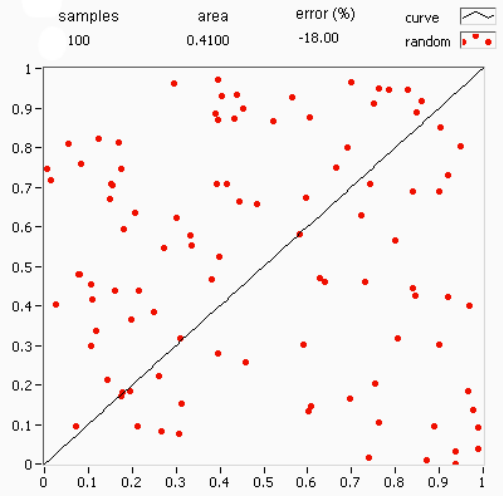
- Monte Carlo integrálás

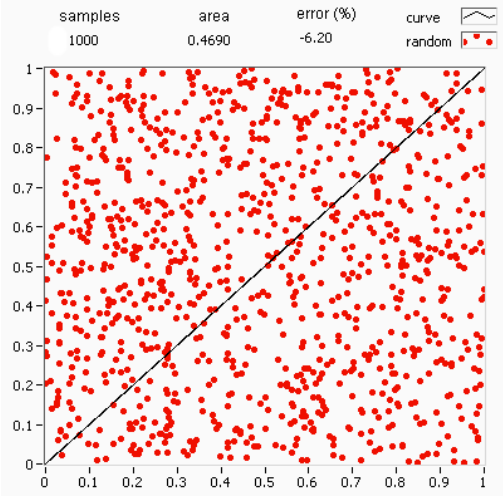
$$I = \int_{a_1}^{b_1} dx_1 \int_{a_2}^{b_2} dx_2 \cdots \int_{a_n}^{b_n} dx_n \cdot f(x_1, x_2, \ldots, x_n) = \int_V f(\bar{x}) d\bar{x}$$

$$I \approx Q_n \equiv V \frac{1}{N} \sum_{i=1}^{N} f(\bar{x}_i) = V \langle f \rangle$$

$$\boxed{I = \lim_{N \to \infty} Q_N}$$







Monte Carlo módszerek

- Monte Carlo integrálás hibája

$$Var(f) \equiv \sigma_N^2 = \frac{1}{N-1}\sum_{i=1}^{N}\left(f(x) - \langle f \rangle\right)^2$$

$$Var\left(\sum_{i=1}^{N} Y_i\right) = \sum_{i=1}^{N} Var(Y_i)$$

$$Var(Q_N) = V^2 \frac{Var(f)}{N} = V^2 \frac{\sigma_N^2}{N}$$

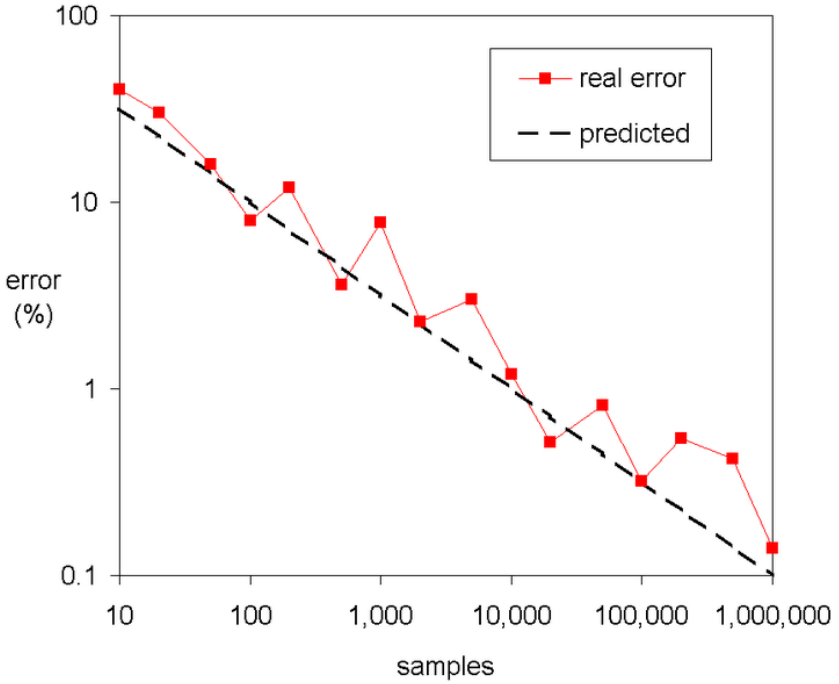
  - ha $\{\sigma_1^2, \sigma_2^2, \ldots, \sigma_n^2\}$ sorozat korlátos, akkor a variancia nullához tart aszimptotikusan $1/N$ -nel

Monte Carlo módszerek

- Monte Carlo integrálás hibája

$$\delta Q_N \approx \sqrt{Var(Q_N)} = V^2 \frac{\sigma_N}{\sqrt{N}}$$

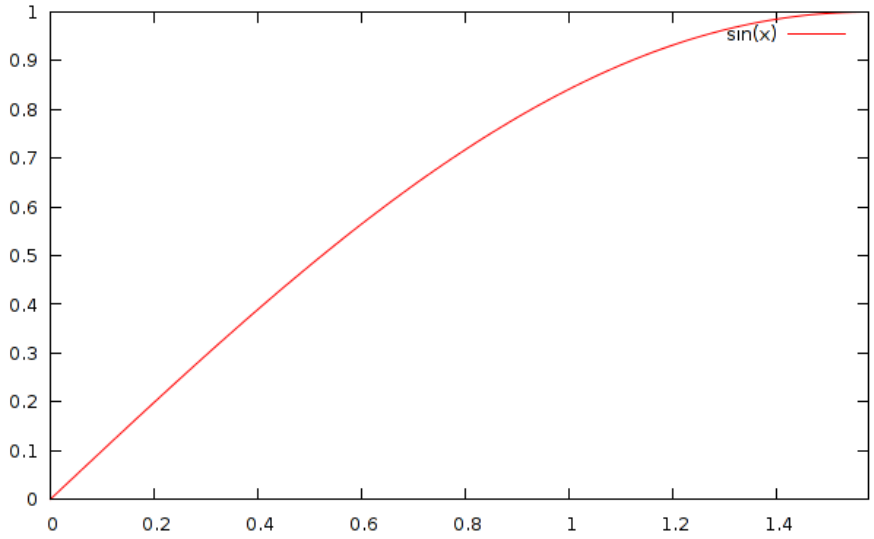
  - a hiba $1/\sqrt{N}$ -nel csökken

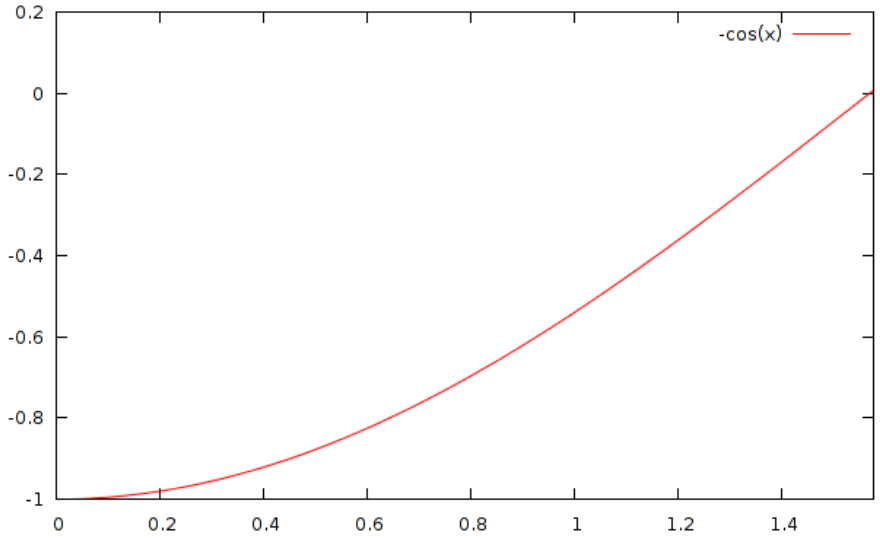


Monte Carlo integrálás

- 1D Monte Carlo integrálás

$$I = \int_0^{\pi/2} \sin(x)dx = -\cos(x)\Big|_0^{\pi/2} = -\cos(\pi/2) - (-\cos(0)) = 1$$





Monte Carlo integrálás

- Hatékonyan implementálható a GPU-n
  - Független minták kiértékelhetőek szálanként
  - Az eredmény redukcióval számítható

```
#define M_PIP2 1.57796327f
__kernel
void mcInt1D(const int sampleNumber, __global float* integral){
  int id = get_global_id(0);

  float w = 1.0f / sampleNumber;
  float partialIntegral = 0.0f;
  for(int i = 0; i < sampleNumber; ++i){
    float rnd = (float)RAND();
    partialIntegral += sin(rnd * M_PIP2) * w * M_PIP2;
  }
  integral[id] = partialIntegral;
}
```

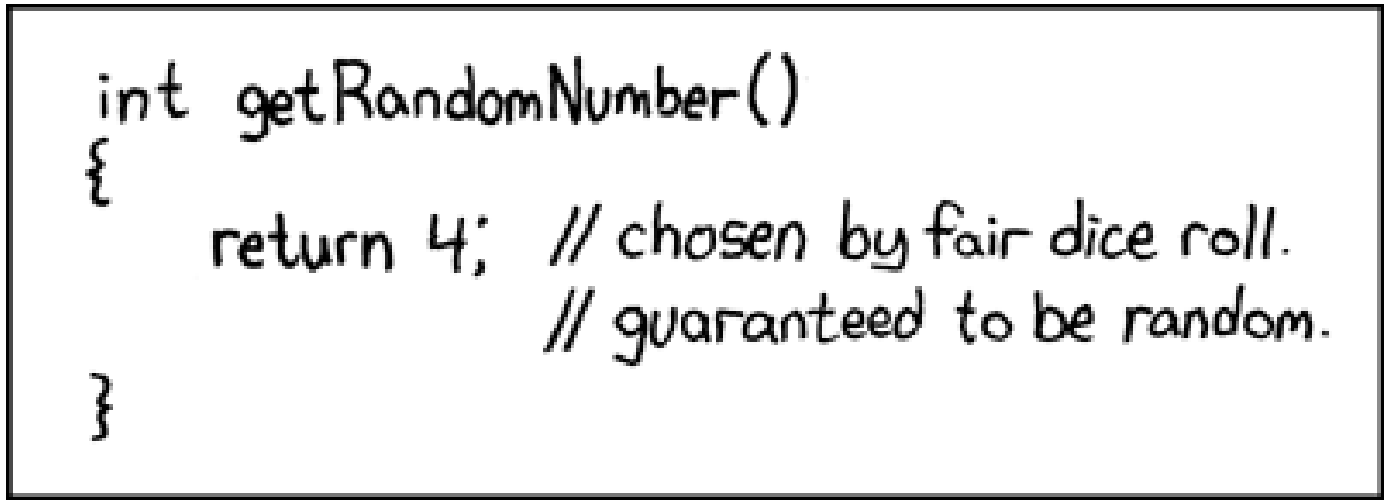
Monte Carlo integrálás

- 1D Monte Carlo integrálás

Minták száma	Integrál
1+e1	0.981062
1+e1	1.04901
1+e3	1.00353
1+e4	1.0059
1+e5	1.00888
1+e6	1.00751
1+e7	1.00716

Véletlen számok

- A Monte Carlo módszerek lelke
  - Hogyan generálhatóak?
  - Milyen tulajdonságaik vannak?



http://xkcd.com/221/

Véletlen számok generálása

- Valódi véletlen szám generátor
  - Valamilyen fizikai folyamat alapján
    - Atmoszférikus zaj (random.org)
    - Hardware megszakítások (linux kernel)
    - Radioaktív bomlások száma és ideje
    - Kvantum mechanikai folyamatok
  - Titkosításhoz használják elsődlegesen
  - Tipikusan lassú és nehézkes a használatuk

Véletlen számok generálása

- Kvázi random szám generátor
  - A cél az *n* dimenziós tér egyenletes kitöltése
  - A konstrukció során alacsony diszkrepancia

  - Halton sorozat
  - Sobol sorozat
  - van der Corput sorozat

  - Magasabb dimenzióknál nehézkes lehet alkalmazni

Véletlen számok generálása

- Álvéletlen szám generátor
  - Determinisztikus algoritmusok
  - Legyen hasonlóan kaotikus mint a valódi véletlen
  - Legyen hosszú a periódus ideje

  - Problémák
    - Bizonyos kiinduló állapotokra a periódus rövidül
    - Nagy mintaszámnál sérülhet az eloszlás
    - Korrelálhatnak a generált minták
    - Nagy dimenziókra kiterjesztés

Véletlen számok minősége

- A véletlen számok minősége kulcsfontosságú!
  - Statisztikai minőség
    - Mennyire kaotikus?
    - Mennyire jó eloszlást generál?
  - Periódus hossz
    - Mikor kapunk vissza egy korábbi minta sorozatot?
  - Diszkrepancia
    - Mennyire egyenletesen tölti ki a teret?

Statisztikai minőség

- Diehard tesztek
  - Születésnap paradoxon
    - Véletlen pontok távolságának exponenciális eloszlásúnak kell lennie.
  - Squeeze teszt
    - Szorozzuk a 2^{31}-ent [0:1] közötti véletlen számmal addig amíg az eredmény 1 lesz. A szükséges szorzások számának azonos eloszlásúnak kell lennie mint az eredeti eloszlás.
  - Parkoló teszt
    - Helyezzünk el véletlenszerűen egység köröket egy 100x100-as négyzetben. Amennyiben az aktuális kör átlapolódna egy másikkal válasszunk új pozíciót. 12,000 próba után a sikeresen elhelyezett körök száma normál eloszlást kell kövessen.

Periódus hossz

- A determinisztikus generátor körbe fog fordulni!
  - A kérdés az, hogy mikor!
- Mitől függ a periódus hossza?
- Hogyan növelhető?
- Mikor elfogadható a periódus?
  - Ha n véletlen mintát használunk, a periódus legyen legalább n^2

Diszkrepancia

- Mennyire egyenletesek a generált minták?

$$\lim_{n\to\infty} \frac{\left|\{s_1,\ldots,s_n\}\cap[c,d[\right|}{n} = \frac{d-c}{b-a}$$

- Diszkrepancia

$$D(N) = \sup_{a\le c\le d\le b}\left|\frac{\left|\{s_1,\ldots,s_n\}\cap[c,d[\right|}{N} - \frac{d-c}{b-a}\right|$$

  - A sorozat egyenletes eloszlású, ha $\lim_{N\to\infty} D(N) = 0$

Alacsony diszkrepanciájú sorozat

- Halton sorozat
  - Kvázi véletlen, alacsony diszkrepanciájú sorozat

```
FUNCTION(index, base)
BEGIN
  result = 0;
  f = 1 / base;
  i = index;
  WHILE (i > 0)
  BEGIN
       result = result + f * (i % base);
       i = FLOOR(i / base);
       f = f / base;
   END
   RETURN result;
END
```

b = 2	b = 3
1/2	1/3
1/4	2/3
3/4	1/9
1/8	4/9
5/8	7/9
3/8	2/9
7/8	5/9
1/16	8/9
9/16	1/27
…	…

Alacsony diszkrepanciájú sorozat

```
__kernel
void haltonSequence(const int randomNumbers,
                    const int base, __global float* randomGPU){

  int id = get_global_id(0);
  int maxID = get_global_size(0);

  float inv_base = 0.0;
  float rng = 0.0;

  seedHalton(id * randomNumbers, base, &inv_base, &rng);

  for(int i=0; i < randomNumbers; ++i){
    randomGPU[id+i*maxID]=stepHalton(&rng, inv_base);
  }
}
```

Alacsony diszkrepanciájú sorozat

```
void seedHalton(ulong i, int base,
                float* inv_base, float* value){

  float f = (*inv_base) = 1.0/base;

  (*value) = 0.0;

  while( i > 0){
    (*value) += f * (float)(i % base);
    i /= base;
    f *= (*inv_base);
  }
}
```

Alacsony diszkrepanciájú sorozat

```
float stepHalton(float *value, float inv_base){
  float r = 1.0 - (*value) - 0.0000000001;
  if(inv_base < r) {
    (*value) += inv_base;
  } else {
    float h = inv_base, hh;
    do{
      hh = h;
      h *= inv_base;
    } while (h >= r);
    (*value) += hh + h - 1.0;
  }
  return (*value);
}
```

Álvéletlen generátorok

- Lineáris kongruencia generátor
  - Knuth (1969)
  - Átmenet függvény: $x_{n+1} = (ax_n + c) \bmod m$
  - Könnyen implementálható
  - Ismert statisztikai hibái vannak!

Lineáris Kongruencia Generátor

```
uint stepLCG(uint *z, uint A, uint C){
  return (*z) = (A * (*z) + C);
}

__kernel
void randomLCG(const int randomNumbers,
               __global float* randomsSeed,
               __global float* randomGPU){

  int id = get_global_id(0);
  int maxID = get_global_size(0);

  uint rng = randomsSeed[id];
  for(int i=0; i < randomNumbers; ++i){
    randomGPU[id + i * maxID] =
     (float)stepLCG(&rng, 1664525, 1013904223UL) / 0xffffffff;
  }
}
```

Álvéletlen generátorok

- Késleltetett Fibonacci Generátor
  - Knuth (1969)
  - Átmenet függvény: $x_{n+1} = \left(x_n \otimes x_{n-k}\right) \bmod m$
  - Statisztikailag jó, ha *k* nagy
  - Nagy állapot változó tér

Késleltetett Fibonacci Generátor

```
uint stepLFG(uint *z, __global uint *znmk, uint A, uint C){
  return (*znmk) = (*z) = (A * (*z) + C) + (*znmk);
}

__kernel
void randomLFG(const int randomNumbers, __global float* randomsSeed,
               const int randomStateSize, __global uint* randomState,
               __global float* randomGPU){

  int id = get_global_id(0);
  int maxID = get_global_size(0);

  // bootstrap
  uint rng = randomsSeed[id];
  for(int i=0; i < randomStateSize; ++i){
    randomState[id + i * maxID] =  stepLCG(&rng, 1664525, 1013904223UL);
  }

  // Lagged Fibonacci Generator
  int nmkIndex = 0;
  for(int i=0; i < randomNumbers; ++i){
    randomGPU[id + i * maxID] =
       (float)stepLFG(&rng, &randomState[nmkIndex], 1664525, 1013904223UL) / 0xffffffff;

    nmkIndex = (nmkIndex + 1) % randomStateSize;
  }

}
```

Álvéletlen generátorok

- Kombinált Tausworthe Generátor
  - Az alapja egy bináris mátrix transzformáció
  - Vektor sorozatokat állít elő
  - A független sorozatokat kombinálja

  - Nagyobb peridóus idejű (pl. 2^{113})
  - Magasabb dimenziókban korrelációt mutathat!

Kombinált Tausworthe Generátor

```
uint stepCTG(uint *z, uint S1, uint S2, uint S3, uint M){
  uint b=((((*z)<<S1)^(*z))>>S2);
  return (*z) = ((((*z)&M)<<S3)^b);
}

__kernel
void randomCTG(const int randomNumbers, __global float2* randomsSeed,
                                        __global float* randomGPU){
  int id = get_global_id(0);
  int maxID = get_global_size(0);

  uint rng1 = randomsSeed[id].x;
  uint rng2 = randomsSeed[id].y;
  for(int i=0; i < randomNumbers; ++i){
    uint randNum = stepCTG(&rng1, 13, 19, 12, 4294967294UL)^
                   stepCTG(&rng2,  2, 25,  4, 4294967288UL);
    randomGPU[id + i * maxID] =  (float)randNum / 0xffffffff;
  }
}
```

Álvéletlen generátorok

- Hibrid Generátor
  - Különböző típusú generátor kombinációja
  - Pl. Lineáris Kongruencia és Tausworthe
  - Jobb statisztikai tulajdonság és hosszabb periódus

```
float stepHybrid(uint* rng1, uint* rng2, uint* rng3, uint* rng4){
  return 2.3283064365387e-10 * (                        // 2^121
          stepCTG(rng1, 13, 19, 12, 4294967294UL) ^     // 2^31-1
          stepCTG(rng2, 2, 25, 4, 4294967288UL)   ^     // 2^30-1
          stepCTG(rng3, 3, 11, 17, 4294967280UL)  ^     // 2^28-1
          stepLCG(rng4,1664525,1013904223UL)            // 2^32
         );
}
```

Álvéletlen generátorok

- Mersenne Twister
  - Makoto Matsumo és Takuji Nishimura (1997)
  - Periódus ideje egy Mersenne prím szám
    - Mersenne prím: $M_p = 2^p - 1$
  - Nagyon nagy periódus idő (2^{19937}-1)
  - Nagyon jó statisztikai tulajdonságokkal rendelkezik

Mersenne Twister

```
__kernel void MersenneTwister(__global float* d_Rand,
                              __global mt_struct_stripped* d_MT,
                              int nPerRng){
    int globalID = get_global_id(0);

    int iState, iState1, iStateM, iOut;
    unsigned int mti, mti1, mtiM, x;
    unsigned int mt[MT_NN], matrix_a, mask_b, mask_c;

    //Load bit-vector Mersenne Twister parameters
    matrix_a = d_MT[globalID].matrix_a;
    mask_b   = d_MT[globalID].mask_b;
    mask_c   = d_MT[globalID].mask_c;

    //Initialize current state
    mt[0] = d_MT[globalID].seed;
    for (iState = 1; iState < MT_NN; iState++)
        mt[iState] = (1812433253U * (mt[iState - 1] ^ (mt[iState - 1] >> 30)) + iState) & MT_WMASK;

    iState = 0;
    mti1 = mt[0];
    for (iOut = 0; iOut < nPerRng; iOut++) {
        iState1 = iState + 1;
        iStateM = iState + MT_MM;
        if(iState1 >= MT_NN) iState1 -= MT_NN;
        if(iStateM >= MT_NN) iStateM -= MT_NN;
        mti  = mti1;
        mti1 = mt[iState1];
        mtiM = mt[iStateM];

            // MT recurrence
        x = (mti & MT_UMASK) | (mti1 & MT_LMASK);
            x = mtiM ^ (x >> 1) ^ ((x & 1) ? matrix_a : 0);

        mt[iState] = x;
        iState = iState1;

        //Tempering transformation
        x ^= (x >> MT_SHIFT0);
        x ^= (x << MT_SHIFTB) & mask_b;
        x ^= (x << MT_SHIFTC) & mask_c;
        x ^= (x >> MT_SHIFT1);

        //Convert to (0, 1] float and write to global memory
        d_Rand[globalID + iOut * MT_RNG_COUNT] = ((float)x + 1.0f) / 4294967296.0f;
    }
}
```

Szórás csökkentés

- Végtelen sok minta esetén végtelenül pontos eredményt kapunk.
  - Nem túl praktikus…
- Mitől függ az integrálás pontossága?
  - A minták szórásától:
    - $\tilde{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i \approx E[X]$
    - $V[X] \approx \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \tilde{\mu})^2$
  - Hogyan befolyásolható a szórás?

Szórás csökkentés

- Adaptív mintavételezés
  - Állítsunk felső korlátot a szórásra
  - Iteratívan értékeljük ki a mintákat
    - Számítsuk újra a várható értéket és a szórást
  - Amennyiben a szórás a korlát alá kerül a várható érték jó közelítése az integrálnak!

  - A nehézség a felső korlát meghatározása
  - Kevés minta esetén nem biztos, hogy jó az eredmény! (Egy darab minta szórása nulla!)

Szórás csökkentés

- Rétegzett mintavételezés
  - A minták tartományát csökkentve csökken a szórás
  - Osszuk fel a teljes tartományt kisebb régiókra és függetlenül mintavételezzük!
    - $V[X + Y] = V[X] + V[Y]$
  - Minden régióba kell mintának esnie, így nő a szükséges minták száma!

Szórás csökkentés

- Fontosság szerinti mintavétel
  - Egy minta hozzájárulása: $\frac{f}{p}$, f a minta értéke és p a mintához tartozó valószínűség
  - Egyenletes mintavételezés esetén egy kis valószínűségű minta „elrontja” az átlagot.
  - Válasszuk a minták valószínűségét az értékükkel arányosan!
    - Az eloszlás „hasonlítson” az integrandusra
    - Valószínűségi súlyozást alkalmazzunk

Sztochasztikus differenciál egyenlet

- Differenciál egyenlet
  - Az ismeretlen deriváltjai is megjelennek benne

Bessel féle differenciál egyenlet

$$x^2 \frac{\mathrm{d}^2 y}{\mathrm{d}x^2} + x \frac{\mathrm{d}y}{\mathrm{d}x} + (x^2 - \alpha^2) y = 0$$

Black Scholes differenciál egyenlet

$$\mathrm{d}S_t = \mu S_t \mathrm{d}t + \sigma S_t \mathrm{d}W_t$$

Sztochasztikus differenciál egyenlet

- Black-Scholes egyenlet
  - Részvény ár változás

$$dS_t = \mu S_t dt + \sigma S_t dW_t$$

  - S_t: a részvény t időpontbeli ára
  - μ :a sztochasztikus folyamat átlagának változása (stochastic drift)
  - σ : az ár változási valószínűsége (volatility)
  - W :Wiener féle sztochasztikus folyamat (Brown mozgás)

Sztochasztikus differenciál egyenlet

- Monte Carlo szimuláció
  - Egymástól független trajektóriák számítása
  - Várható érték számítás

$$E[S(t)] = \frac{\sum_{i=1}^{N} S_i(t)}{N}$$

  - Szórás számítás

$$\sigma(t) = \sqrt{E(S(t))^2 - E(S(t)^2)}$$

Sztochasztikus differenciál egyenlet