Mètodes Numèrics I Probabilístics

Pràctica 2: Gaussianes o calcular probabilitats amb l'ordinador

Grau: Matemàtica Computacional i Analítica de Dades

Assignatura: Mètodes Numèrics i Probabilístics

NIUs dels estudiants: 1600959, 1544112

Noms dels estudiants: Pau Blasco Roca, Alberto Real Quereda

Índex

1 Implementació del treball

El programa principal anomenat “generador” utilitza les següents llibreries:

- <stdio.h>
- <stdlib.h>
- <time.h>
- <math.h>
- “aleatori.h”

El header “aleatori.h” conté les funcions: “generaUniforme” i “generaNormal”.

2 Funcions d' "aleatori.c"

Aquest arxiu conté les funcions:

- generaUniforme
- generaNormal

2.1 Funció generaUniforme

- generaUniforme(float **a**, float **b**, float* **r**):
  - **a**: nombre real que defineix l'inici de l'interval.
  - **b**: nombre real que defineix el final de l'interval.
  - ***r**: apuntador a una llista de dimensió 2 on guardarem els nombres aleatoris.

PROCÉS:

La funció guardarà a ***r** dos nombres de l'interval $[a, b]$ seleccionats pseudoaleatoriament.

2.1.1 Proves amb generaUniformes

Taula 1: Proves amb "generaUniformes()"

Interval	Output 1	Output 2	Output 3	Output 4
$[1, 10]$	6.06642 i 3.47812	6.89080 i 4.91022	9.75189 i 3.18939	1.88620 i 4.06019
$[-2, 2]$	0.26570 i 1.48318	1.48800 i -1.93470	-0.11837 i 0.32436	-1.68411 i -0.82574
$[-3, -1]$	-2.88167 i -1.39901	-2.22679 i -2.15851	-1.51357 i -1.77266	-1.65352 i -2.61731

2.2 Funció generaNormal

- generaNormal(float **mu**, float **sigma**):
  - **mu**: nombre real que defineix la mitjana a una distribució normal.
  - **sigma**: nombre real que defineix la desviació estàndar a una distribució normal.

PROCÉS:

La funció crida a "generaUniforme(-1,1,**r**)" per generar dos nombres aleatoris dins de l'interval [-1,1] i repeteix aquest procés mentre que la suma d'aquests nombres al quadrat sigui més gran que 1. Una vegada aconseguim dos valors vàlids calculem **n1** i **n2** de la següent manera:

$$n_1 = u\sqrt{\frac{-2\log s}{s}} \qquad n_2 = v\sqrt{\frac{-2\log s}{s}}$$

I fem:

$$n_1 = n_1 \cdot \sigma + \mu \qquad n_2 = n_2 \cdot \sigma + \mu$$

per tal d'ajustar aquests valors a la distribució normal amb la qual volem treballar.

2.2.1 Proves amb generaNormal

Taula 2: Proves amb "generaNormals()"

	Output 1	Output 2	Output 3	Output 4
$\mu = 0$, $\sigma = 1$	1.49098 i 1.59567	-2.00217 i -0.98668	-0.19983 i 0.61018	-0.65155 i 0.10102
$\mu = 5$, $\sigma = 10$	9.05048 i -0.30036	6.60223 i 11.55895	-15.32416 i -6.00150	8.63551 i 11.48892
$\mu = 1$, $\sigma = 3$	3.80116 i -0.69441	-0.04304 i 1.79985	1.25523 i -2.77290	-0.06164 i 0.84497

3 Manual d'utilització

Per fer servir aquest programa primer l'hem de compilar amb:

gcc -c aleatori.c -Wall -o aleatori.o

gcc -c generador.c -Wall -o generador.o

gcc aleatori.o generador.o -lm -o generador -static

Una vegada hem compilat només hem de cridar al programa de la següent manera:

./generador μ σ

I ens retornarà dues mostres de la distribució $N(\mu, \sigma)$ per la terminal.

4 Anàlisi de les dades de “comprovador.sh”

Per aquest apartat hem programat el fitxer “comprovador.sh”, el qual hem de cridar de la següent manera:

./script.sh p on p es un argument que ha de introduïr l'usuari.

Aquest executa el codi en C un milió de vegades (ja que per cada execució, com hem explicat, ens retorna dos valors aleatoris). Per tant, generarem dos milions de valors que es guardaran en un fitxer “out.txt”.

A més, el nostre programa retorna el nombre de nombres generats totals, el nombre de valors generats tals que $P(|x-\mu| \leq |x|) < p$ i el quocient entre aquests dos valors. Pel que hem observat de forma empírica, el quocient és quasi igual a **p**.

Per tant, veiem que el quocient s'aproxima, per la Llei dels Grans Nombres, a **p**; i ambdós ens mostren la probabilitat d'observar un nombre inferior a la cota donada.

4.1 Millora del sistema de comprovació

Després de fer algunes proves per a estimar i comprendre el valor de **p**, ens vam adonar que el programa trigava al voltant de 12 hores (uns 43200 segons) a executar-se 2000000 vegades. Per això, vam decidir canviar el sistema de comprovació i fer un programa en C que s'encarregués de la tasca.

El programa es compila de la següent manera:

```
gcc -c integracio.c -Wall -o integracio.o

gcc -c aleatori2M.c -Wall -o aleatori2M.o

gcc -c generador2M.c -Wall -o generador2M.o

gcc aleatori2M.o generador2M.o integracio.o -lm -o generador2M -statics
```

I el cridem de la següent manera:

./generador2M p

Aquest programa triga uns 18 segons a executar-se, sent així **2400** vegades més ràpid que l'anterior. Ens escriu els resultats en un fitxer “file.txt”, i també imprimeix per pantalla el quocient que aproxima **p**.