Aktuelle Probleme der experimentellen Teilchenphysik (Modul P23.1.2b)

Statistical Methods of Data Analysis

Ulrich Husemann

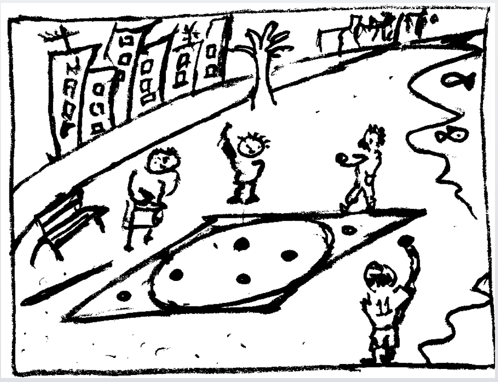
Humboldt-Universität zu Berlin
Winter Semester 2010/2011

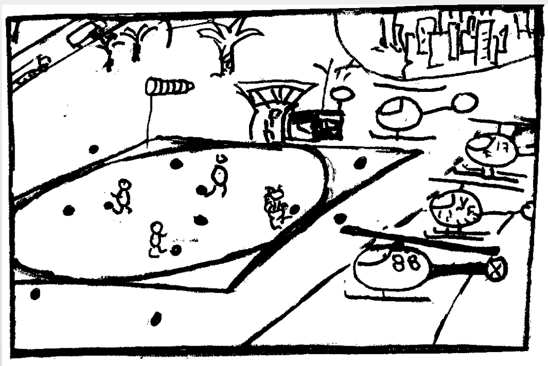
4 Monte Carlo Methods

- 4.1 Introduction
- 4.2 Random number generators
  - real random numbers
  - pseudo random numbers
  - quasi random numbers
- 4.3 Random numbers for arbitrary PDFs
  - Transformation method
  - Rejection sampling (Hit-or-miss method)
  - Integration
  - Reweighting

Monte Carlo Methods

- Introduction To Monte Carlo Algorithms
  - W. Krauth, arXiv:cond-mat/9612186v2
- Example: Monte Carlo Determination of π



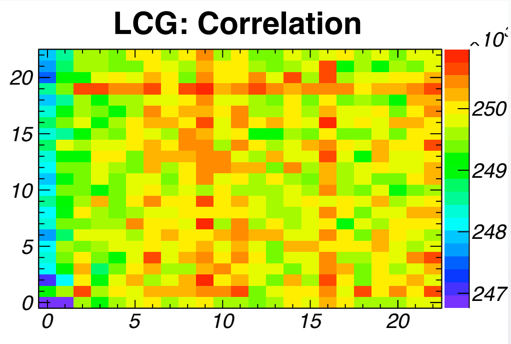


Random Number Generators

$$r_{i+i} = (r_i \cdot a + c) \mod m$$

Linear Congruent Generator
as implemented in ROOT TRandom class

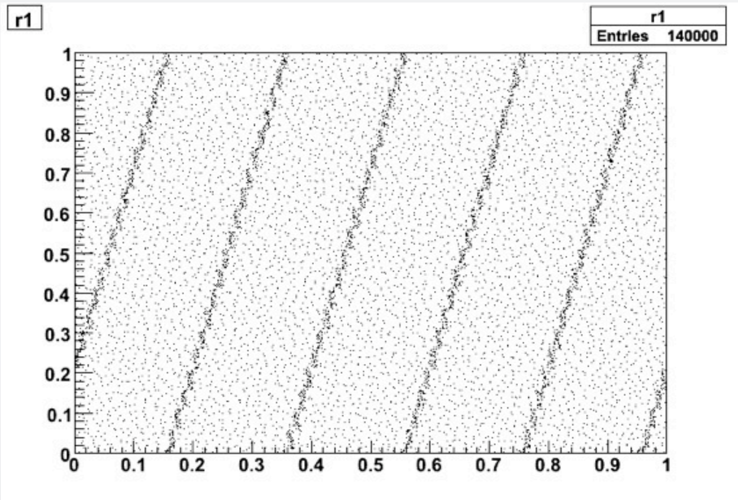
```
}

//______________________________________________________________________________
Double_t TRandom::Rndm(Int_t)
{
//  Machine independent random number generator.
//  Based on the BSD Unix (Rand) Linear congrential generator
//  Produces uniformly-distributed floating points between 0 and 1.
//  Identical sequence on all machines of >= 32 bits.
//  Periodicity = 2**31
//  generates a number in ]0,1]
//  Note that this is a generator which is known to have defects
//  (the lower random bits are correlated) and therefore should NOT be
//  used in any statistical study.

#ifdef OLD_TRANDOM_IMPL
   const Double_t kCONS = 4.6566128730774E-10;
   const Int_t kMASK24  = 2147483392;

   fSeed *= 69069;
   UInt_t jy = (fSeed&kMASK24); // Set lower 8 bits to zero to assure exact float
   if (jy) return kCONS*jy;
   return Rndm();
#endif

   const Double_t kCONS = 4.6566128730774E-10; // (1/pow(2,31))
   fSeed = (1103515245 * fSeed + 12345) & 0x7fffffffUL;

   if (fSeed) return  kCONS*fSeed;
   return Rndm();
}

//______________________________________________________________________________
```



Correlation between floating point representations of mantisse (23 least significant bits) of two consequent random numbers

Random Number Generators

- Problems of Linear Congruent Generator

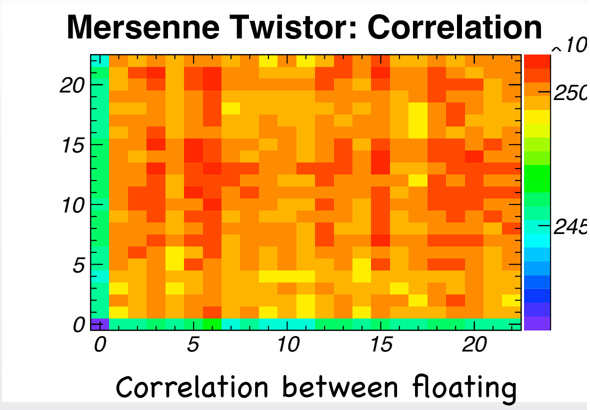


r0 = 4711,
a = 205,
c = 29573,
m = 139968

Random Number Generators

Mersenne Twistor
as implemented in ROOT TRandom3 class

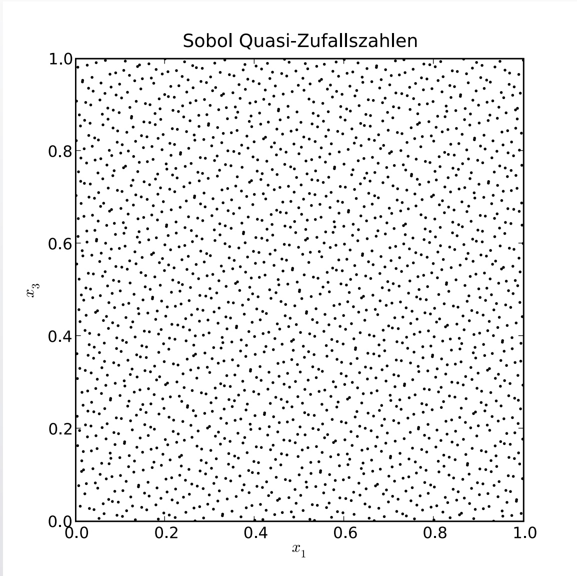
```
//___________________________________________________________________________
Double_t TRandom3::Rndm(Int_t)
{
//  Machine independent random number generator.
//  Produces uniformly-distributed floating points in ]0,1]
//  Method: Mersenne Twistor

   UInt_t y;

   const Int_t  kM = 397;
   const Int_t  kN = 624;
   const UInt_t kTemperingMaskB =  0x9d2c5680;
   const UInt_t kTemperingMaskC =  0xefc60000;
   const UInt_t kUpperMask =       0x80000000;
   const UInt_t kLowerMask =       0x7fffffff;
   const UInt_t kMatrixA =         0x9908b0df;

   if (fCount624 >= kN) {
      register Int_t i;

      for (i=0; i < kN-kM; i++) {
         y = (fMt[i] & kUpperMask) | (fMt[i+1] & kLowerMask);
         fMt[i] = fMt[i+kM] ^ (y >> 1) ^ ((y & 0x1) ? kMatrixA : 0x0);
      }

      for (   ; i < kN-1    ; i++) {
         y = (fMt[i] & kUpperMask) | (fMt[i+1] & kLowerMask);
         fMt[i] = fMt[i+kM-kN] ^ (y >> 1) ^ ((y & 0x1) ? kMatrixA : 0x0);
      }

      y = (fMt[kN-1] & kUpperMask) | (fMt[0] & kLowerMask);
      fMt[kN-1] = fMt[kM-1] ^ (y >> 1) ^ ((y & 0x1) ? kMatrixA : 0x0);
      fCount624 = 0;
   }

   y = fMt[fCount624++];
   y ^=  (y >> 11);
   y ^= ((y << 7 ) & kTemperingMaskB );
   y ^= ((y << 15) & kTemperingMaskC );
   y ^=  (y >> 18);

   if (y) return ( (Double_t) y * 2.3283064365386963e-10); // * Power(2,-32)
   return Rndm();
}
```



Correlation between floating point representations of mantisse (23 least significant bits) of two consequent random numbers

Quasi random number

- Example: Sobol sequence
  - Better uniformity
  - Faster integration
  - Not useful Monte Carlo event generation



Algorithms

Tabelle 5.1: Erzeugungsalgorithmen von Zufallszahlen einiger wichtiger Wahrscheinlichkeitsverteilungen aus gleichverteilten Zufallszahlen z in $[0,1[$.

Wahrscheinlichkeitsdichte	Wertebereich	Algorithmus
$f(x) = \frac{1}{b-a}$	$[a,b[$	$x = (b-a)\cdot z + a$
$f(x) = 2x$	$[0,1[$	$x = \max(z_1, z_2)$ or $x = \sqrt{z}$
$f(x) \sim x^{r-1}$	$[a,b[$	$x = [(b^r - a^r)\cdot z + a^r]^{1/r}$
$f(x) \sim \frac{1}{x}$	$[a,b[$	$a \cdot (b/a)^z$
$f(x) = \frac{1}{x^2}$	$]1,\infty]$	$x = 1/z$
$f(x) = \frac{1}{k} e^{-x/k}$	$]0,\infty]$	$x = -k \ln z$
$f(x) = x e^{-x}$	$]0,\infty]$	$x = -\ln(z_1 \cdot z_2)$
$f(x) = -\ln x$	$[0,1[$	$x = z_1 \cdot z_2$
Gauss: $f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{x^2}{2\sigma^2}}$	$[-\infty,\infty]$	$x = \sigma\sqrt{-\ln z_1^2}\cdot\cos(2\pi z_2)$
Breit–Wigner: $f(x) = \frac{\Gamma}{2\pi}\cdot\frac{1}{(x-\mu)^2 + (\Gamma/2)^2}$	$[-\infty,\infty]$	$x = [\tan\pi(z-0.5)]\cdot\Gamma/2 + \mu$

[H. Kolanoski, Stat. Methoden der Datenanalyse, SS08]

Monte-Carlo Integral

- Example: $f(x) = -x^4 + x^2 + 1 \to \int_{-1}^{1} \mathrm{d}x\, f(x) = \frac{34}{15}$
- Result from 10000 pairs of random numbers: 2.2705
  → relative deviation: 1.7×10^{-3}

$$\varepsilon = \frac{N_{\mathrm{hit}}}{N} = 0.907 \to \sigma(\varepsilon) = \sqrt{\frac{\varepsilon(1-\varepsilon)}{N}} = 2.9 \times 10^{-3}$$