Appunti Lezione 05

Lorenzo Visca

1 Soluzione: verifica del teorema del limite centrale

Codice di riferimento: `centralSol.C`

Il codice di riferimento implementa alcune migliorie rispetto a quello fatto come esercizio:

- La macro prende come argomento extra una stringa, a seconda del valore passatole il programma utilizza una diversa funzione per generare i numeri casuali. La funzione `randomlong`, presentata a titolo di esempio, è un metodo più inefficiente che rifiuta i numeri estratti se non rientrano nell'intervallo desiderato. Per scegliere la funzione da utilizzare si definisce il puntatore a funzione `randFunc` che viene inizializzato in base al valore della stringa passata come argomento.
- Si utilizza un seed fisso per il generatore di numeri casuali in modo da poter riprodurre i risultati ottenuti in caso di necessità (debugging).
- Viene creato un array di istogrammi per gestirli in modo più efficiente.
- Gli istogrammi hanno normalmente un range 5σ, ma sono stati aggiustati per evitare di uscire dai limiti fisici dell'istogramma (0 e N).
- Viene utilizzata un'unica somma cumulativa per tutti gli istogrammi, inserendo il valore calcolato nell'istogramma corretto in base al numero di estrazioni corrispondente.

2 Codici

2.1 centralSol.C

```
#include <Riostream.h>
#include <TH1D.h>
#include <TFile.h>
#include <TCanvas.h>
#include <TRandom3.h>
#include <TStyle.h>
#include <TF1.h>
using namespace std;

const int nHisto = 7;
const int entries = 2000000;

double random1(const double& w)
{
    double r = gRandom->Rndm();
    return (r<0.5) ? (2.*w*r) : (1. - 2.*w*(1.-r));
}

double randomeff(const double& w)
{
    double r = 2.*w*gRandom->Rndm();
    return (r<w) ? r : (1.-2.*w+r);
}

double randomlong(const double& w)
{
    double r = gRandom->Rndm();
    while(r>w && r<(1.-w))
        r = gRandom->Rndm();
    return r;
}

void centralSol(const double& w = 0.2, TString choice = "", unsigned int seed = 98765)
{
    double (*randFunc)(const double&);
    if(choice.Contains("1") || choice.Contains("random1"))
    {
        randFunc = &random1;
        cout << "Using random1 function" << endl;
    }
    else if(choice.Contains("long") || choice.Contains("randomlong"))
        randFunc = &randomlong;
    else
        randFunc = &randomeff;

    gRandom->SetSeed(seed);

    if(w<0 || w>0.5)
    {
        cout << "w deve essere compreso tra 0 e 0.5" << endl;
        return;
    }

    cout << setprecision(5) << setiosflags(ios::scientific) << "CENTRAL LIMIT THEOREM
    SIMULATION" << endl;

    char nome[40];
    char titolo[40];

    TH1D* hist[nHisto];

    const int N[] = {1, 2, 5, 10, 50, 100, 300, 900, 2000};

    for(int i=0; i<nHisto; i++)
    {
```

```
65          snprintf(nome, 15, "hist%d", N[i]);
66          snprintf(titolo, 30, "Sum of %d values", N[i]);
67
68          double mean = 0.5 * (double)N[i];
69          double stdev = sqrt(((1./6.)*(2.*w*w - 3.*w + 3) - 0.25) * (double)N[i]);
70
71          double xmin = mean - 5.*stdev;
72          if(xmin<0.) xmin = 0.;
73          double xmax = mean + 5.*stdev;
74          if(xmax>(double)N[i]) xmax = (double)N[i];
75
76          hist[i] = new TH1D(nome, titolo, 100, xmin, xmax);
77      }
78
79      for(int i=0; i<entries; i++)
80      {
81          double sum = 0.;
82          for(int j=0; j<N[nHisto-1]; j++)
83          {
84              sum += randFunc(w);
85              for(int k=0; k<nHisto; k++)
86                  if(j == N[k]-1) hist[k]->Fill(sum);
87          }
88      }
89
90      TCanvas *cv[nHisto];
91      gStyle->SetOptFit();
92
93      for(int i=0;i<nHisto;i++)
94      {
95          cv[i] = new TCanvas();
96          hist[i]->SetFillColor(kOrange-6);
97          if(N[i]>4) hist[i]->Fit("gaus");
98          hist[i]->Draw();
99      }
100
101 }
```