Universität Karlsruhe (TH)
Forschungsuniversität • gegründet 1825

Institut für Angewandte und
Numerische Mathematik

Dr. G. Bohlender
Dipl.–Math. techn. K. Feldhoff
Dipl.–Math. techn. M. Richter

02.05.2007

Programmieren: Einstieg in die Informatik mit Java

SS 2007

Aufgabenblatt 3

Bearbeitungszeitraum: 03.05.2007 – 16.05.2007

Aufgabe 6 *Zufallsgenerator*

Ein einfaches Verfahren, um eine Folge von im Intervall $[0, 1)$ gleichverteilten Zufallszahlen $x_0, x_1, x_2, \ldots$ zu generieren, besteht darin den folgenden Algorithmus umzusetzen:
Ausgehen von einem sogenannten *Keim* x_0 und zweier vorgegebener Zahlen a und b berechnet man alle weiteren Zufallszahlen $x_1, x_2, \ldots$ gemäß

$$y_{n+1} = ax_n + b,$$
$$x_{n+1} = y_{n+1} - \lfloor y_{n+1} \rfloor,$$

für $n = 0, 1, 2, \ldots$. Dabei bezeichnet $\lfloor y \rfloor$ die *Gaußklammer* von y, d.h. die größte ganze Zahl, die kleiner als y ist.
Erstellen Sie ein Java–Programm mit dem Namen `Zufallszahlen`, das eine Folge von Zufallszahlen berechnet. Gehen Sie dabei wie folgt vor:

(a) Erstellen Sie eine statische Methode mit dem Namen `zufall`, die drei formale Parameter `a`, `b` und `x` vom Typ `double` besitzt und einen `double`–Wert zurückgibt. Berechnen Sie in dieser Methode die nächste Zufallszahl x_{n+1}, wobei der Wert von x_n im formalen Parameter `x` übergeben wird. Die Werte für a und b sollen in den formalen Parametern `a` und `b` übergeben werden. Geben Sie den Wert der nächsten Zufallszahl zurück.

Hinweis: Die Gaußklammer kann mit der Java–Methode `Math.floor()` berechnet werden.

(b) Erstellen Sie eine `main`–Methode, welche die ersten 20 Zufallszahlen $x_0, \ldots, x_{19}$ auf der Konsole ausgibt. Verwenden Sie als Keim $x_0 = 0,5$ sowie $a = 12054,6729308$ und $b = 0,318328791694$.

Aufgabe 7 (Pflichtaufgabe) *Marktgleichgewicht*

In einem idealisierten Markt wird der *Marktpreis* eines Guts durch einen Ausgleichsprozess zwischen Angebot und Nachfrage bestimmt. Ein solcher Markt kann durch ein einfaches mathematisches Modell beschrieben werden. Man wählt dazu zwei Funktionen, welche die Prozesse am Markt modellieren. Eine Funktion, die sogenannte *Angebotsfunktion* p_A, bildet eine *nachgefragte Menge* m auf einen Preis $p_A(m)$ ab. Diese Funktion ist im allgemeinen eine monoton steigende Funktion (Je mehr nachgefragt wird, desto höher ist der Preis). Eine zweite Funktion, die sogenannte *Nachfragefunktion* p_N, bildet eine *angebotene Menge* m auf einen Preis $p_N(m)$ ab. Diese Funktion ist im allgemeinen eine monoton fallende Funktion (Je mehr angeboten wird, desto niedriger ist der Preis). Den Marktpreis p_0 erhält man, indem man den Schnittpunkt der Angebots- und der Nachfragekurve bestimmt. Die Stelle, an der sich beide Kurven schneiden entspricht dabei der *umgesetzten Menge* m_0.
Erstellen Sie ein Java–Programm mit dem Namen *Marktgleichgewicht*, das einen idealisierten Markt simuliert. Gehen Sie dabei wie folgt vor:

(a) Erstellen Sie eine statische Methode mit dem Namen `angebot`, die einen formalen Parameter `menge` vom Typ `double` besitzt und einen `double`–Wert zurückgibt. Diese Methode soll zu einer übergebenen Menge m den Wert der Angebotsfunktion $p_A(m)$ gemäß

$$p_A(m) = 10 + 5\sqrt{m}$$

zurückgibt. Erstellen Sie eine zweite statische Methode mit dem Namen `nachfrage`, die zu einer übergebenen Menge m den Wert der Nachfragefunktion $p_N(m)$ gemäß

$$p_N(m) = 80\,\mathrm{e}^{-0.04m}$$

zurückgibt. Erstellen Sie anschließend eine dritte statische Methode mit dem Namen `ungleichgewicht`, die zu einer gegebenen Menge m das sogenannte *Marktungleichgewichts* $u(m)$ zurückgibt. Das Marktungleichgewicht ist dabei die Differenz zwischen Angebots- und Nachfragefunktion, d.h.

$$u(m) = p_A(m) - p_N(m).$$

Hinweis: Die Quadratwurzel und die Exponentialfunktion können mit den Java–Methoden `Math.sqrt()` und `Math.exp()` berechnet werden.

(b) Bestimmen Sie die Stelle m_0, an der sich die Angebots- und die Nachfragekurve schneiden. An dieser Stelle gilt $u(m_0) = 0$. Gesucht ist also die Nullstelle der Funktion u. Erstellen Sie eine statische Methode mit dem Namen `nullstelle`, die keine formalen Parameter besitzt, und die die Nullstelle von u zurückgibt. Verwenden Sie zur Nullstellenbestimmung den sogenannten *Bisektionsalgorithmus*:

Beim Bisektionsalgorithmus wählt man ein Anfangsintervall $[a^{(0)}, b^{(0)}]$, in dem sich die Nullstelle von u befindet. Als nächstes bestimmt man die Intervallmitte $c^{(0)} = \frac{1}{2}(a^{(0)} + b^{(0)})$ und wertet u an dieser Stelle aus. Danach wird das Intervall, in dem sich die Nullstelle befindet, iterativ verkleinert. Das geschieht in folgender Weise:

Solange $|u(c^{(i)})|$, $i = 0, 1, 2, \ldots$, größer als ein vorgegebener Wert `TOL` ist, werden folgende Schritte durchgeführt:

BITTE WENDEN

- Es werden die Funktionswerte $u(a^{(i)})$ und $u(b^{(i)})$ bestimmt.
- Gilt $u(a^{(i)})\,u(c^{(i)}) < 0$, werden die Intervallgrenzen wie folgt angepasst:

$$a^{(i+1)} = a^{(i)}, \qquad b^{(i+1)} = c^{(i)}.$$

  Gilt $u(a^{(i)})\,u(c^{(i)}) \geq 0$, werden die Intervallgrenzen wie folgt angepasst:

$$a^{(i+1)} = c^{(i)}, \qquad b^{(i+1)} = b^{(i)}.$$

- Es wird die Mitte des neuen Intervalls $c^{(i+1)} = \frac{1}{2}(a^{(i+1)} + b^{(i+1)})$ bestimmt und $u(c^{(i+1)})$ berechnet.
- Anschließend wird der Iterationsindex i formal um Eins erhöht.

Der Algorithmus bricht ab, wenn $|u(c^{(i)})|$ kleiner oder gleich dem vorgegebenen Wert `TOL` ist. Der Wert von $c^{(i)}$ entpricht dann dem Näherungswert für die Nullstelle.

Setzen Sie den Bisektionsalgorithmus mit einer `while`-Schleife um. Definieren Sie drei Variablen `a`, `b` und `c` vom Typ `double`, in denen Sie die Werte $a^{(i)}$, $b^{(i)}$ und $c^{(i)}$ für $i = 0, 1, 2, \ldots$ speichern. Definieren Sie drei weitere Variablen `ua`, `ub` und `uc`, in denen Sie die Werte $u(a^{(i)})$, $u(b^{(i)})$ und $u(c^{(i)})$ speichern. Verwenden Sie $a^{(0)} = 0$, $b^{(0)} = 100$ und `TOL` $= 10^{-6}$. Es ist nicht notwendig, den Iterationsindex i zu speichern.

Hinweis: Die Betragsfunktion kann mit der Java–Methode `Math.abs()` berechnet werden.

(c) Erstellen Sie eine `main`–Methode, welche die umgesetzte Menge m_0 und den Marktpreis p_0 auf der Konsole ausgibt.

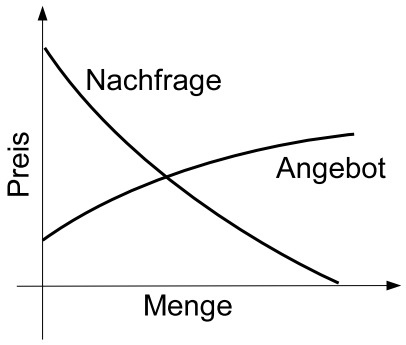

Abbildung 1: Menge–Preis–Diagramm