FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Fachbereich Elektrotechnik
und Informationstechnik

Prof. Georg Hoever

SS 2022

5. Praktikum zur Höhere Mathematik 2 für (Wirtschafts-)Informatik

Ziel dieses Praktikums ist die Monte-Carlo-Simulation einer Lotto-Ziehung.

Dazu soll die Funktion `int rand()` aus der `stdlib`-Bibliothek genutzt werden. Diese liefert Zufallszahlen zwischen 0 und `RAND_MAX`. Mit dem Modulo-Operator `%` erhält man damit durch `rand()%n` eine (ungefähr) gleichverteilte Zufallszahl aus $\{0, \dots, n-1\}$.

Durch `void srand(int s)` kann der Zufallszahlengenerator initialisiert werden. Bei gleichem `s` werden dann die gleichen Zufallszahlen erzeugt. Durch einen Aufruf von beispielsweise `srand(time(NULL))` erhält man dann zu verschiedenen Zeiten verschiedene Zufallswerte.

1. Aufgabe

Implementieren Sie eine Klasse `CZufall` mit den folgenden (public-)Methoden:

- `int wert(int a, int b)`, die eine (ungefähr) gleichverteilte Zufallszahl $n \in \mathbb{Z}$ mit $a \leq n \leq b$ liefert.

  Hinweis: Um `rand()` zu nutzen müssen Sie `<stdlib.h>` einbinden.

- `void initialisiere(int s)`, die den Zufallsgenerator mit `srand(s)` initialisiert.

- `void test(int a, int b, int N)`, die N mal eine Zufallszahl zwischen a und b zieht und ermittelt, wie oft dabei die Werte a, $a+1$, ..., b auftreten, und diese Häufigkeiten ausgibt.

- `void test_falsch(int a, int b, int N)`, die wie `test` funktioniert, allerdings vor jeder einzelnen Ziehung mit `initialisiere(time(NULL))` den Zufallsgenerator neu initialisiert.

  Hinweis: Um `time()` zu nutzen müssen Sie `<time.h>` einbinden.

Welche Ausgaben erhalten Sie, wenn Sie

a) mehrfach hintereinander die beiden Funktionen

   `void initialisiere(s)` und `test(3,7,10000)`

   mit gleichem Wert für s aufrufen,

b) mehrfach hintereinander die beiden Funktionen

   `void initialisiere(s)` und `test(3,7,10000)`

   mit mit verschiedenen Werten für s aufrufen,

c) mehrfach hintereinander die beiden Funktionen

   `void initialisiere(time(NULL))` und `test(3,7,10000)` aufrufen,

d) `test_falsch(3,7,10000)` aufrufen?

Interpretieren Sie die entsprechenden Ergebnisse!

2. Aufgabe

Implementieren Sie eine Klasse `CLotto`, mit der Sie eine k-aus-n-Lotto-Ziehung (ohne Zusatzzahl) simulieren können, also die Ziehung von k verschiedenen Zahlen zwischen 1 und n (jeweils inklusive).

Implementieren Sie (unter Benutzung der Klasse `CZufall`) dazu

a) einen Konstruktor, dem man als Argument die Werte k und n sowie eine Integer-Zahl s übergibt. Bei $s < 0$ wird der Zufallszahlengenerator mit `time(NULL)` initialisert, ansonsten mit s.

b) ein privates Attribut, das einen Tippzettel speichern kann, und eine public-Methode zum Setzen des Tippzettels,

c) eine Methode, die eine k-aus-n-Lotto-Ziehung (ohne Zusatzzahl) simuliert und eine entsprechende Ziehung (z.B. als Vektor oder als Array) zurückgibt.

   Nutzen Sie dazu Ihre Klasse `CZufall`.

   Hinweis: Achten Sie darauf, dass bei einer Ziehung keine Zahlen doppelt vorkommen dürfen!

d) eine Methode, die eine k-aus-n-Lotto-Ziehung durchführt und die Anzahl der mit dem Tippzettel übereinstimmenden Zahlen zurückgibt.

3. Aufgabe

Berechnen Sie simulativ mittels einer Monte-Carlo-Simulation (mit N Wiederholungen) die Wahrscheinlichkeit, genau r Richtige bei einer k-aus-n-Ziehung zu tippen. Simulieren Sie dabei zwei unterschiedliche Spieler-Typen:

1) Spieler 1 nutzt immer den gleichen Tippzettel.

   Legen Sie dazu einen willkürlichen oder (einmalig) zufällig gezogenen Tippzettel an. Nutzen Sie dann den wiederholten Aufruf der Methode aus 2d) zur Simulation.

2) Spieler 2 nutzt jedes Mal einen neuen Tippzettel.

   Führen Sie dazu wiederholt die folgenden beiden Schritte aus:

   - Ziehen Sie (mit der Methode aus 2c)) einen Tippzettel, und legen Sie diesen entsprechend 2b) ab.
   - Nutzen Sie dann Ihre Methode aus 2d) um zu schauen, ob es bei einer Ziehung genau r Richtige gibt.

Implementieren Sie eine entsprechende Funktion mit den Parametern r, k, n, N und *typ*, und testen Sie die Funktion mit den konkreten Werten r, k und n aus Ihrem Praktikums-e-Test, Aufgabe 3, und einem geeigneten großen N (s. dazu auch den Abschnitt „Für Interessierte").

Für Interessierte:

Wie genau sind die Simulationsergebnisse von Aufgabe 3 in Abhängigkeit von der Anzahl N der Monte-Carlo-Simulationen?

Experimentieren Sie dazu mit verschiedenen N (z.B. von 100 bis 10^6).

Eine Möglichkeit, die Genauigkeit abzuschätzen, ist, sie mit dem exakten Ergebnis zu vergleichen, das Sie ja durch Ihren Praktikums-e-Test, Aufgabe 3, haben. Man kann dann beispielsweise den Mittelwert der Abweichungen zum exakten Ergebnis bei mehrfacher Wiederholung der Simulation mit gleichem N berechnen.

In realen Situationen, in denen man Monte-Carlo-Simulationen einsetzt, kennt man den exakten Wert nicht. Als Genauigkeit kann man dann die Standardabweichung heranziehen (s. Skript, Definition 4.11; zur Berechnung s. der Tipp unten).

Berechnen Sie (auf die eine oder andere Art) die Genauigkeit für verschiedene N. Erkennen Sie ein System? Wie sehr verkleinert sich die Genauigkeit, wenn Sie N verzehnfachen?

Man kann zeigen, dass die Genauigkeit proportional zu ${}^1\!/\!\sqrt{N}$ ist, d.h. bei einer Verhundertfachung von N erhalten Sie eine Dezimalstelle mehr Genauigkeit.

Tipp zur Berechnung der Standardabweichung: Die Formel aus Definition 4.11, 2., legt nahe, dass man zunächst alle Ziehungen durchführen muss, mit den Werten dann den Mittelwert $\overline{x}$ berechnen kann und dann mit den (abgespeicherten) Werten x_k die Varianz und damit dann die Standardabweichung berechnen kann.

Man kommt aber auch ohne die Abspeicherung aller Werte x_k aus: Es gilt

$$\begin{aligned}
s^2 &= \frac{1}{N-1}\sum_{k=1}^{N}(x_k-\overline{x})^2 \;=\; \frac{1}{N-1}\sum_{k=1}^{N}\left(x_k^2-2x_k\overline{x}+\overline{x}^2\right)\\
&= \frac{1}{N-1}\Big(\sum_{k=1}^{N}x_k^2-2\overline{x}\sum_{k=1}^{N}x_k+\sum_{k=1}^{N}\overline{x}^2\Big)\\
&= \frac{1}{N-1}\Big(\sum_{k=1}^{N}x_k^2-2\overline{x}\cdot N\cdot\overline{x}+N\cdot\overline{x}^2\Big)\\
&= \frac{1}{N-1}\Big(\sum_{k=1}^{N}x_k^2-N\cdot\overline{x}^2\Big) \;=\; \frac{1}{N-1}\sum_{k=1}^{N}x_k^2-\frac{N}{N-1}\overline{x}^2.
\end{aligned}$$

Man kann nun parallel $\sum_{k=1}^{N}x_k$ und $\sum_{k=1}^{N}x_k^2$ bilden, ohne die x_k abzuspeichern, und dann am Ende die Varianz und damit die Standardabweichung entsprechend der letzten Darstellung berechnen.