Programmieren für Physiker

Interfakultatives Institut für Anwendungen der Informatik
Institut für Theoretische Teilchenphysik

Prof. Dr. M. Steinhauser, Dr. A. Mildenberger
`http://comp.physik.kit.edu`

SS 2017 – Blatt 10
Bearbeitungszeitraum: bis 05. Juli 2017

Klausurtermin: Dienstag, 25. Juli 2017, 17.30 Uhr.
Es ist erforderlich, sich bis zum 18.07.2017 unter https://campus.studium.kit.edu/ zur Klausur anzumelden. Zur Teilnahme sind 80% der Pflichtaufgaben erfolgreich zu lösen.

Aufgabe 25: Normalverteilte Zufallszahlen **Pflichtaufgabe**

Gelegentlich werden in numerischen Simulationen Zufallszahlen benötigt, die der Gaußschen Normal-Verteilung

$$\mathcal{P}(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$$

genügen. Viele Zufallszahlen-Generatoren liefern jdeoch gleichverteilte Zufallszahlen. Wie können damit normalverteilte erzeugt werden?
Eine Möglichkeit ist die sogenannte Polarmethode (G. Marsaglia, 1964, basierend auf einer Idee von Box und Muller, 1958):

S1. Generiere zwei Zufallszahlen v_1 und v_2, die im Intervall $[-1, 1]$ gleichverteilt sind.
S2. Berechne $s = v_1^2 + v_2^2$.
S3. Falls $s \geq 1$ oder $s = 0$: Gehe zurück zu Schritt S1.
S4. Setze

$$z_1 = v_1 \cdot \sqrt{\frac{-2 \ln s}{s}} \qquad \text{und} \qquad z_2 = v_2 \cdot \sqrt{\frac{-2 \ln s}{s}}.$$

Die beiden Variablen z_1 und z_2 sind jeweils normalverteilt (und unabhängig voneinander).

Dieses Verfahren soll in einer Klasse `normal_generator` implementiert werden. Eine Methode `get` soll eine normalverteilte Zufallszahl ermitteln und zurückgeben. Hier soll jedoch die Besonderheit des obigen Algorithmus ausgenutzt werden, der ja immer zwei Zufallszahlen generiert. Beim ersten Aufruf von `get` werden zwei normalverteilte Zufallszahlen berechnet, eine davon wird sofort zurückgegeben und die andere in einer Mitgliedsvariablen abgespeichert. Bei einem zweiten Aufruf von `get` kann der bereits berechnete Wert verwendet werden, es ist also keine neuer Durchlauf des obigen Algorithmus nötig. Bei einem dritten Aufruf wäre ist wieder eine Rechnung nötig, bei einem vierten nicht, etc.
Um dieses Verhalten zu erzielen, ist in der Klasse intern eine Statusvariable notwendig, die festhält, ob noch ein Wert vorliegt. Diese Statusvariable sollte schon vor dem ersten `get`-Aufruf einen sinnvollen Wert enthalten, deswegen ist ein Konstruktor zu schreiben, der die Statusvariable entsprechend initialisiert.
Im Hauptprogramm erzeugen Sie bitte zunächst $N = 10^5$ Zufallszahlen x_i und berechnen Sie den Mittelwert m und die Standardabweichung σ:

$$m = \frac{1}{N} \sum_{i=1}^{N} x_i \qquad \sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_i - m)^2} = \sqrt{\frac{1}{N} \left(\sum_{i=1}^{N} x_i^2 \right) - m^2} \ .$$

In einem zweiten Schritt soll mittels weiterer 10^5 Zufallszahlen bestimmt werden, wie groß der Anteil der Zahlen ist, deren Betragsabweichung vom Mittelwert größer als σ, 2σ und 3σ ist.

Hinweis: Der Ausdruck `2.0*rand()/(double)RAND_MAX - 1` ergibt eine gleichverteilte Zufallszahl im Intervall $[-1, 1]$. Die Initialisierung des Zufallszahlengenerators kann beispielsweise im Konstruktor erfolgen.

Zusatzfragen: (i) Wie erhält man aus den normalverteilten Zufallszahlen allgemein Gauß-verteilte Variablen mit vorgegebenen Parametern Mittelwert und Varianz? (ii) Wie oft werden durchschnittlich die Schritte S1–S3 ausgeführt, bis ein Wert $s < 1$ gefunden ist? (iii) Warum liefert dieses Verfahren die gewünschte Verteilung?

Aufgabe 26: Acht-Damen-Problem **Pflichtaufgabe**

Auf einem Schachbrett sollen acht Damen so platziert werden, dass keine Dame eine andere Dame schlagen kann. Im Schachspiel bewegen sich Damen waagrecht, senkrecht und diagonal beliebig weit.
Das Programm soll einen sogenannten *Backtracking*-Algorithmus verwenden. Beginnen Sie mit einem leeren Spielfeld und besetzen Sie es zeilenweise nach folgendem Schema:
In der aktuellen Zeile werden nacheinander alle acht Spaltenpositionen für die neue Dame ausprobiert. Bei jedem Einzelnen dieser acht Versuche wird nun getestet, ob die neue Dame von einer bisher schon vorhandenen Dame geschlagen werden kann. Falls ja, ist diese Konstellation nicht weiter zu verfolgen. Falls nein, wird die Dame an dieser Position (temporär) vermerkt und das gleiche Verfahren zur Positionsfindung in der nächsten Zeile begonnen.
Zusätzlich ist zu berücksichtigen: Wenn eine Dame in der achten Zeile regelgerecht gesetzt werden konnte, ist eine Lösung gefunden und diese soll ausgegeben werden.

Diese Art der Lösungssuche kann am besten *rekursiv* durch eine Funktion programmiert werden, die jeweils in einer Schleife alle Spalten der aktuellen Zeile ausprobiert und sofort bei einer gültigen Positionierung sich selbst aufruft. Die zu bearbeitende Zeile soll als Funktionsargument übergeben werden. Um die Konfiguration der Damen während der Lösungssuche aufzubewahren, ist ein eindimensionales Feld zweckmäßig, in welchem zu jeder Zeile die Spaltenposition der Dame vermerkt wird.

Geben Sie die Lösungen auf dem Bildschirm aus.

Zusatzfragen (freiwillig): (i) Wieviele Lösungen gibt es insgesamt? (ii) Modifizieren Sie Ihr Programm so, dass die Anzahl der Lösungen für n Damen auf einem $n \times n$ Spielfeld berechnet wird.

Aufgabe 27: Auflösung einer Rekursion **freiwillig**

Gegeben sei eine rekursive Funktionen, die folgende, relativ einfache Struktur habe:

```
Type2 f (Type1 x)
{
  if( test(x)) return q(x) ;
  return f(p(x)) ;
}
```

Dabei sind `Type1 p (Type1)` und `Type2 q (Type1)` sowie `bool test (Type1)` beliebige Funktionen, die hier für Rechenoperationen mit dem Argument `x` stehen. Geben Sie mit Hilfe einer Schleife eine nicht-rekursive Formulierung der obigen Funktion `f` an.
