Aplikačný SW

Short codes and compiling

Intro

- Cieľom je ukázať si jednoduché príklady makra
- Využitý bude c++, ale cieľom nie je vysvetľovať c++
- Na webe nájdete viacero dobrých kurzov (napr. „C++ in 21 days“)
- Využitý bude ROOT, ale primárnym cieľom nie je vysvetľovať ROOT. Chceme iba demonštrovať jeho využitie
- bližšie info, návody a príklady viď.: http://root.cern.ch

Kompilácia c++ kódu

- Editácia kódu v ľubovoľnom plain-text editore
- Následná kompilácia cez g++ zdroj.cpp
- Bez použitia ďalších parametrov sa vytvorí súbor a.out, ktorý sa spustí ako ./a.out
- Parametre g++
  - -v informácia o verzii
  - -o output.bin definuje výstupný súbor

Jednoduchý Hello World

Príklad Hello World

```
#include <iostream>
#include <stdlib.h>
using namespace std;


int main ()
{
  cout << "Hello World" << endl;
  return 0;
}
```

Hello World s volaním funkcie

```
#include <iostream>
#include <stdlib.h>
using namespace std;


/*
Ukážka funkcie Hello world
*/
void PrintHelloWorld()
{
    cout << "Hello World" << endl;
}
int main ()
{
    PrintHelloWorld();
    return 0;
}
```

Nezabúdať na komentáre.
Zaviesť si štruktúrovanie kódu.

Komentár

Definícia funkcie

Príkaz sa ukončuje bodkočiarkou.

Volanie funkcie

Implementácia výpočtu

```
#include <iostream>
#include <stdlib.h>
using namespace std;

void PrintDelenie()
{
     cout << „Podiel 5/7 je " << 5/7 << endl;
}

int main ()
{
     PrintDelenie();
     return 0;
}
```

Výsledok je nula. Prečo?
Ako musíme zmeniť kód aby sa nám vypísal výsledok v podobe desatinného čísla?

Implementácia výpočtu II.

```
#include <iostream>
#include <stdlib.h>
using namespace std;

void PrintDelenie()
{
    cout << „Podiel 5/7 je " << (double)5/7 << endl;
}

int main ()
{
    PrintDelenie();
    return 0;
}
```

Implementácia výpočtu III.

```
#include <iostream>
#include <stdlib.h>
using namespace std;

void PrintDelenie(int x, int y)
{
    cout<<"Podiel "<<x<<"/"<<y<<" je " << (double) x/y << endl;
}

int main()
{
  PrintDelenie(5,7);
  return 0;
}
```

Deklarácia premenných

Implementácia výpočtu IV.

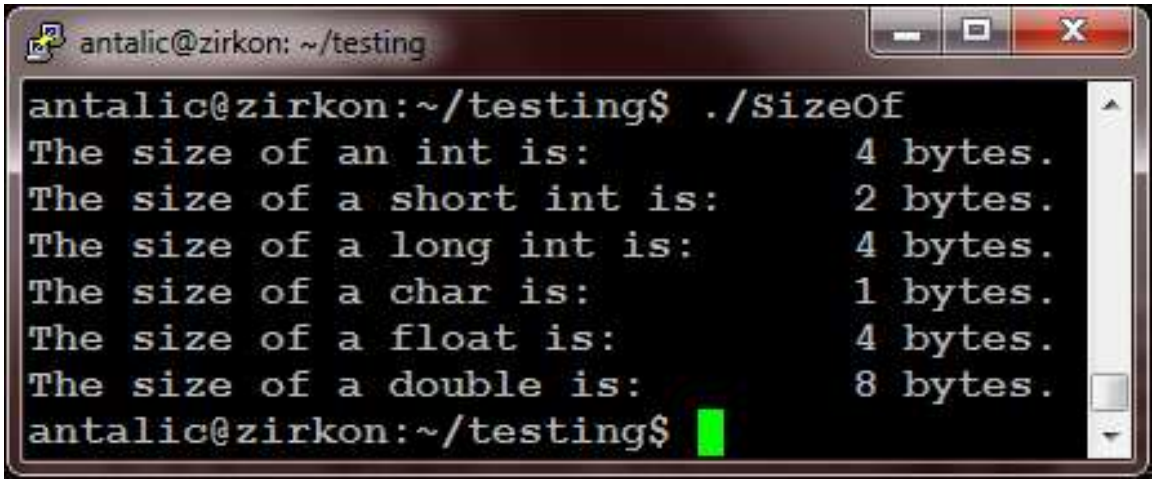
```
#include <iostream>
#include <stdlib.h>
using namespace std;

void PrintDelenie(int x, int y)
{
    cout<<"Podiel "<<x<<"/"<<y<<" je " << (double) x/y << endl;
}


int main()
{
  PrintDelenie(5.5,7);
  return 0;
}
```

Pozor na chybu! Aj napriek zadaniu desatinného čísla, je brané celé číslo.

Pozor – kompilátor môže, ale nemusí dať varovanie.

Rôzne typy premenných

```
#include <iostream>
#include <stdlib.h>
using namespace std;

int main()
{
  cout<<"The size of an int is:\t\t„ << sizeof(int)<<" bytes.\n";
  cout<<"The size of a short int is:\t" << sizeof(short)<< " bytes.\n";
  cout<<"The size of a long int is:\t„ << sizeof(long) << " bytes.\n";
  cout<<"The size of a char is:\t\t„ << sizeof(char) << " bytes.\n";
  cout<<"The size of a float is:\t\t„ << sizeof(float) << " bytes.\n";
  cout<<"The size of a double is:\t„ << sizeof(double) << " bytes.\n";
  return 0;
}
```



Návrat hodnoty z funkcie

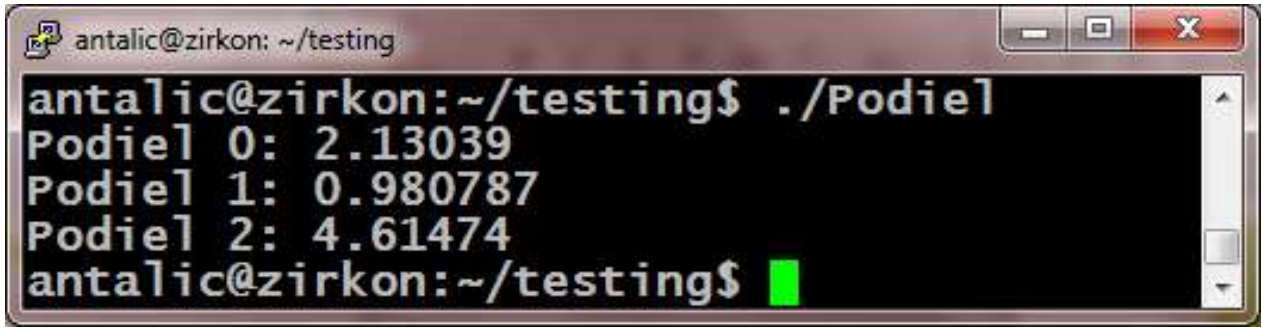
```
#include <iostream>
#include <stdlib.h>
using namespace std;

double PrintDelenie(double x, double y)
{
   return (double) x/y;
}


int main()
{
  cout << PrintDelenie(5.5,6)<<endl;
  return 0;
}
```

Tentokrát musí byť funkcia definovaná ako double, pretože už dáva výstup.

Pozor na deklaráciu typu funkcie a na typy premenných

Podiel náhodných čísel

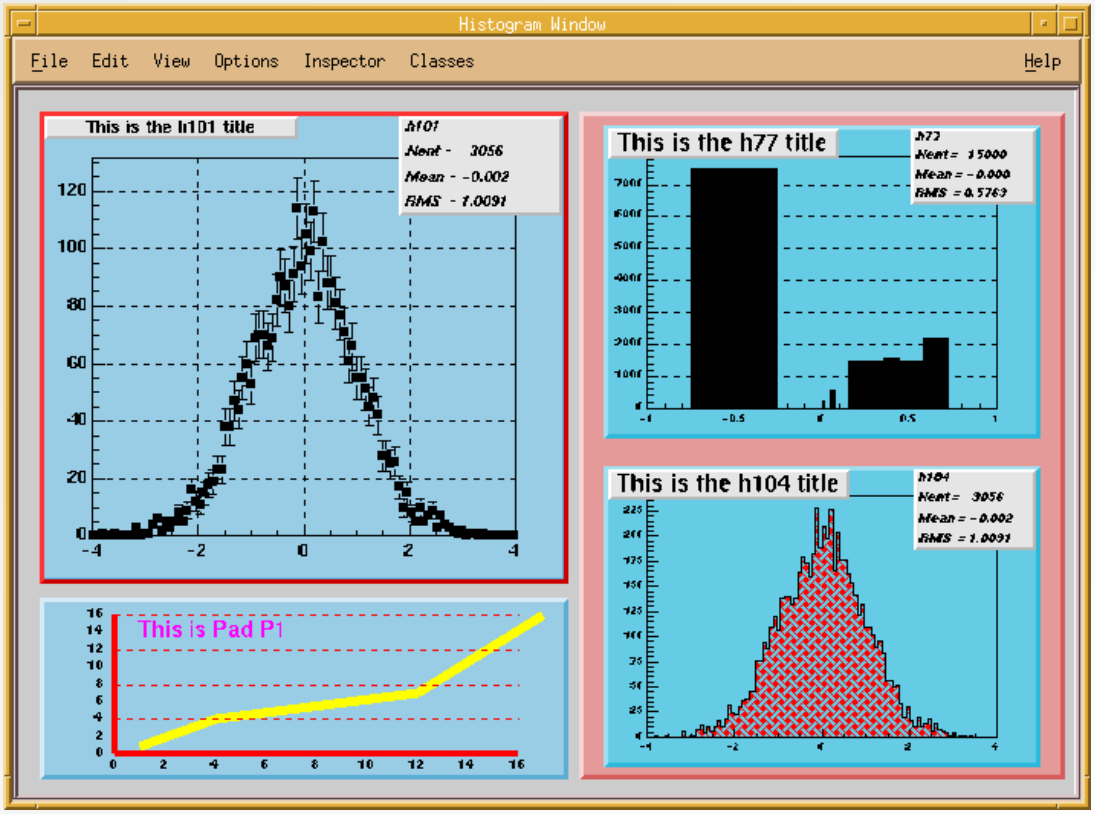
```
#include <iostream>
#include <stdlib.h>
using namespace std;

double PodielRand() {
        double U1=rand()/(double)RAND_MAX;
        double U2=rand()/(double)RAND_MAX;
   return U1/U2;
}


int main()
{
  for(int i=0;i<3;i++)
    cout << “Podiel “<<i<<“: “<<PodielRand<<endl;
  return 0;
}
```



Canvas

Canvas je oblasť v aktuálnom okne zobrazujúca histogramy. Každý canvas môže byť delený do viacerých oblastí – pad.

TCanvas *c1 = new Tcanvas("c1",„New Canvas",800,1000); //def. Canvasu



Príklad z ROOT Reference Guide (root.cern.ch/root/html/index.html)

Histogramy v ROOTe

Histogram má definované biny. Očakávajú sa tri číselné premenné pre každú os – počet binov, začiatok a koniec rozsahu pre os.

ROOT poskytuje 1d, 2d aj 3d histogramy

```
TH1D *h1 = new TH1D("h1", "histogram title", 100, 0., 1.);
TH2D *h2 = new TH2D("h2", "histogram title", 100, 0., 10., 10, 0., 1.);
TH3D *h3 = new TH3D("h3", histogram title", 100, 0., 10., 10, 0., 1., 10, 0.,1.);
```

Základy práce s histogramom

Plnenie histogramu

h1a->Fill(value); // plnenie histogramu

Vykreslenie histogramov
h1a->Draw();

Prekrytie histogramov
h1b->Draw("same");

Fitovanie napr. Gauss funkciou

H1->Fit(“gaus”);

Grafy v ROOTe

Graf nemá definované biny. Plnia sa jednotlivé body – každé s vlastným číselným indexom (zvyčajne poradovým číslom).

Vytvorenie grafu:

TGraph *GraphXY=new TGraph(n);

Kde *n* definuje rozsah bodov (počet)

Plnenie grafu:

GraphXY->SetPoint(i,x,y);

Kde *i* je index bodu a x,y sú súradnice bodov.

Implementácia v ROOTe

```
#include <iostream>
#include <stdlib.h>
void PodielRand()                  // Názov totožný s menom makra
{
  srand((unsigned)time(NULL));              // nastavý „seed“ pre náhodné čísla
  for (int i=0;i<3;i++)                     // generuje sa 100 čísel
  {
    double U1= rand()/(double)RAND_MAX;
    double U2= rand()/(double)RAND_MAX;
    cout << “Podiel “<<i<<“: “ << U1/U2 <<endl;
  }
}
```

Najjednoduchší test:

Root pustíme z adresára, kde máme napísané makro

Vykonanie makra: .x PodielRand.cpp

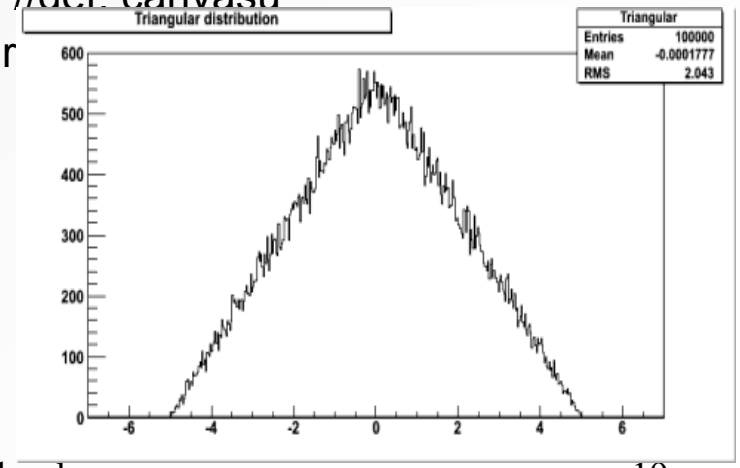
Implementácia s histogramom

```
#include <iostream>
#include <stdlib.h>
void TriangleDistROOT(){                // Názov totožný s menom makra
  TH1D *Triangle = new TH1D("Triangle","Triangle Dist",100,0,1);
  double TriangleRand;
  srand((unsigned)time(NULL));          // nastavý „seed“ pre náhodné čísla
  for (int i=0;i<100;i++) {             // generuje sa 100 čísel
    TriangleRand = TriangleRandomNumber();
    cout << TriangleRand << endl;       // iba kontrolný výpis
    Triangle->Fill(TriangleRand);       // plnenie histogramu
  }
  TCanvas *c1 = new Tcanvas("c1","Triangle",800,1000); //def. canvasu
  Triangle->Draw();                     // vykreslenie histogramu
}

double TriangleRandomNumber() {
        double U1=(rand()/(double)RAND_MAX);
        double U2=(rand()/(double)RAND_MAX);
        double w=(U1+U2)/2;
        return w;
}
```

Akú distribúciu dostaneme?

Implementácia s histogramom

```
#include <iostream>
#include <stdlib.h>
void TriangleDistROOT(){                // Názov totožný s menom makra
  TH1D *Triangle = new TH1D("Triangle","Triangle Dist",100,0,1);
  double TriangleRand;
  srand((unsigned)time(NULL));          // nastavý „seed“ pre náhodné čísla
  for (int i=0;i<100;i++) {             // generuje sa 100 čísel
    TriangleRand = TriangleRandomNumber();
    cout << TriangleRand << endl;       // iba kontrolný výpis
    Triangle->Fill(TriangleRand);       // plnenie histogramu
  }
  TCanvas *c1 = new Tcanvas("c1","Triangle",800,1000); //def. canvasu
  Triangle->Draw();                     // vykreslenie histogr
}

double TriangleRandomNumber() {
        double U1=(rand()/(double)RAND_MAX);
        double U2=(rand()/(double)RAND_MAX);
        double w=(U1+U2)/2;
        return w;
}
```



Nastavenie rozsahu histogramu

Upraviť v makre rozsah osí.

Histogram->GetYaxis->SetRangeUser(2,50);

```
Void PriemerRand(){
        TH1D *Priemer= new TH1D(„Priemer",„PriemerRand",100,0,1);
…
        Priemer->Fill(RandAvg);     // plnenie histogramu
…
        TCanvas *c1 = new Tcanvas("c1", „Priemer",800,1000); //def. Canvasu
        Priemer->GetYaxis()->SerRangeUser(2,50);
        Priemer>Draw();                 // vykreslenie histogramu
…
}
```

Trees v ROOTe

Záznamy, v ktorých sa môžu ukladať jednotlivé sady premenných, napríklad eventy obsahujúce signály z rôznych detektorov, časové údaje alebo logické bity.

Tieto záznamy potom tvoria štruktúru, ktorú môžeme spracovávať po jednotlivých eventoch v analýze, alebo uložiť na disk pre neskoršie spracovanie.

Dôležitou vlastnosťou tree je možnosť filtrovania eventov podľa zvolených kritérií.

Filtrovanie tree bez implementácie podmienok do kódu

tree->Draw("x>>meno_hist(Nbins,x_low,x_high)","y<7 && y>3);

Vytvorenie a naplnenie tree

Vytvorenie tree a nacitanie hodnôt z text súboru.

```
ifstream in1;
in1.open("input_tree.txt");
Double_t x;
Double_t y;
TTree *MyTree = new TTree("MyTree","Tree with two branches");
MyTree->Branch("x",&x,"xvar/D");
MyTree->Branch("y",&y,"yvar/D");
while(1)
{
    in1 >> x >> y;
    MyTree->Fill();
    if (!in1.good()) break;
}
```

Otvorenie súboru

Vytvorené tree a branch(e) pre premenné x a y.
/D v treťom parametri označuje typ premennej (double)

Načítanie premenných

Plnenie tree

Plnenie hodnôt z tree do spektier

Vytvorenie grafov, histogramov, načítanie veľkosti tree a plnenie hodnôt

```
Int_t nentries = (Int_t)MyTree->GetEntries();
TH1D *HistX=new TH1D("HistX","X variables",10,0,10);
TH2D *HistXY=new TH2D("HistXY","XY 2D plot",10,0,10,10,0,10);
TGraph *GraphXY=new TGraph(nentries);
    for (Int_t i=0;i<nentries;i++)
    {
          MyTree->GetEntry(i);
          HistX->Fill(x);
          GraphXY->SetPoint(i,x,y);
          HistXY->Fill(x,y);
    }
```

Získanie veľkosti tree

Vytvorenie histogramov a grafu

Načítanie i-tej položky

Plnenie histogramov a grafu

Uloženie tree do suboru a filter

Vytvorenie súboru a zápis tree

```
TFile *MyTreeFile = new TFile("MyTreefile.root","recreate");
tree->Write();
```

Vykreslenie premenných z tree (dá sa aj bez deklarovania histogramu)

```
MyTree->Draw("x>>Hist(100,0,10)");
```

Vykreslenie premenných z tree s definovanám filtra

```
MyTree->Draw("x>>Hist(100,0,10)","y>3.0 && y<4.0");
```

Vypísanie informácie a štatistiky o tree

```
MyTree->Print();
```

Poissonovo rozdelenie

Poissonovo rozdelenie udáva pravdepodobnosť nájdenia práve k javov v danom intervale (napr. priestore a čase) nezávisle od seba.

Aproximácia binomického rozdelenia, pre veľký počet pokusov (zvyčajne ak $n>30$ a $p<0.1$).

Toto rozdelenie sa používa na popis experimentálnych hodnôt, ktoré reprezentujú počet udalostí za jednotku času (napr. počet rádioaktívnych rozpadov, počet zákazníkov v obchode, počet áut prechádzajúcich cez úsek diaľnice a pod.)

Poissonovo - vyjadrenie

Hustota pravdepodobnosti tohto rozdelenia je

$$f(k;\mu) = \frac{\mu^k e^{-\mu}}{k!} \quad k = 0,1,2,3... \quad \mu > 0$$

Ak máme n pokusov s pravdepodob nosťou p tak stredný počet realizovaných prípadov je $\mu = np$.

Funkcia pravdepodobnosti nám definuje pravdepodobnosť, že sa pri danom μ realizuje práve hodnota k

Stredná hodnota $\bar{k} = \mu$

Disperzia $\sigma^2 = \mu$

Príklad jazdeckej jednotky pruskej armády

Prvá reálna aplikácia Poissonovho rozdelenia v reálnej situácii (Ladislaus Bortkiewicz, 1898)
Za 20 rokov bolo v pruskej armade v jednotke s 10 jazdcami 122 ľudí ukopaných koňmi k smrti. V niektoré roky nenastali žiadne úmrtia, ale vyskytol sa aj rok, keď zahynuli štyria. Otázka je, či je proces čisto náhodný a ako opísať pravdepodonosť, že niekto zahynie.

To je ekvivalentné, akoby sme sledovali vzorku 200 členov zaradených do jednotky počas jedného roku (pričom mŕtvy člen sa vždy nahradí novým). Teda 200 vojako-rokov.
Na koľkých pozíciách príde ku koľkým úmrtiam?

Príklad jazdeckej jednotky pruskej armády

$$f(k;\mu) = \frac{\mu^k e^{-\mu}}{k!} \quad k = 0,1,2,3... \quad \mu > 0$$

Priemerná hodnota (úmrtia na jednej pozícii)
μ = 122/200 = 0,61 úmrtia na jedného vojaka na rok.

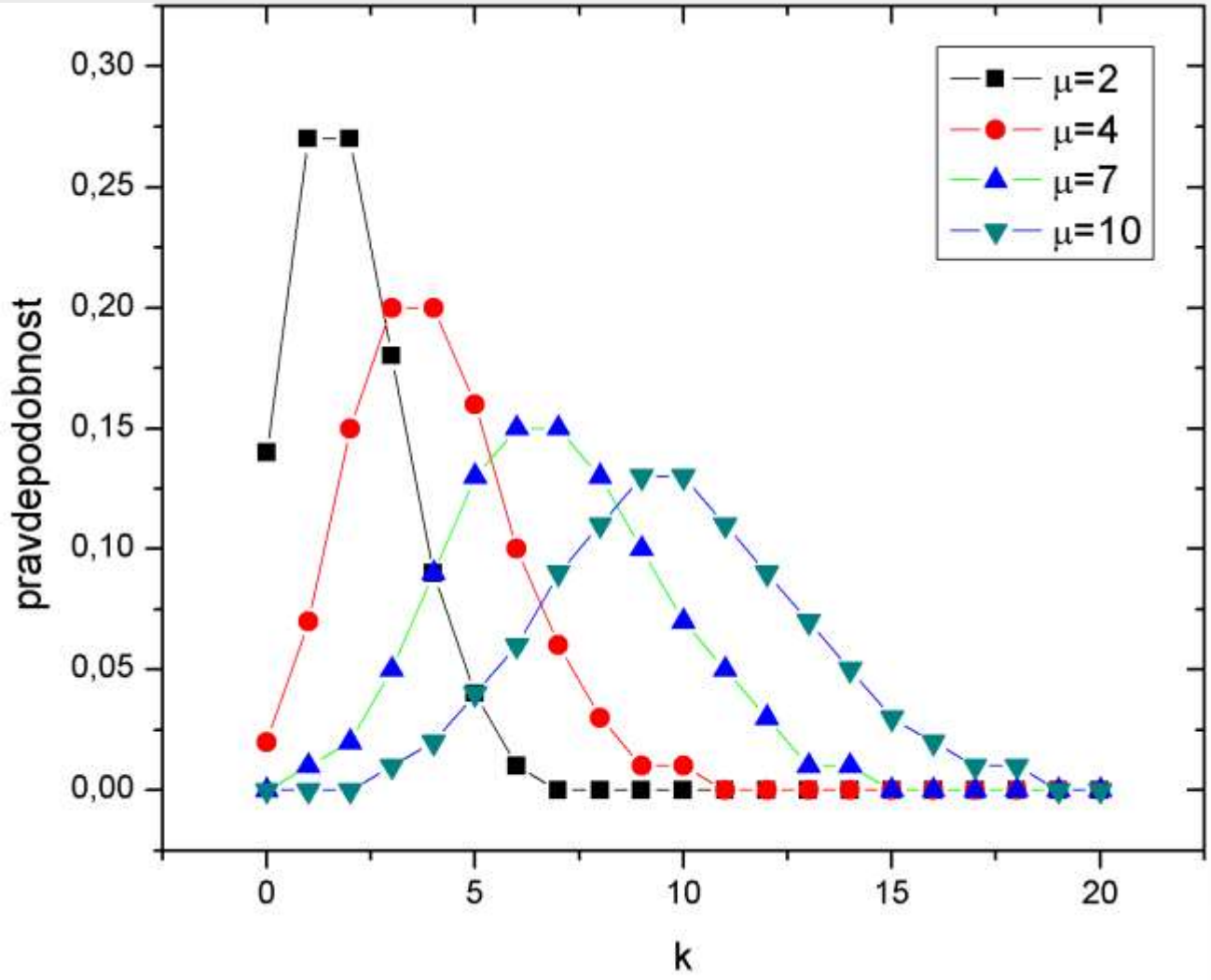
Príklad jazdeckej jednotky pruskej armády

$$f(k;\mu) = \frac{\mu^k e^{-\mu}}{k!} \quad k = 0,1,2,3... \quad \mu > 0$$

Priemerná hodnota (úmrtia na jednej pozícii)
μ = 122/200 = 0,61 úmrtia na jedného vojaka na rok.

# úmrtí /vojaka /rok	Pravdepodobnosť	Skutočných úmrtí	Odhadnutých úmrtí
0	0,543508691	109	108,6701738149
1	0,3314440301	65	66,2888060271
2	0,1010904292	22	20,2180858383
3	0,0205550539	3	4,1110107871
4	0,0031346457	1	0,626929145
5	0,0003824268	0	0,0764853557

Poisson pre jednotlivé prípady



Kumultívna distribučná funkcia

Zoberieme si sumu jednotlivých členov – tzv. kumulatívnu distribučnú funkciu

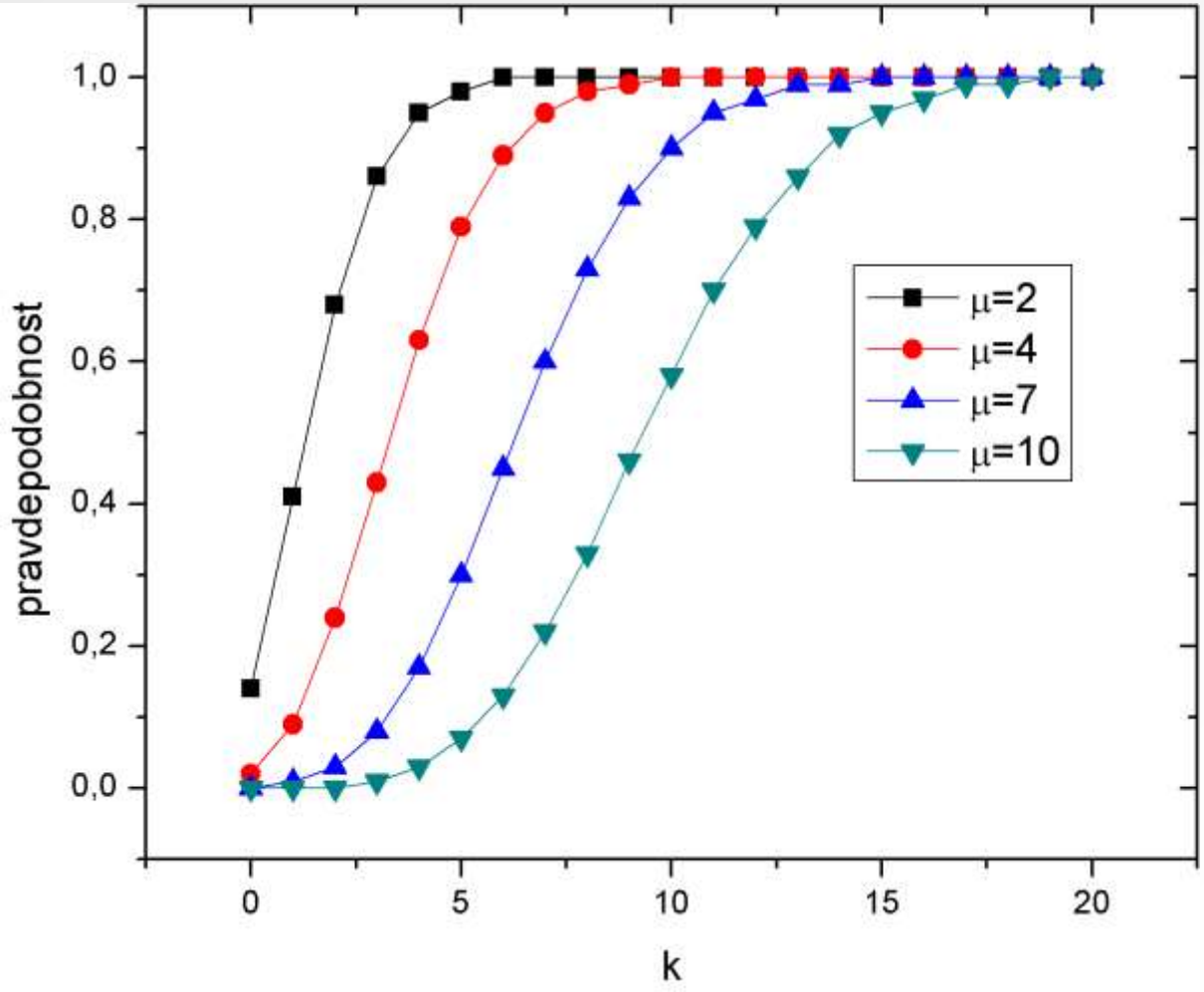
$$D(k;\mu)=\sum_{k=0}^{+\infty}\frac{\mu^k e^{-\mu}}{k!}=e^{-\mu}\sum_{k=0}^{+\infty}\frac{\mu^k}{k!}=e^{-\mu}e^{\mu}=1$$

Este sa pozrieme na pomer dvoch susednych clenov

$$\frac{f(k+1;\mu)}{f(k;\mu)}=\frac{\dfrac{\mu^{k+1}e^{-\mu}}{(k+1)!}}{\dfrac{\mu^k e^{-\mu}}{k!}}=\frac{\mu}{(k+1)}$$

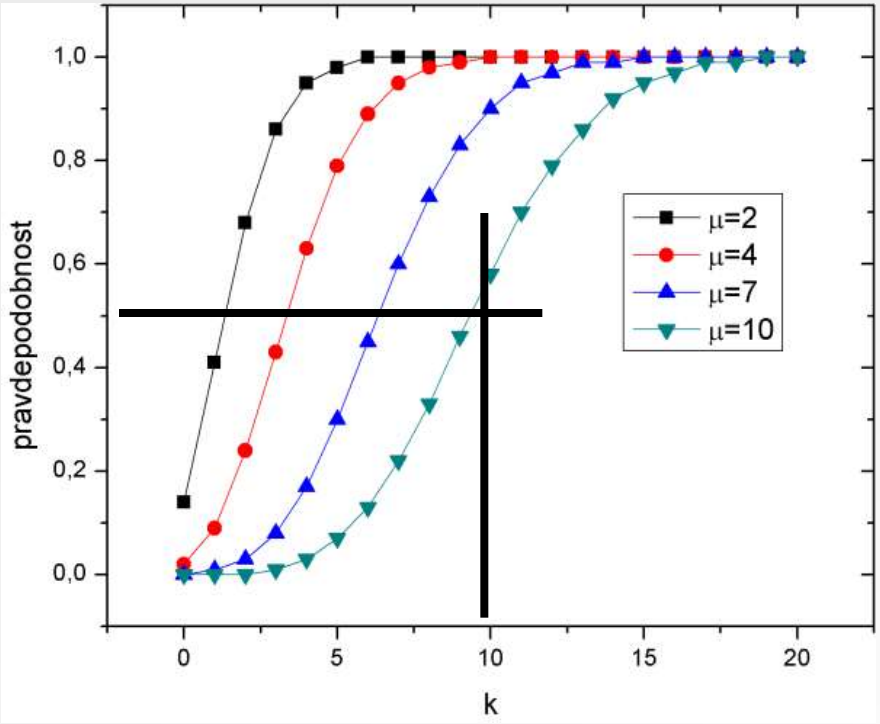
Poisson pre jednotlivé prípady

Kumulatívna distribučná funkcia



Generovanie náhodných čísel s Poisson distribúciou

Idea: Vygenerujem číslo s rovnomerným rozdelením od 0 po 1 a označím ho p. Pre distribúciu s daným μ vypočítame kumulatívne funkcie D pre všetky hodnoty k.



Ako náhodné číslo s Poissonovým rozdelením akceptujeme prvé číslo k, pre ktoré je hodnota $D > p$.

Táto tzv. kumulačná metóda je viac-menej využiteľná vždy, keď vieme jednoducho vypočítať kumulatívnu funkciu D.

Optimalizácia generátora

Opakovaný výpočet faktoriálov a exponenciálnych funkcií je časovo náročný. Preto si doratáme člen $f(k+1)$ pomocou členu $f(k)$ a pripočítame k predchádzajúcej sume, pričom $P(0) = e^{-\mu}$

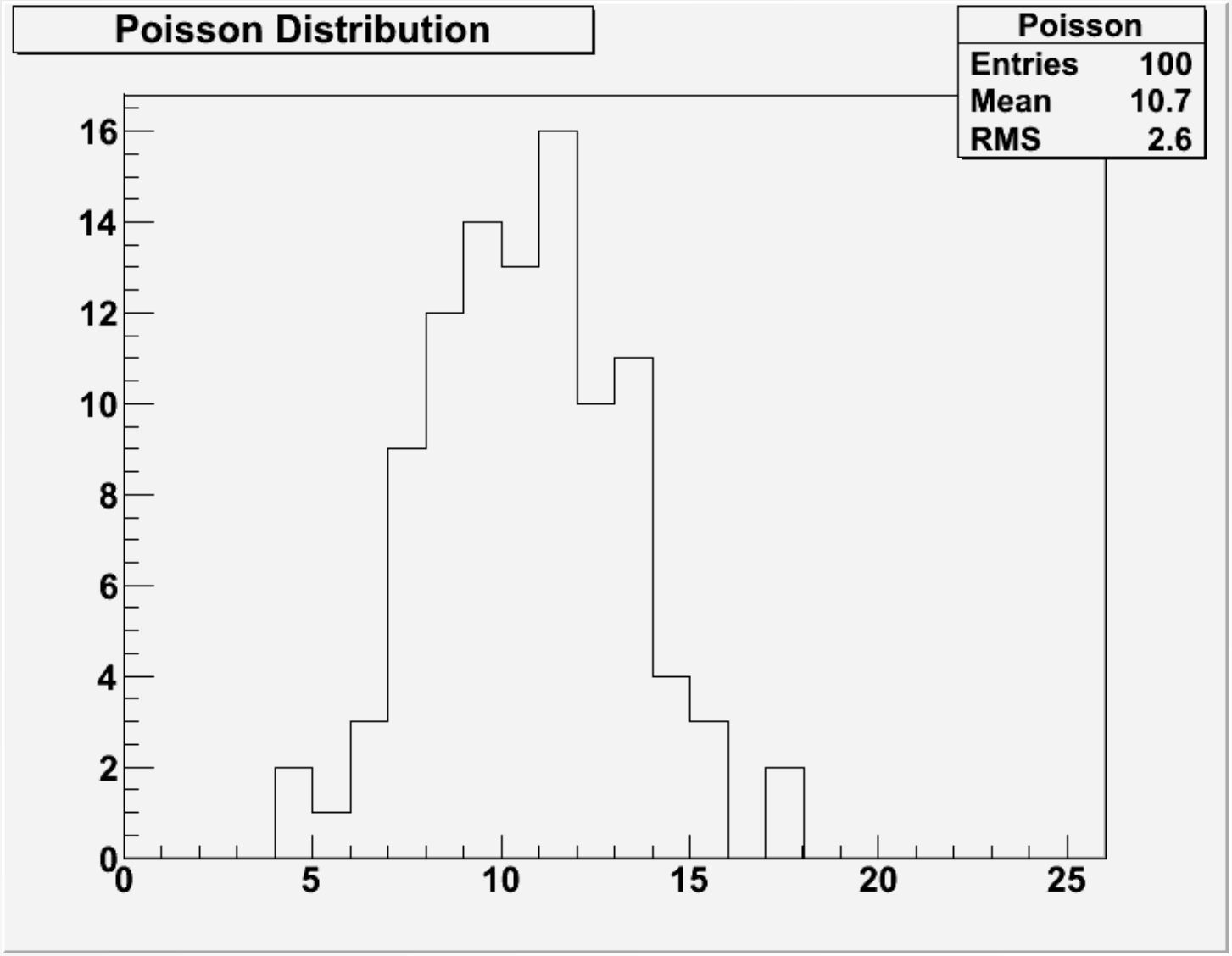
= využijeme

$$\frac{f(k;\mu)}{f(k-1;\mu)} = \frac{\frac{\mu^k e^{-\mu}}{(k)!}}{\frac{\mu^{k-1} e^{-\mu}}{(k-1)!}} = \frac{\mu}{k}$$

Generovanie náhodných čísel s Poisson distribúciou

```
const int PoissonRandomNumber(const double mi)
{
  int k=0;                          //Počítadlo
  const int max_k = 1000;     //horný limit pre k
  double p = rand()/(double)RAND_MAX; //rovnom.rozd.(0,1)
  double P = exp(-mi);      //P(0) - prvý člen kumulatívu
  double sum=P;                     //kumulačná funkcia (ozn.f)
  if (sum>=p) return 0;             //už prvý člen splní podm.
  for (k=1; k<max_k; ++k) {   //Prechádzame cez všetky k
    P*=mi/(double)k;           //kumulat. pre ďalšie k
    sum+=P;                         //Navyšujeme kumulatív
    if (sum>=p) break;              //Stop prekročíme učený limit
  }
  return k;                         //návrat vygenerovaného čísla
}
```

Príklad pre μ=10



Poisson pre špeciálne prípady

Aproximáciou binomického rozdelenia pre $n \to \infty$ s pravdepodobnosťou $p \to 0$.

Pre vysoké μ je podobný gaussovmu rozdeleniu

Gauss rozdelenie

Hustota pravdepodobnosti je definovaná ako

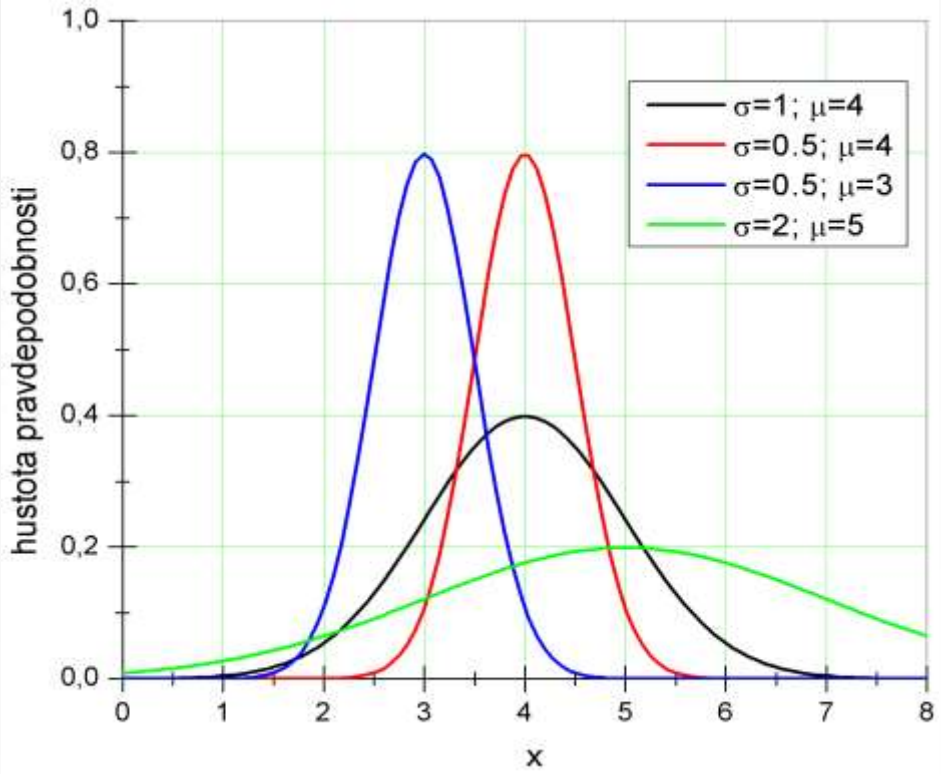
$$f(x;\mu,\sigma) = \frac{1}{\sigma \cdot \sqrt{2\pi}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad \text{podm.} \sigma > 0$$

Stredná hodnota

$$\bar{x} = \mu$$

Disperzia

$$\sigma^2 = \sigma^2$$



Generovanie Gauss rozdelenia

Zvyčajne vychádzajú generátory z náhodného čísla s rovnomerným rozdelením.

Možnosti máme približné a presné.

Presnou metódou by bolo vychádzať z prevrátenej hodnoty kumulatívnej funkcie, ale nemáme vhodné analytické vyjadrenie.

Kumulatívna distribučná funkcia:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{t^2}{2}} dt = \frac{1}{2}\left[1 + erf\left(\frac{x}{\sqrt{2}}\right)\right], x \in \Re$$

Generovanie Gauss rozdelenia

Jednou z možností je aproximácia polynómom – nepresnosti pri hodnotách 4-5 x sigma od strednej hodnoty.

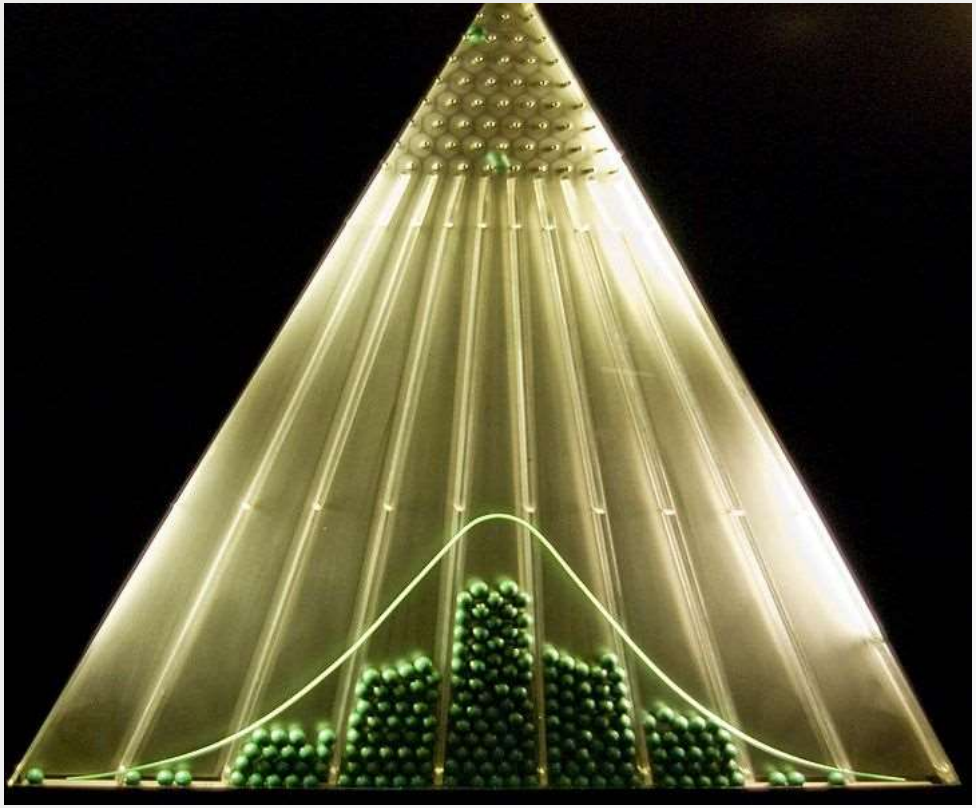
Pomôcť si môžeme centrálnou limitnou vetou

Ak máme postupnosť náhodných premenných Xn, ktoré sú rozdelené podľa rôznych rozdelení pravdepodobnosti, s konečnými strednými hodnotami a disperziami, potom pre dostatočne veľké n získavame približne Gaussovo rozdelenie pre ich sumu:

$$\sum_{i=1}^{n} X_i$$

Dôležitá poznámka: „rozdelené podľa rôznych rozdelení“ treba chápať, že z daných náhodných premenných ani jedna nemusela byť pôvodne rozdelená Gaussovsky.

Galtonova doska



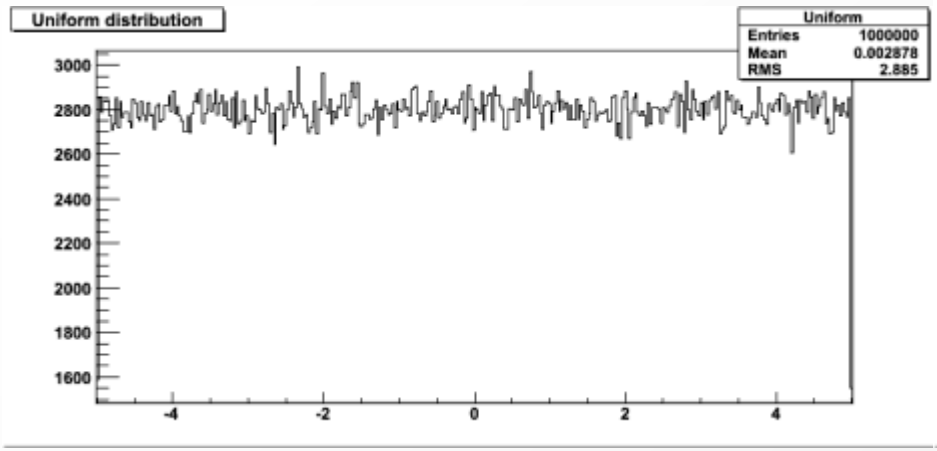
			1			
		1		1		
	1		2		1	
	1		3		3	1
1		4		6		4		1
1	5		10		10		5	1

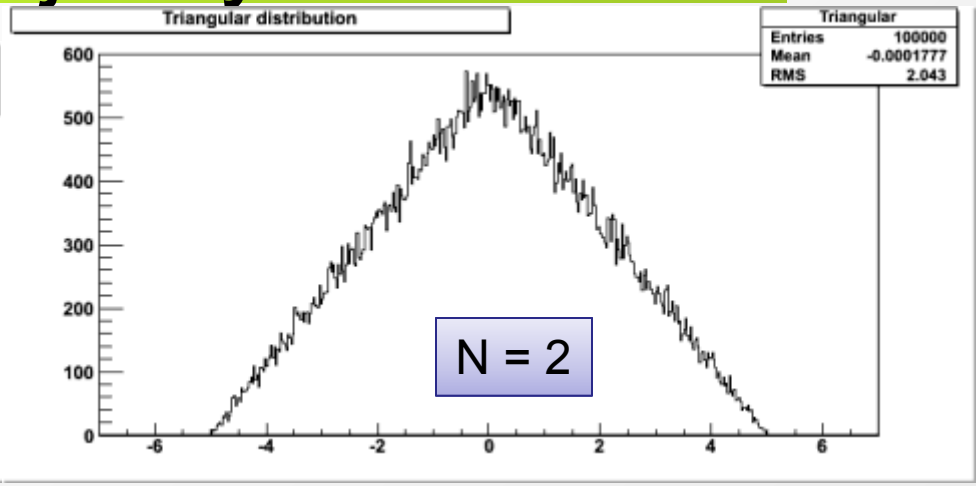
Zariadenie na demonštráciu centrálnej limitnej vety – Sir Francis Galton. V podstate prezentuje, že gauss rozdelenie je aproximované binomickým rozdelením.
Inak povedané, spustené guličky sa prerozdelujú podľa pascalovho trojuholníka určujúceho koeficienty binomického rozdelenia.

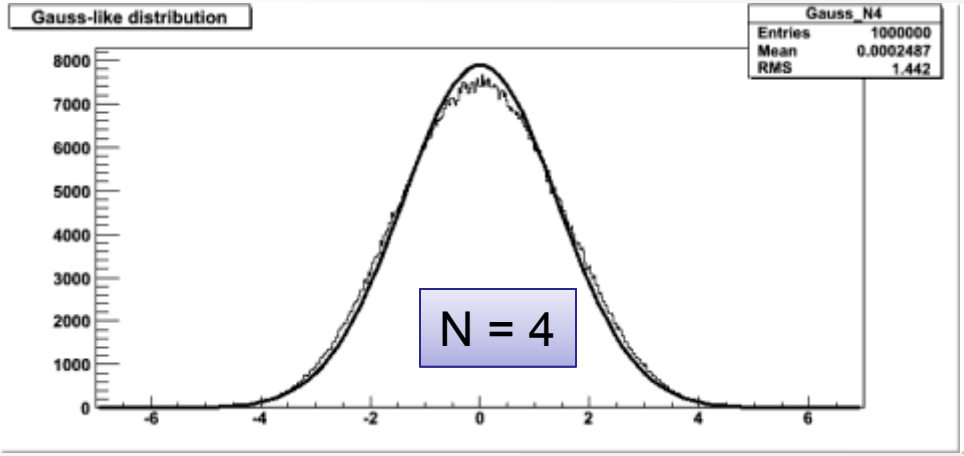
Centrálna limitná veta - implementácia

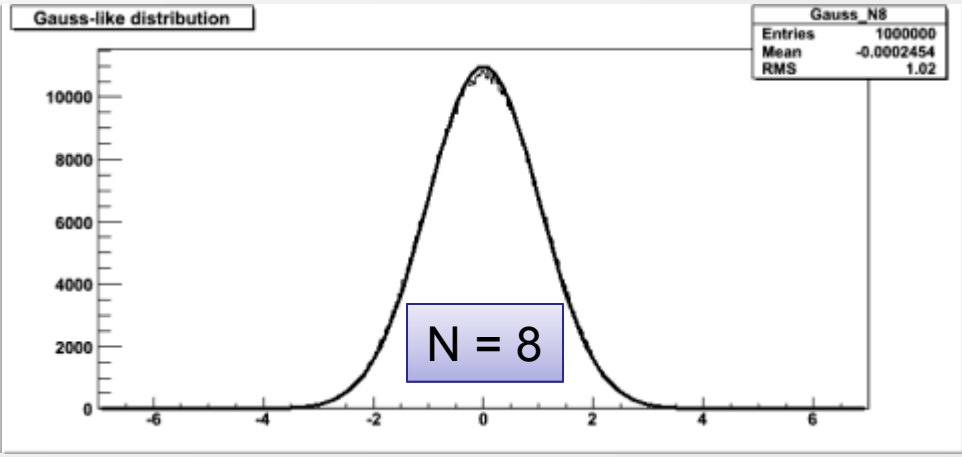
```
Void CentralLimit()
{
        double U1=0;
        int N = 12; // počet iterácií
        for(int i=0;I<N;i++)
                U1=U1+10*rand()/(double)RAND_MAX)-5.0;
        U1=U1/i;
        return U1;
}
```

Praktické overenie centrálnej limitnej vety









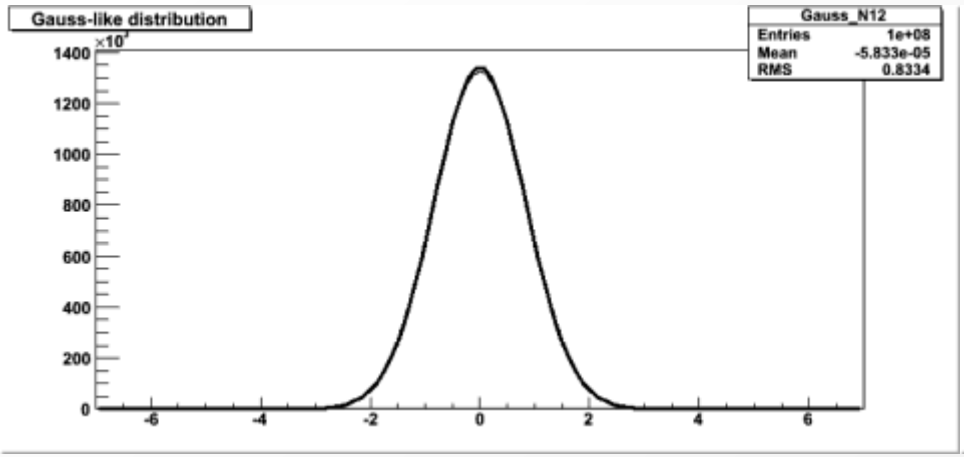
Overenie funkčnosti generátora

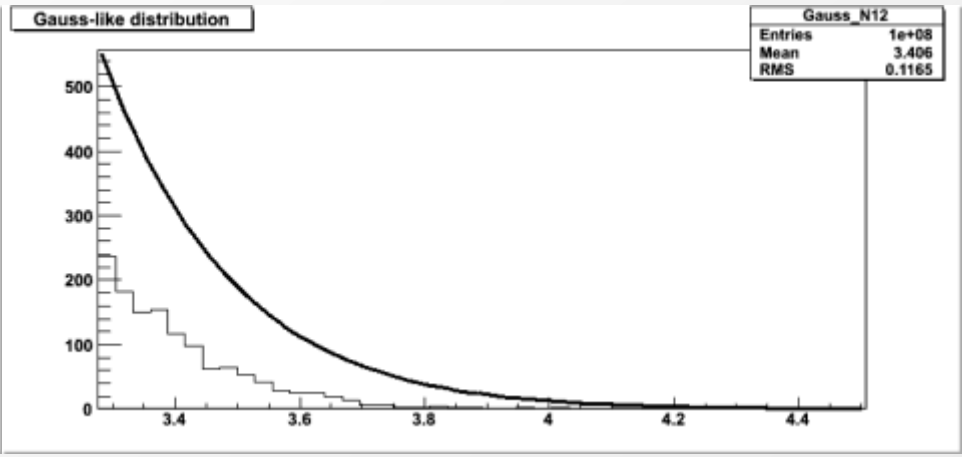
Prefitovanie distribúcie pomocou volania metódy Fit

```
for (int i=0; i<50000; i++) {
                    GaussRandomNumber(k1,k2);
                    Gauss->Fill(k1);
        ….
        }
TCanvas *c1 = new Tcanvas("c1", „Priemer",800,1000); //def. Canvasu
Gauss->Draw();
Gauss->Fit(“gauss”);
```

Centr. lim. veta Pre N = 12

Poznatok 1: Výrazné spomalenie výpočtu
Poznatok 2: Aká je miera presnosti?





Vynikajúci súlad pre celkový tvar distribúcie...

... stále sú však násobné rozdiely pre oblasť $4\times\sigma$, ktoré môžu byť problémom pri simuláciách a opisoch situácií s vysookou citlivosťou.

Upozornenie: Centrálna limitná veta je použiteľná len ak sa nepožaduje skutočná gauss distribúcia.

Box-Muller metóda (1958)

Dve premenné *U1* a *U2* s rovnomerným rozdelením z intervalu (0,1) prevedieme matematickou transformáciou na dve premenné Y1 a Y2 s Gauss rozdelením.

$$Y_1 = \sqrt{-2\ln(U_1)}\cos(2\pi U_2)$$

$$Y_2 = \sqrt{-2\ln(U_1)}\sin(2\pi U_2)$$

+ Veľmi jednoduchá implementácia
- Nevýhody: Numerická nestabilita pre U1 v okolí 0.
- Pomalá metóda – volá sa priveľa funkcií. (Čiastočné riešenie je vrátiť užívateľovi jednu hodnotu a druhú odložiť pre ďalšie volanie funkcie)

Nevýhody sú kritické pri masívnych stochastických výpočtoch.

Box-Muller v polárnych súradniciach

Problémy rieši čiastočne polárna forma Box-Muller transformácie. Originálne už z roku 1968.

Akceptujú sa iba niektoré vygenerované čísla (tzv. rejection methods) - cca $\pi/4$ čísel.

Konkr. po vygenerovaní U1 a U2 z rovnomerného rozdelenia sa zoberie táto dvojica iba keď je súčin kvadrátov $w = U_1^2 + U_2^2$ menší ako 1 ($w < 1$).

Následne sa zavedie transformácia $w \leftarrow \sqrt{\left(-2 \times ln(w)\right)/w}$ a

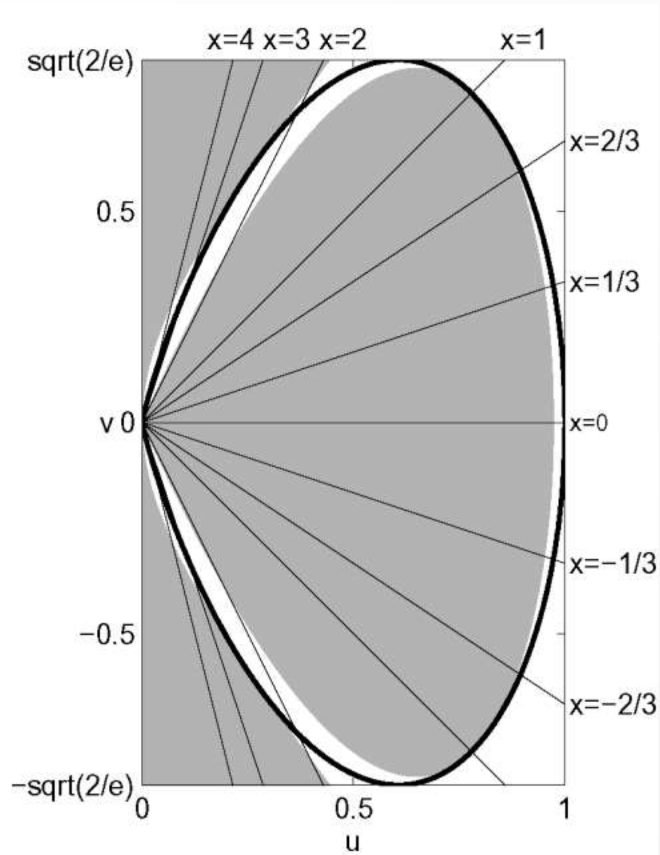
náhodné čísla s gauss rozdelením sú:

$G_1 = U_1 \times w$ a $G_2 = U_2 \times w$

Ratio-of-Uniforms

Ďalšia relatívne jednoduchá presná metóda.

Autori Kinderman a Monahan (1977), vylepšil J. L. Leva (1992)



Body vo vnútri krivky sú akceptované, body mimo krivky sú zamietnuté.

Generujú sa čísla U_1 a V_1 s rovnomerným rozdelením, pričom $U_1 \in (0,1)$ a $V_1 \in (-1,1)$. V_1 sa následne upraví ako

$$V_1 \longleftarrow V_1 \cdot \sqrt{\frac{2}{e}} \Big/ U_1$$

Stanislav.Antalic@fmph.uniba.sk

Ratio of Uniforms (idea)

Potom sa získané hodnoty U_1 a V_1 podrobia sérii testov

1) Automaticky sa akceptujú body, kde $V_1^2 \leq \left(5 - 4e^{1/4}U_1\right)$

2) Ak bod nevyhovuje podmienke 1) urobí sa test a posudzujú sa ďalej iba body kde $V_1^2 < \left(4e^{-1.35}/U_1 + 1.4\right)$

3) Ak bod vyhovuje podmienke 2) akceptuje sa ak

$$V_1^2 \leq \left(-4\ln\left(U_1\right)\right)$$

Výhoda je, že iba pre časť hodnôt sa počíta logaritmus.
Pre zvyšné sa využívajú iba jednoduché numerické operácie.
Nevýhoda: generujeme 2 rovnomerné čísla a získame iba jedno s gauss rozdelením.

Porovnanie

Čas potrebný pre vygenerovanie 1×10^9 čísel.

Centrálna limitná veta (N = 12)	Box-Muller	Polar	Ratio of Uniforms
6 min 6 sek	3 min 10 sek	3 min 3 sek	2 min 16 sek

Upozornenie: Výsledky silne závisia od optimalizácie kompilátora, realizácie kódu (napr. kontrol. výpisov) a samozrejme HW.

Poznámka: Box-Muller a Polar nám dávaju dva výsledky. Ak generujeme náhodné čísla s gauss-rozdelením často a dokážeme uschovať jedno číslo do ďalšieho volania generátora tak s tým získame faktor 2 vo výkone.

THE END