

Test von Pseudo-Zufallszahlen

Vorstellung und Umsetzung von vier Verfahren

Maturaarbeit von Daniel Würsch, 6D

betreuende Lehrperson: Rolf Peterhans

Kantonsschule Zug

2002/2003

Inhaltsverzeichnis

1. Vorwort	2
2. Die <code>Uniform</code> - und die <code>rand()</code> -Funktion	3
3. Vier Verfahren zum Test von Pseudo-Zufallszahlen und ihre Umsetzung in je einem Programm	6
3.1. Kolmogorow-Smirnow-Test	7
3.1.1. Verfahren	7
3.1.2. Umsetzung	7
3.1.3. Nullhypothese	8
3.1.4. Programm	9
3.2. χ^2 -Test	14
3.2.1. Verfahren	14
3.2.2. Umsetzung	14
3.2.3. Nullhypothese	15
3.2.4. Programm	16
3.3. Run-Test	21
3.3.1. Verfahren	21
3.3.2. Umsetzung	21
3.3.3. Nullhypothese	25
3.3.4. Programm	25
3.4. Poker-Test	30
3.4.1. Verfahren	30
3.4.2. Umsetzung	30
3.4.3. Nullhypothese	31
3.4.4. Programm	32
4. Resultate der Tests der <code>Uniform</code> -Funktion	35
4.1. Kolmogorow-Smirnow-Test	35
4.2. χ^2 -Test	37
4.3. Run-Test	39
4.4. Poker-Test	41
5. Diskussion	43
6. Anmerkungen	44
7. Literaturverzeichnis	45

1. Vorwort

Lange Zeit wusste ich nicht, in welchem Fach ich meine Maturaarbeit schreiben sollte, ich entschied mich letztendlich recht kurzfristig für die Mathematik. Dabei war ich mir relativ sicher, dass ich etwas im Zusammenhang mit Wahrscheinlichkeitsrechnung und Zufallszahlen tun wollte. Nachdem ich schon für längere Zeit ein Thema gesucht hatte, schlug mir Herr Peterhans vor, mich mit Zufallszahlengeneratoren-Tests auseinanderzusetzen, worauf ich auch einige Programme in TopPascal (Version 3.06) schrieb, um einfache Versionen dieser Tests umzusetzen.

Die ersten Versionen dieser Tests testeten den Zufallszahlengenerator von TopPascal, ich hatte aber von Anfang an vor, die endgültigen Tests mit einer anderen Funktion durchzuführen. Schliesslich entschied ich mich, den Zufallszahlengenerator des TI-92 von Texas Instruments,¹ die `rand()`-Funktion, zu testen. Ich wählte diese Funktion, weil es mich interessierte, welche Qualität ein Zufallszahlengenerator haben könne, der auf den vergleichsweise beschränkten Mitteln eines „etwas grösseren Taschenrechners“ aufbauen müsse. Doch schlussendlich habe ich nicht diese Funktion getestet, sondern stattdessen die ihr – so weit ich weiss – ziemlich ähnliche `Uniform`-Funktion.

Ich glaube durchaus nicht, dass nur gerade vier Tests genügen würden, die Qualität eines Zufallszahlengenerators ausreichend zu bestätigen, um ihn für ernsthafte Anwendungen einzusetzen. Doch es ist auch nicht mein Ziel, mit dieser Arbeit die von der `Uniform`-Funktion generierten Zahlen erschöpfend zu beurteilen – obwohl ich sie beurteilen werde – es ist vielmehr mein Ziel, die vier vorgestellten Testverfahren ein wenig kennen zu lernen, von denen vor allem der Kolmogorow-Smirnow- und der χ^2 -Test (χ^2 -Anpassungstest) mir relativ wichtig scheinen, da sie gut mit anderen Tests verknüpft werden können.

Wie schon die ersten Anfangsprogramme, habe ich auch die vier von mir behandelten Testverfahren in TopPascal-Programmen umgesetzt. Ich wählte TopPascal, weil ich mit dieser Programmiersprache vom Informatik-Unterricht her schon relativ gut vertraut war. Aus heutiger Sicht denke ich jedoch, dass dies ein Fehler war, denn ich musste die Programme teilweise stark verändern – womit sie auch unübersichtlicher wurden – nur weil es nicht möglich war, in dieser Programmiersprache Werte und Listen einer bestimmten Grösse wie gewünscht zu behandeln.

2. Die Uniform- und die rand()-Funktion

Als ich mich endgültig entschied, den Zufallszahlengenerator des TI-92 – der neuere Voyage 200 von Texas Instruments benutzt, so weit ich weiss, den genau gleichen Zufallszahlengenerator wie der TI-92 – zu testen, die rand()-Funktion zu testen,² war ich mit der Umsetzung der Testverfahren schon relativ weit und wusste deshalb, dass ich in mehreren Tests jeweils mehr als eine Million Zahlen testen würde. Es schien mir zu umständlich, alle Zahlen mit dem TI-92 selbst zu erzeugen und sie einem Computer zu übermitteln. Ich versuchte deshalb herauszufinden, wie die rand()-Funktion genau lautet, um die Pseudo-Zufallszahlen jeweils direkt in den Programmen selbst erzeugen zu können.

Doch leider gelang mir dies nicht, ich weiss nicht mit letzter Sicherheit, wie die rand()-Funktion lautet. Als Antwort auf mein Email vom 3. Juli 2002 an TI-Cares@ti.com, einen Hilfedienst von Texas Instruments, worin ich fragte, welche Regeln die rand()-Funktion ausmachen würden, erhielt ich zwei Tage später die Antwort „[...] Texas doesn't want to communicate anything about the way calculator works, so no information is available for that kind of things, sorry. [...]“.

Glücklicherweise wies mich Claudius M. Zingerli auf einen Aufsatz von Pierre L'Ecuyer hin, der im Juni 1988 in der Zeitschrift Communications of the ACM erschienen war,³ wo ich dann folgende Funktion fand:

```
FUNCTION Uniform : REAL;
  VAR
    Z, k : INTEGER;
  BEGIN
    k := s1 DIV 53668;
    s1 := 40014 * (s1 - k * 53668) - k * 12211;
    IF s1 < 0 THEN s1 := s1 + 2147483563;

    k := s2 DIV 52774;
    s2 := 40692 * (s2 - k * 52774) - k * 3791;
    IF s2 < 0 THEN s2 := s2 + 2147483399;

    Z := s1 - s2;
    IF Z < 1 THEN Z := Z + 2147483562;

    Uniform := Z * 4.656613E-10
  END
```

Die Uniform-Funktion scheint mit der rand()-Funktion beinahe identisch zu sein: Durch Ergänzen der Zeile „Uniform := Z * 4.656613E-10“ auf „Uniform := Z * 4.656613059555E-10“ erhält man eine Funktion, die, ausgehend von s1 = 12345 und s2 = 67890 (in der rand()-Funktion mit seed1 und seed2 bezeichnet), für die folgenden hunderttausend Pseudo-Zufallszahlen die gleichen Werte ausgibt wie die rand()-Funktion des TI-92, falls die Werte der Pseudo-Zufallszahlen der oben beschriebenen und wie angegeben geänderten Funktion auf vierzehn Nachkommastellen gerundet werden. Die jeweiligen Seeds der beiden Funktionen sind natürlich auch ohne diese Änderungen identisch.

Natürlich hätte es zu viel Zeit gebraucht, für alle Startseeds von null bis zum Beispiel 2147483563 zu prüfen, ob die beiden Funktionen jeweils die gleichen Pseudo-Zufallszahlen ausgeben. Es kann also sein, dass ich einige Unterschiede zwischen der rand()-Funktion des TI-92 und der oben beschriebenen Uniform-Funktion übersehen habe.

Doch für bestimmte Werte, die in der Uniform-Funktion entweder für die Seeds oder die Pseudo-Zufallszahlen spezielle Werte ergeben, habe ich die beiden Funktionen verglichen. Dabei entdeckte ich folgende Unterschiede:

- Bei $s_1 = 40692$ und $s_2 = 40014$ wird die beschriebene Funktion als Pseudo-Zufallszahl einen Wert ausgeben, der beinahe eins ist, die TI-92-Funktion jedoch den Wert null. Die errechneten Seeds sind aber bei beiden Funktionen dieselben. Betrachtet man die Uniform-Funktion, wird klar, dass $s_1 = 40692$ und $s_2 = 40014$ Werte sind, bei denen für die errechneten Seeds jeweils $s_1 = s_2$ gilt, somit wäre $z = 0$. Wollte man diesen Unterschied beseitigen, könnte man deshalb die Zeile „IF $Z < 1$ THEN $Z := Z + 2147483562$;“ in „IF $Z < 0$ THEN $Z := Z + 2147483562$ “ umändern, auf diese Weise würde zu z nichts hinzugezählt und die Funktion gäbe wie gewünscht den Wert null aus. Natürlich gibt es neben $s_1 = 40692$ und $s_2 = 40014$ noch weitere Fälle, in denen nach einem Durchlauf $s_1 = s_2$ gilt, die Startseeds müssen nur die Gleichung $40014 * (s_1 - (s_1 \text{ div } 53668) * 53668) - (s_1 \text{ div } 53668) * 12211 = 40692 * (s_2 - (s_2 \text{ div } 52774) * 52774) - (s_2 \text{ div } 52774) * 3791$ erfüllen, die sich für Werte, die kleiner als 53668 beziehungsweise 52774 sind, auf $40014 * s_1 = 40692 * s_2$ – dabei müssen s_1 und s_2 natürliche Zahlen sein – vereinfacht.
- Für den Fall $s_2 = 2147483399$ ergibt sich in der Uniform-Funktion $s_2 = 0$, worauf s_2 in allen weiteren Durchläufen null bleibt. Doch wird $s_2 = 2147483399$ durch die Funktion des TI-92 nicht zu null, sondern zu $s_2 = 429460057$. Ist am Anfang $s_2 = 0$, wird es durch die `rand()`-Funktion auf $s_2 = 2147483563$ gesetzt, dies ergibt in einem nächsten Durchlauf natürlich nicht mehr $s_2 = 0$, sondern – entsprechend der Uniform-Funktion – $s_2 = 6673488$.
- Ist $s_1 = 2147483563$, ergibt dies mit der Uniform-Funktion $s_1 = 0$, was in weiteren Durchläufen nur immer $s_1 = 0$ ergibt. Dies ist bei der `rand()`-Funktion anders: Wird $s_1 = 2147483563$ eingegeben, beträgt s_1 nach einem Durchlauf 858981421. Wird $s_1 = 0$ eingegeben, ändert der Wert auf $s_1 = 2147483563$. In der Folge wird s_1 aber seltsamerweise nicht zu 858981421 sondern für s_1 ergibt sich – wie es der Uniform-Funktion zufolge eigentlich sein sollte – null. Im nächsten Durchlauf wird der Wert wieder auf $s_1 = 2147483563$ gesetzt. s_1 nimmt dann abwechslungsweise die Werte 0 und 2147483563 an.

Dieser letzte Unterschied stellt den einzigen mir bekannten Fall dar, bei dem zwei exakt gleiche Startwerte für s_1 , nach der Verarbeitung durch die `rand()`-Funktion, zwei verschiedene Endwerte für s_1 ergeben. Es scheint eine Rolle zu spielen, ob zuerst $s_1 = 0$ gesetzt wird, was 2147483563 ergibt, oder ob ohne diesen Umweg gleich $s_1 = 2147483563$ gesetzt wird.

Ich bin mir nicht sicher, ob diese Veränderungen der Uniform-Funktion, die anscheinend in der `rand()`-Funktion vorgenommen wurden, wirklich sinnvoll sind.

Während das Abändern der Zeile „IF $Z < 1$ THEN $Z := Z + 2147483562$;“ wahrscheinlich nicht allzu viel an der Qualität der Pseudo-Zufallszahlen verändert, denke ich, dass die Umgehung eines bleibenden $s_2 = 0$ geringfügig vorteilhaft ist, obwohl dies die Funktion etwas verlangsamt, da immer überprüft werden muss, ob $s_2 = 0$ ist.

Durch die Behebung des letzten genannten Unterschiedes wird die Funktion ebenfalls leicht verlangsamt, da sie nun nicht nur den letzten Wert von s_1 sondern die beiden letzten Werte von s_1 kennen muss. Falls diese Änderung nur den Fall $s_1 = 0$ und $s_1 = 2147483563$ betrifft, lässt sich diese Verlangsamung nicht rechtfertigen, denn die Änderung bringt gar keine Vorteile mit sich. Der in diesem einzigen mir bekannten Fall zwischen den Extremwerten null und 2147483563 schwankende Wert von s_1 schadet wahrscheinlich der Qualität der Pseudo-Zufallszahlen sogar. Es wäre wahrscheinlich besser, wenn s_1 beispielsweise von null aus auf $s_1 = 2147483563$ gesetzt würde, was dann –

wie bei der direkten Eingabe von $s1 = 2147483563$ – in einem weiteren Durchlauf $s1 = 858981421$ ergäbe.

Wie schon geschrieben, scheinen sich die Unterschiede zwischen den beiden Funktionen nur bei wenigen Zahlen zu zeigen, die Änderung der Zeile „IF $Z < 1$ THEN $Z := Z + 2147483562$;“ betrifft sogar nur bestimmte Zahlenpaare. Ich habe mich deshalb entschlossen, anstatt der `rand()`-Funktion des TI-92 die oben beschriebene `Uniform`-Funktion zu testen – allerdings mit der Abänderung „`Uniform := Z * 4.656613059555E-10`“. Die so entstandene Funktion gibt anscheinend für die meisten Startwerte von $s1$ und $s2$ die gleichen Pseudo-Zufallszahlen wie die `rand()`-Funktion aus.

Ich denke, dass dieser Entscheid, die weiteren Unterschiede der `Uniform`-Funktion zur `rand()`-Funktion nicht miteinzubeziehen, nur sehr geringe Auswirkungen auf die Resultate der Tests haben wird, im besten Fall vielleicht sogar gar keine.

Ich werde also zur Erzeugung der Zahlen in den im Folgenden beschriebenen Programmen die `Uniform`-Funktion benutzen. Dabei werde ich jeweils bei $s1$ bzw. `seed1 = 12345` und $s2$ bzw. `seed2 = 67890` beginnen. Der Grund dafür – wenn dies überhaupt ein Grund ist – ist, dass `seed1` und `seed2` vom TI-92 auf diese Startwerte – auf die Werk-einstellung – gesetzt werden, wenn der Befehl `RandSeed 0` ausgeführt wird.⁴

3. Vier Verfahren zum Test von Pseudo-Zufallszahlen und ihre Umsetzung in je einem Programm

Es gibt zwei grundlegend verschiedene Arten von Verfahren zum Test von Pseudo-Zufallszahlengeneratoren: Es gibt empirische und theoretische Tests.

Empirische Tests beruhen auf der Berechnung einer Folge von Pseudo-Zufallszahlen nach den vorgegebenen Bildungsregeln und der Prüfung der so erhaltenen Pseudo-Zufallszahlenfolge mittels statistischer Tests.

Theoretische Tests basieren auf der Überlegung, von den zur Errechnung der Pseudo-Zufallszahlen angewendeten Bildungsregeln auf die Eigenschaften der erzeugten Zahlen zu schliessen.⁵

Alle vier von mir betrachteten Verfahren sind empirische Tests. Man kann sie ausserdem in Unabhängigkeits- und Gleichverteilungstests unterteilen: In Unabhängigkeitstests wird überprüft, ob bestimmte Abfolgen von Zahlen mit grösserer Wahrscheinlichkeit auftreten als andere, es wird also die Häufigkeit des Auftretens verschiedener Abfolgen betrachtet. In Gleichverteilungstests wird untersucht, ob die Pseudo-Zufallszahlen im ganzen Bereich, in welchem sie liegen, überall mit gleich grosser Wahrscheinlichkeit auftauchen, also gleichverteilt sind.

3.1. Kolmogorow-Smirnow-Test

3.1.1. Verfahren

Der Kolmogorow-Smirnow-Test basiert auf dem Vergleich der erwartungsmässigen theoretischen Verteilungsfunktion $F(x)$ von Zufallszahlen mit der empirischen Verteilungsfunktion $F_n(x)$ der zu testenden Zahlen, dabei ist

$$F_n(x) = \frac{\text{Anzahl der } X_1, X_2, \dots, X_n \text{ für die } \leq x}{n} \text{ und } F(x) = P(X \leq x),^6$$

wobei X_1, X_2, \dots, X_n die zu testenden n Zahlen sind.⁷ Beim Kolmogorow-Smirnow-Test wird also die Verteilung der Zahlen getestet, diese sollten gleichverteilt sein.

Zuerst werden die Pseudo-Zufallszahlen der Grösse nach sortiert, so dass $X_1 \leq X_2 \leq \dots \leq X_n$, dies erleichtert die Ermittlung von $F_n(x)$. Dann werden die beiden Werte K_n^+ und K_n^- berechnet, K_n^+ zeigt das Maximum der positiven Abweichungen der empirischen Verteilungsfunktion $F_n(x)$ von der theoretischen Verteilungsfunktion, K_n^- das Maximum der negativen Abweichungen:

$$K_n^+ = \sqrt{n} \max_{-\infty < x < +\infty} (F_n(x) - F(x));$$

$$K_n^- = \sqrt{n} \max_{-\infty < x < +\infty} (F(x) - F_n(x)),^8$$

wobei der Faktor \sqrt{n} nur wegen der Abhängigkeit der Standardabweichung von $F_n(x)$ von $1/\sqrt{n}$ da ist – diese ist $\sqrt{F(x)(1-F(x))/n}$. Durch \sqrt{n} werden K_n^+ und K_n^- also so vergrössert, dass die Standardabweichung von $F_n(x)$ nicht mehr von n abhängig ist, was die Berechnung der Wahrscheinlichkeit des Auftauchens der K_n^+ und K_n^- in Abhängigkeit von n vereinfacht.⁹

Die Werte K_n^+ und K_n^- besitzen nämlich bestimmte, von Rundungsfehlern abgesehen exakt berechenbare Wahrscheinlichkeiten, bei wirklichen Zufallszahlenfolgen aufzutreten. Damit lässt sich eine Aussage über die Hypothese machen, dass die beobachtete Verteilung der Zahlen eine zufällige sein könnte.

3.1.2. Umsetzung

Die Anzahl der zu betrachtenden Zahlen muss bei diesem Test einerseits ziemlich gross gewählt werden, um eine begründete Aussage zu machen, andererseits werden mit wachsendem n umso mehr lokale Abhängigkeiten übersehen. Taucht beispielsweise in der zu testenden Zahlenfolge eine längere Folge fast gleicher Werte auf – sagen wir, eine beliebig lange zu testende Folge von ansonsten ungefähr zwischen null und eins gleichverteilten reellen Zahlen beginne mit zehn Werten, die alle zwischen 0.09 und 0.1 schwanken – kann dies mit steigendem n zunehmend relativiert werden, da bei grossen Werten von n weitaus mehr Werte im Bereich 0.09 bis 0.1 erwartet werden dürfen als nur um die zehn. Die lokal als sehr wahrscheinlich nicht zufällig eingestufte Folge würde deshalb nicht mehr auffallen und es ergäben sich für K_n^+ und K_n^- unter Umständen sogar Werte, die für eine global gesehen zufällige Verteilung sprächen, jedoch betreffend des lokalen Verhaltens der Zahlenfolge nicht mehr aussagekräftig wären.

Ich werde deshalb den Kolmogorow-Smirnow-Test für tausend aufeinanderfolgende, $n = 1000$ Zahlen lange Pseudo-Zufallszahlenfolgen durchführen und die so erhaltenen tausend K_{1000}^+ und tausend K_{1000}^- , einem Vorschlag Knuths folgend,¹⁰ wiederum je einem Kolmogorow-Smirnow-Test unterziehen. Um Missverständnisse zu vermeiden, werde ich von nun an die K_{1000}^\pm , die man durch die Kolmogorow-Smirnow-Tests der tausend mal tausend Pseudo-Zufallszahlen selbst erhält, als „ K_{1000}^\pm erster Ebene“ bezeichnen, für die K_{1000}^\pm , die man dann durch Kolmogorow-Smirnow-Tests der Verteilung dieser „ K_{1000}^\pm erster Ebene“ erhält, werde ich von nun an den Begriff „ K_{1000}^\pm zweiter Ebene“ verwenden. (Mit diesen Begriffen gesprochen, werde ich also zuerst je tausend K_{1000}^+ und K_{1000}^- erster Ebene berechnen, dann die zwei beobachteten Verteilungen einem Kolmogorow-Smirnow-Test unterziehen und so je zwei K_{1000}^+ und K_{1000}^- zweiter Ebene erhalten.) Mit dieser Methode werde ich lokal wie auch global nicht zufälliges Verhalten zu einem gewissen Grade erkennen können.

Da ich überprüfen will, ob die von der Uniform-Funktion generierten Pseudo-Zufallszahlen reelle – eigentlich sind es nur rationale – zwischen null und eins gleichverteilte Zahlen sind, muss ich als theoretische Verteilungsfunktion $F(x) = P(X \leq x) = x$ verwenden.

Für den dann folgenden Kolmogorow-Smirnow-Test der Verteilung der je tausend erhaltenen K_{1000}^+ und K_{1000}^- erster Ebene verwende ich als theoretische Verteilungsfunktion die von Knuth angegebene Näherung $F(x) = 1 - e^{-2x^2}$, wobei $x \geq 0$,¹¹ zur Ermittlung der K_{1000}^+ und K_{1000}^- zweiter Ebene.

Das gesamte gerade beschriebene Testverfahren werde ich, beginnend mit `seed1 = 12345` und `seed2 = 67890` zwanzigmal durchführen, ich werde somit zwanzig Millionen aufeinanderfolgende, von der Uniform-Funktion generierte Pseudo-Zufallszahlen untersuchen.

3.1.3. Nullhypothese

Die achtzig errechneten K_{1000}^\pm zweiter Ebene, welche die Abweichung der empirischen Verteilungsfunktion der beobachteten K_{1000}^+ und K_{1000}^- erster Ebene von der Näherungsfunktion $F(x) = 1 - e^{-2x^2}$, wobei $x \geq 0$, beschreiben, treten, wie diese auch, mit bestimmten Wahrscheinlichkeiten auf. Man kann deshalb natürlich auch eine Wahrscheinlichkeit dafür angeben, dass ein Wert kleiner oder gleich einem beliebigen K_n^+ oder K_n^- ist. Werte für K_n^+ oder K_n^- können in Abhängigkeit von n und dieser Wahrscheinlichkeit für $n > 30$ näherungsweise wie folgt berechnet werden:

$$K_{n,p}^\pm = y_p - \frac{1}{6\sqrt{n}} + O(1/n), \text{ wobei } y_p^2 = \frac{1}{2} \ln(1/(1-p)), \text{ wobei } p = P(K_{1000}^\pm \leq x).^{12}$$

(Dies ist natürlich eine Näherung für eine Umkehrfunktion von $F(x) = 1 - e^{-2x^2}$, wobei $x \geq 0$, die ja wiederum eine Näherung für $P(K_{1000}^\pm \leq x)$ ist.) Ich werde hier einige Werte für $n = 1000$ angeben:

$P(K_{1000}^{\pm} \leq x)$	0.01	0.05	0.25	0.5	0.75	0.95	0.99
x	0.07	0.15	0.37	0.58	0.83	1.22	1.51

Tabelle 1: Ausgewählte Punkte der Wahrscheinlichkeitsverteilung von K_{1000}^+ und K_{1000}^-

Aufgrund dieser Überlegungen werde ich folgende Nullhypothese H_0 prüfen: Die Verteilung der vier mal zwanzig K_{1000}^{\pm} zweiter Ebene entspricht der Verteilung, die näherungsweise durch die Dichtefunktion $f(x) = 4x \cdot e^{-2x^2}$, wobei $x \geq 0$, beschrieben wird.

Ob H_0 zutrifft, überprüfe ich anhand der Anzahl der K_{1000}^{\pm} zweiter Ebene, die ausserhalb des Bereiches $[0.07;1.51]$ liegen, in dem sie, falls H_0 zutreffen würde, mit einer Wahrscheinlichkeit von 0.98 zu finden wären (siehe Tabelle 1). Da ich als Irrtumswahrscheinlichkeit $\alpha = 0.06$ wähle, darf jeweils maximal eine der zwanzig Stichproben ausserhalb des Bereiches $[0.07;1.51]$ liegen, denn es ist

$$\sum_{k=2}^{20} \binom{20}{k} \cdot (0.02)^k \cdot (1 - 0.02)^{20-k} = 0.0599 \leq \alpha.$$

Der Ablehnungsbereich ist somit $K = [2;20]$.

3.1.4. Programm

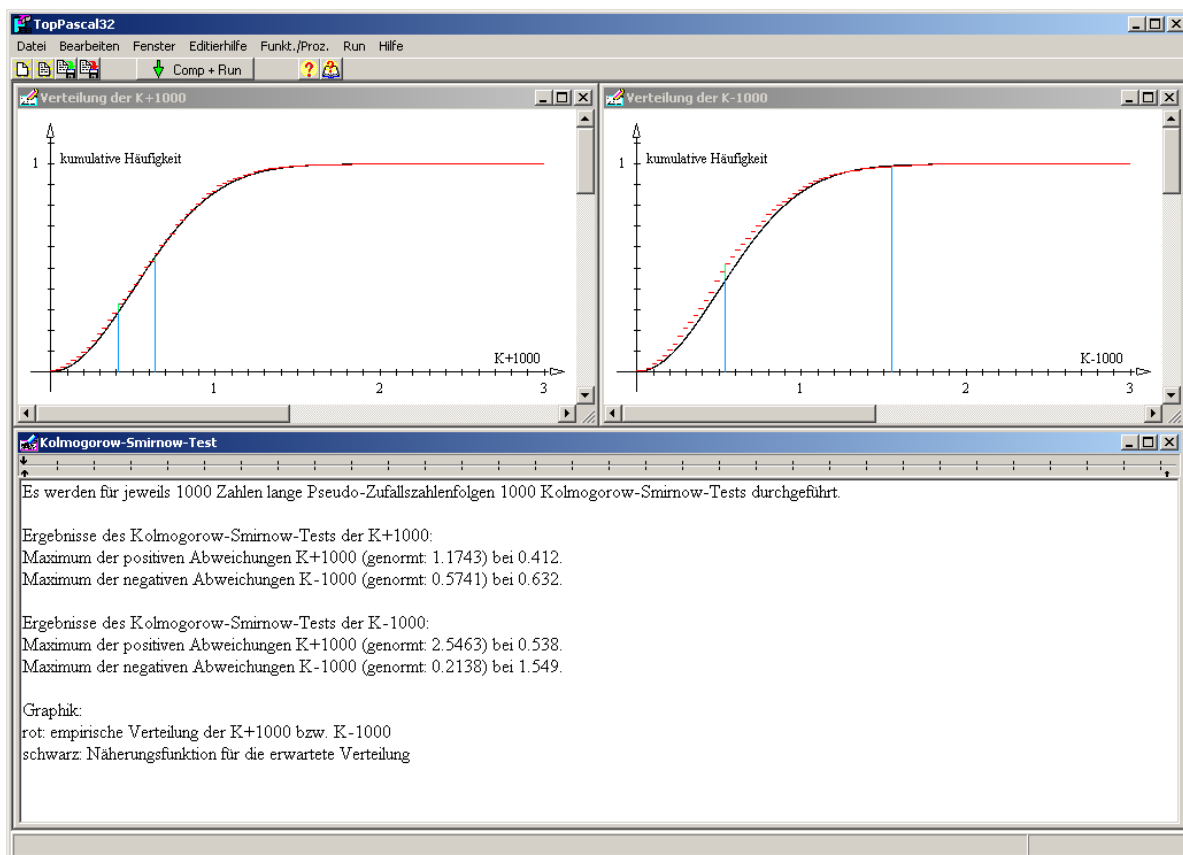


Abbildung 1: Ausgabe des beschriebenen Programms kolmogorowsmirnowtest, wobei ich jedoch obergrenze = 3 gewählt habe; der Ausdruck „genormt“ soll darauf hinweisen, dass die K_{1000}^{\pm} -Werte mit \sqrt{n} multipliziert wurden.

```

program kolmogorowsmirnowtest;

const anzkstests = 1000; anzzahlen = 1000; obergrenze = 5 {Der abschliessende Kolmogorow-Smirnow-Test wird nur für K+anzzahlen und K-anzzahlen zwischen 0 und obergrenze durchgeführt. Ist ein beobachteter Wert grösser als obergrenze, erscheint eine Warnmeldung.}; seed1begin = 12345; seed2begin = 67890;

type arrayks = array[1..anzkstests] of real;

var il, iposmaxkst {Index des höchsten K+anzkstests der Kolmogorow-Smirnow-Tests}, inegmaxkst, seedlend, seed2end, seed1temp, seed2temp : integer;
    posmaxz {K+anzzahlen der Pseudo-Zufallszahlen}, negmaxz, posmaxkst {K+anzkstests der Kolmogorow-Smirnow-Tests}, negmaxkst : real;
    textfensterbez : string;
    zufallszahl : array[1..anzzahlen] of real;
    posmaxlist {Liste der K+anzzahlen der Pseudo-Zufallszahlen}, negmaxlist : arrayks;
    Fn {empirische Verteilungsfunktion der Zufallszahlen, K+anzzahlen bzw. K-anzzahlen}: array[1..obergrenze*anzkstests] of real;

{_____}

procedure zahlenaufnahme;
var il, k, s1, s2, z : integer;

begin
    s1 := seed1temp;
    s2 := seed2temp;
    for il := 1 to anzzahlen do
        begin
            k := s1 div 53668;
            s1 := 40014 * (s1 - k * 53668) - k * 12211;
            if s1 < 0 then s1 := s1 + 2147483563;

            k := s2 div 52774;
            s2 := 40692 * (s2 - k * 52774) - k * 3791;
            if s2 < 0 then s2 := s2 + 2147483399;

            z := s1 - s2;
            if z < 1 then z := z + 2147483562;
            zufallszahl[i1] := z * 4.656613059555*10(-10);
        end;
        seedlend := s1;
        seed2end := s2;
    end;

{_____}

procedure sortmaxz; {Die Pseudo-Zufallszahlen werden ihrer Grösse nach aufsteigend geordnet.}
var il, i2, imax : integer;
    temp : real;

begin
    for il := anzzahlen downto 2 do
        begin
            imax := 1;
            for i2 := 2 to il do
                if zufallszahl[i2] > zufallszahl[imax] then
                    imax := i2;
            temp := zufallszahl[imax];
            zufallszahl[imax] := zufallszahl[i1];
            zufallszahl[i1] := temp;
        end;
    end;

{_____}

procedure kstestz; {Ein Kolmogorow-Smirnow-Test wird für anzzahlen Pseudo-Zufallszahlen durchgeführt; Ausgabe: K+anzzahlen und K-anzzahlen.}
var il, i2 : integer;
    K : array[1..anzzahlen] of real;

```

```

begin
  sortmaxz;

  {Berechnung der empirischen Verteilungsfunktion Fn}
  for il := 1 to anzzahlen do
    Fn[i1] := 0;
  for il := 1 to anzzahlen do
    for i2 := 1 to anzzahlen do
      if zufallszahl[i2] <= il/anzzahlen then
        Fn[i1] := i2/anzzahlen;

  {Berechnung von Fn(x)-F(x)}
  for il := 1 to anzzahlen do
    K[i1] := Fn[i1]-il/anzzahlen;

  {Ermittlung von K+anzahlen und K-anzahlen}
  posmaxz := K[1];
  negmaxz := K[1];
  for il := 2 to anzzahlen do
    begin
      if K[i1] > posmaxz then
        posmaxz := K[i1];
      if K[i1] < negmaxz then
        negmaxz := K[i1];
    end;
  posmaxz := sqrt(anzzahlen)*posmaxz;
  negmaxz := sqrt(anzzahlen)*negmaxz*(-1);

  {Warnmeldung}
  if posmaxz > obergrenze then
    writeln('K+',anzzahlen,' (' ,posmaxz:2:2,' ) > Obergrenze. Obergrenze muss neu gewählt
    werden. ');
  if negmaxz > obergrenze then
    writeln('K-',anzzahlen,' (' ,negmaxz:2:2,' ) > Obergrenze. Obergrenze muss neu gewählt
    werden. ');
  end;

  {
  function ksn(x:real):real; {Näherungsfunktion für die Verteilung der K+anzahlen bzw. K-
  anzzahlen}
  begin
    ksn := 1-exp(-2*(x)^2);
  end;

  {
  procedure sortmaxkst(a:arrayks;vorzeichen:string); {a entsprechend werden die
  K+anzahlen bzw. K-anzahlen ihrer Grösse nach aufsteigend geordnet.}
  var   il,i2,imax : integer;
        temp : real;

  begin
    for il := anzkstests downto 2 do
      begin
        imax := 1;
        for i2 := 2 to il do
          if a[i2] > a[imax] then
            imax := i2;
        temp := a[imax];
        a[imax] := a[il];
        a[il] := temp;
      end;

    if vorzeichen = '+' then
      for il := 1 to anzkstests do
        posmaxlist[il] := a[il]
      else
        for il := 1 to anzkstests do
          negmaxlist[il] := a[il];
    end;

  {

```

```

procedure kstestkst(kwert:arrayks); {Ein Kolmogorow-Smirnow-Test wird für anzkstests
K+anzzahlen oder K-anzzahlen durchgeführt, dabei wird nur die Verteilung für Werte
zwischen 0 und obergrenze betrachtet; Ausgabe: K+anzkstests, K-anzkstests und bei
welchem getesteten Wert sie auftraten.}
var   il,i2 : integer;
        K : array[1..obergrenze*anzkstests] of real;

begin
  {Berechnung der empirischen Verteilungsfunktion Fn}
  for il := 1 to obergrenze*anzkstests do
    Fn[il] := 0;
  for il := 1 to obergrenze*anzkstests do
    for i2 := 1 to anzkstests do
      if kwert[i2] <= il/anzkstests then
        Fn[il] := i2/anzkstests;

  {Berechnung von Fn(x)-F(x)}
  for il := 1 to obergrenze*anzkstests do
    K[il] := Fn[il]-ksn(il/anzkstests);

  {Ermittlung von K+anzkstests und K-anzkstests}
  posmaxkst := K[1];
  negmaxkst := K[1];
  iposmaxkst := 1;
  inegmaxkst := 1;
  for il := 2 to obergrenze*anzkstests do
    begin
      if K[il] > posmaxkst then
        begin
          posmaxkst := K[il];
          iposmaxkst := il;
        end;
      if K[il] < negmaxkst then
        begin
          negmaxkst := K[il];
          inegmaxkst := il;
        end;
    end;
  posmaxkst := sqrt(anzkstests)*posmaxkst;
  negmaxkst := sqrt(anzkstests)*negmaxkst*(-1);
end;

{_____}

procedure graphik(vorzeichen:string);
var   drawfensterbez : string;

begin
  {Graphikfenster}
  drawfensterbez := 'Verteilung der K'+vorzeichen+stringof(anzzahlen);
  showDrawing(drawfensterbez);
  clearDrawing;
  if vorzeichen = '+' then
    setdrawingrect(0,0,510,300);
  if vorzeichen = '-' then
    setdrawingrect(510,0,1020,300);

  {Achsen und Beschriftung}
  uRGBBlack;
  uSetXAchse(-0.04*obergrenze,1.04*obergrenze);
  uSetYAchse(-0.1,1.2);
  uDrawKSXY(0.1,0.1);
  uScaleKSXY(1,1,0,0,times,9);

  uPenUp;
  uSetPos(0.9*obergrenze,0.04);
  writeDraw('K',vorzeichen,anzzahlen);
  uSetPos(0.02*obergrenze,1);
  writeDraw('kumulative Häufigkeit');

  {Näherungsfunktion für die Verteilung der K+anzzahlen bzw. K-anzzahlen}
  for il := 1 to obergrenze*anzkstests do
    uLine(0,(il-1)/anzkstests,ksn((il-1)/anzkstests),il/anzkstests,ksn(il/anzkstests));

```

```

{empirische Verteilungsfunktion der K+anzzahlen bzw. K-anzzahlen}
uRGBColor(1,0,0);
for il := 1 to obergrenze*anzkstests do
  uLine(0,(il-1)/anzkstests,Fn[il],il/anzkstests,Fn[il]);

{Abweichungen}
uRGBColor(0,1,0.4);
uLine(0,inegmaxkst/anzkstests,Fn[inegmaxkst],inegmaxkst/anzkstests,ksn(inegmaxkst/
anzkstests));
uLine(0,iposmaxkst/anzkstests,Fn[iposmaxkst],iposmaxkst/anzkstests,ksn(iposmaxkst/
anzkstests));
uRGBColor(0,0.6,1);
uLine(0,inegmaxkst/anzkstests,0,inegmaxkst/anzkstests,Fn[inegmaxkst]);
uLine(0,iposmaxkst/anzkstests,0,iposmaxkst/anzkstests,ksn(iposmaxkst/anzkstests));
end;

{_____Hauptprogramm_____}

begin
  {Textfenster}
  textfensterbez := 'Kolmogorow-Smirnow-Test';
  showText(textfensterbez);
  clearText;
  setTextRect(0,300,1020,650);
  textFont(1);
  textSize(12);
  writeln('Es werden für jeweils ', anzzahlen, ' Zahlen lange Pseudo-Zufallszahlenfolgen
', anzkstests, ' Kolmogorow-Smirnow-Tests durchgeführt.');
```

```

  {Durchführung der anzzahlen Kolmogorow-Smirnow-Tests}
  seedltemp := seedlbegin;
  seed2temp := seed2begin;
  for il := 1 to anzkstests do
    begin
      zahlenaufnahme;
      kstestz;
      posmaxlist[il] := posmaxz;
      negmaxlist[il] := negmaxz;
      seedltemp := seedlend;
      seed2temp := seed2end;
    end;

  {Durchführung des Kolmogorow-Smirnow-Tests für die Werte der K+anzzahlen}
  sortmaxkst(posmaxlist, '+');
  kstestkst(posmaxlist);

  writeln(' ');
  writeln('Ergebnisse des Kolmogorow-Smirnow-Tests der K+', anzzahlen, ':');
  writeln('Maximum der positiven Abweichungen K+', anzkstests, ' (genormt:
', posmaxkst:4:4, ') bei ', iposmaxkst/anzkstests:3:3, '.');
  writeln('Maximum der negativen Abweichungen K-', anzkstests, ' (genormt:
', negmaxkst:4:4, ') bei ', inegmaxkst/anzkstests:3:3, '.');

  graphik('+');

  {Durchführung des Kolmogorow-Smirnow-Tests für die Werte der K-anzzahlen}
  sortmaxkst(negmaxlist, '-');
  kstestkst(negmaxlist);

  writeln(' ');
  writeln('Ergebnisse des Kolmogorow-Smirnow-Tests der K-', anzzahlen, ':');
  writeln('Maximum der positiven Abweichungen K+', anzkstests, ' (genormt:
', posmaxkst:4:4, ') bei ', iposmaxkst/anzkstests:3:3, '.');
  writeln('Maximum der negativen Abweichungen K-', anzkstests, ' (genormt:
', negmaxkst:4:4, ') bei ', inegmaxkst/anzkstests:3:3, '.');

  graphik('-');

  writeln(' ');
  writeln('Graphik:');
  writeln('rot: empirische Verteilung der K+', anzkstests, ' bzw. K-', anzkstests);
  writeln('schwarz: Näherungsfunktion für die erwartete Verteilung');
end.

```

3.2. χ^2 -Test

3.2.1. Verfahren

Der χ^2 -Test – auch χ^2 -Anpassungstest genannt – betrachtet, ähnlich wie der Kolmogorow-Smirnow-Test, die Verteilung der zu testenden Zahlen – er ist deshalb wie dieser auch ein Gleichverteilungstest.¹³

Die Zahlen der generierten Pseudo-Zufallszahlenfolge werden dazu entsprechend ihrer Grösse einer von k Kategorien zugeteilt. Ich werde die Zahl X_i – bei den von mir getesteten Zahlen wird es sich ja um rationale Zahlen zwischen null und eins handeln – der Kategorie s zuteilen, wenn

$$\frac{s-1}{k} < X_i \leq \frac{s}{k}.$$

Der Wert χ^2 ist nun folgendermassen definiert: Steht p_s für die Wahrscheinlichkeit, dass eine Zahl in Kategorie s fällt, somit das Produkt der Anzahl n der getesteten Zahlen mit p_s für die erwartete Anzahl der Zahlen in dieser Kategorie, und Y_s für die Anzahl der beobachteten Zahlen in dieser Kategorie, ist

$$\chi^2 = \sum_{s=1}^k \frac{(Y_s - n \cdot p_s)^2}{n \cdot p_s}.^{14}$$

Die Wahrscheinlichkeit für einen bestimmten Wert von χ^2 kann hier aus der Anzahl der Freiheitsgrade ν berechnet werden, die bei k Kategorien $\nu = k - 1$ beträgt – was annähernd damit erklärt werden kann, dass die Werte Y_1, Y_2, \dots, Y_k nicht völlig unabhängig voneinander sind, denn weil $Y_1 + Y_2 + \dots + Y_k = n$ gilt, lässt sich Y_k aus Y_1, Y_2, \dots, Y_{k-1} berechnen, die Anzahl der Freiheitsgrade wird also um einen Freiheitsgrad verringert.¹⁵

Die Wahrscheinlichkeit des erhaltenen χ^2 wiederum lässt eine Aussage über die Hypothese, die beobachtete Verteilung der Zahlen könne eine zufällige sein, zu.

3.2.2. Umsetzung

Da die χ^2 -Verteilung eine Näherung ist, ist der χ^2 -Test nur aussagekräftig, wenn die Anzahl der betrachteten Zahlen gross genug ist. Als Faustregel für die Grösse von n gibt Knuth an, dass n mindestens so gross sein müsse, dass für jede Klasse der Erwartungswert der Anzahl der Beobachtungen $n \cdot p_s$ mindestens fünf oder mehr betrage, ein grösseres n sei jedoch besser.

Doch obwohl grosse Werte von n klarer zeigen können, dass eine Zahlenfolge global nicht zufällig ist, macht er auch darauf aufmerksam, dass umso mehr lokale Abweichungen übersehen werden, je grösser n wird. Dies kommt daher, dass diese Abweichungen in den Kategorien, die bei gleichbleibender Breite aber steigendem n immer mehr Werte enthalten, meistens ausgeglichen werden: In einer Kategorie, die zu viele Werte enthält, kann zum Beispiel die erste Hälfte der Kategorie viel weniger beobachtete Werte und die zweite viel mehr als erwartet beinhalten, ohne dass man diese sehr wahrscheinlich nicht zufällige Verteilung am χ^2 -Wert ablesen könnte.

Einerseits muss ich also Knuths Faustregel erfüllen, andererseits sollte n nicht allzu gross sein. Ich werde deshalb, wiederum einem Vorschlag Knuths folgend,¹⁶ gleich vorgehen wie bei den Kolmogorow-Smirnow-Tests: Ich werde tausend χ^2 -Tests mit jeweils $n = 1000$ Zahlen und $k = 101$ Klassen durchführen, danach unterziehe ich die erhaltenen tausend χ^2 -Werte einem Kolmogorow-Smirnow-Test.

Ich werde also in einem Kolmogorow-Smirnow-Test die empirische Verteilungsfunktion der erhaltenen χ^2 -Werte mit deren theoretischer Verteilungsfunktion vergleichen. Diese ist

$$F_n(x) = \int_0^x \frac{x^{(l-1)} e^{-\frac{x}{2}}}{2^l \Gamma(l)} \cdot dx \text{ für } x > 0,^{17}$$

wobei $l = \frac{\nu}{2}$ und

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt,$$

in meinem Fall mit $l = 50$ kann ich den Zusammenhang $\Gamma(n) = (n-1)!$ verwenden, der für alle $n \in \mathbb{N}$ gilt.¹⁸ Es ist somit bei $l = 50$

$$F_n(x) = \int_0^x \frac{x^{(50-1)} e^{-\frac{x}{2}}}{2^{50} (50-1)!} \cdot dx \text{ für } x > 0.$$

Im Programm werde ich das Integral durch eine Summe ersetzen müssen, es ist

$$F_n(x) \approx \sum_{i=1}^m \frac{(i \cdot \Delta x)^{49} e^{-\frac{i \cdot \Delta x}{2}}}{2^{50} 49!} \cdot \Delta x \text{ für } x > 0,$$

wobei $m = \lfloor x / \Delta x \rfloor$. Ich werde dabei willkürlich mit $\Delta x = 1/1000$ rechnen. Wahrscheinlich wird das auf diese Weise ersetzte Integral durch die Summe mehr als ausreichend genau angenähert werden, doch ich habe dies nicht mit numerischen Methoden überprüft.

Auch dieses Testverfahren werde ich, mit den Startwerten $seed1 = 12345$ und $seed2 = 67890$ beginnend, zwanzigmal durchführen, es werden demnach wieder dieselben zwanzig Millionen Pseudo-Zufallszahlen untersucht wie in den in Kapitel 3.1. beschriebenen Kolmogorow-Smirnow-Tests.

3.2.3. Nullhypothese

Ich werde letztendlich je zwanzig Werte für K_{1000}^+ und K_{1000}^- erhalten. Da ich auch hier $n = 1000$ Beobachtungen betrachten werde, gelten für die Erwartungswerte die gleichen Feststellungen wie für die unter 3.1. beschriebenen Kolmogorow-Smirnow-Tests.

Ich kann deshalb die gleiche Nullhypothese H_0 wie unter 3.1.3. aufstellen: Die Verteilung der zwanzig K_{1000}^\pm entspricht der Verteilung, die näherungsweise durch die Dichtefunktion $f(x) = 4x \cdot e^{-2x^2}$, wobei $x \geq 0$, beschrieben wird.

Da ich als Irrtumswahrscheinlichkeit wieder $\alpha = 0.06$ wähle und dabei wieder die Anzahl der K_{1000}^\pm , die ausserhalb des Bereiches $[0.07; 1.51]$ liegen, betrachte, darf jeweils maximal

eine der zwanzig Stichproben ausserhalb des Bereiches $[0.07;1.51]$ liegen (siehe Tabelle 1), der Ablehnungsbereich ist somit $K = [2;20]$.

3.2.4. Programm

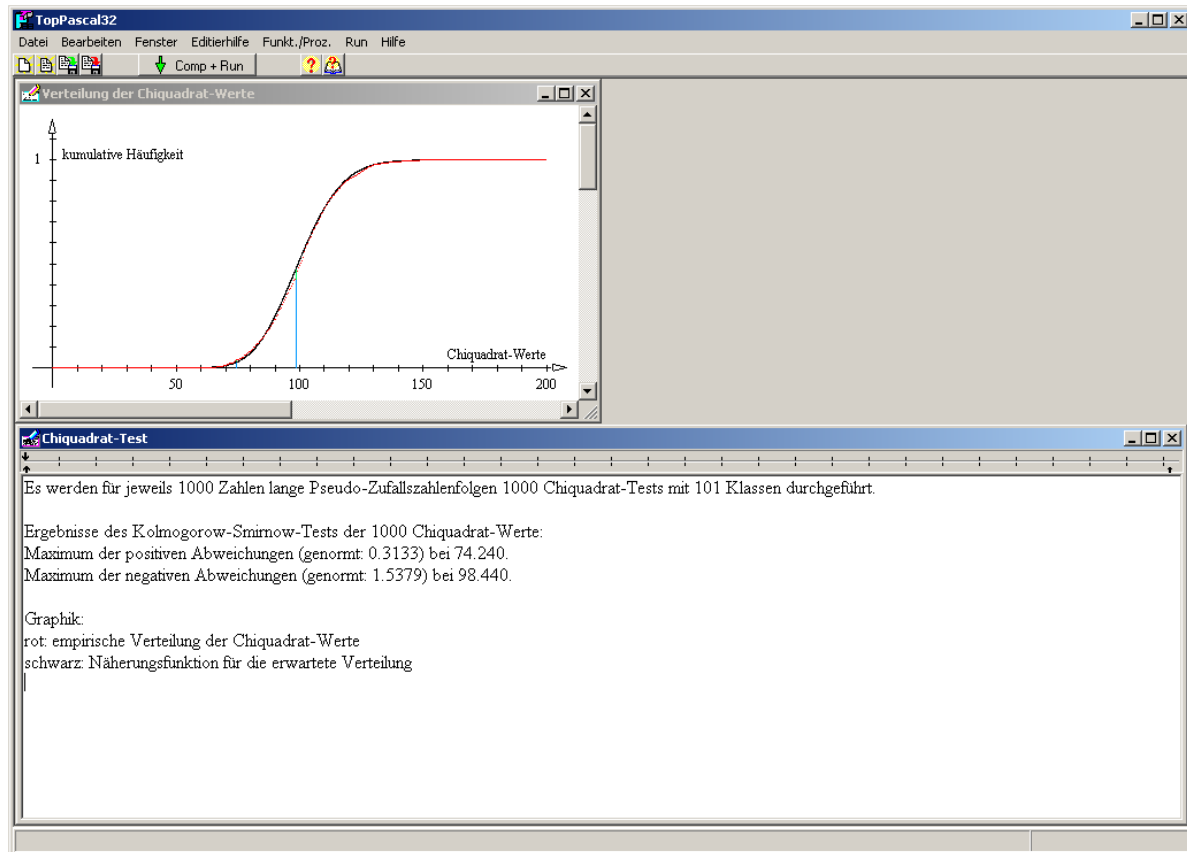


Abbildung 2: Ausgabe des beschriebenen Programms chiquadrattest; der Ausdruck „genormt“ weist darauf hin, dass die K_{1000}^{\pm} -Werte mit \sqrt{n} multipliziert wurden.

```

program chiquadrattest;

const anzcqttests = 1000; anzklassen = 101; anzzahlen = 1000; dxcqnb = 1/1000; dxcqng =
1/100 {deltax der Summe für die Näherungsfunktion für die Verteilung der Chiquadrat-
Werte für die Berechnungen beziehungsweise die Graphik}; obergrenze = 200 {Der
Kolmogorow-Smirnow-Test wird nur für Chiquadrat-Werte zwischen 0 und obergrenze
durchgeführt. Ist ein Chiquadrat-Wert grösser als obergrenze, erscheint eine
Warnmeldung.}; seedlbegin = 12345; seed2begin = 67890; tt = 40 {Da bei "array[1..x] of
real" x < 6554 sein muss, kann die empirische Verteilungsfunktion Fn der Chiquadrat-
Werte nicht ganz so genau berechnet werden, wie ohne diese Beschränkung, allerdings ist
der dadurch entstehende Fehler sehr klein. Je grösser tt gewählt wird, umso ungenauer
wird die Berechnung, dabei muss bei 1000 Chiquadrat-Tests und einer Obergrenze von 200
tt > 30 (200*1000/31 < 6554) und ein Teiler von 1000 sein.};

var il, iposmaxcqt {Index des K+anzcqttests der Chiquadrat-
Werte}, inegmaxcqt, seedlend, seed2end, seedltemp, seed2temp : integer;
chiquadrat, posmaxcqt {K+anzcqttests der Chiquadrat-Werte}, negmaxcqt : real;
drawfensterbez, textfensterbez : string;
chiquadratlist : array[1..anzcqttests] of real;
zufallszahl : array[1..anzzahlen] of real;
Fn {empirische Verteilungsfunktion der Chiquadrat-Werte}:
array[1..obergrenze*anzcqttests div tt] of real;

{
}

procedure zahlenaufnahme;
var il, k, s1, s2, z : integer;

```

```

begin
  s1 := seed1temp;
  s2 := seed2temp;
  for i1 := 1 to anzzahlen do
    begin
      k := s1 div 53668;
      s1 := 40014 * (s1 - k * 53668) - k * 12211;
      if s1 < 0 then s1 := s1 + 2147483563;

      k := s2 div 52774;
      s2 := 40692 * (s2 - k * 52774) - k * 3791;
      if s2 < 0 then s2 := s2 + 2147483399;

      z := s1 - s2;
      if z < 1 then z := z + 2147483562;
      zufallszahl[i1] := z * 4.656613059555*10(-10);
    end;
  seedlend := s1;
  seed2end := s2;
end;

{ _____ }

procedure cqtest; {Ein Chiquadrat-Test wird für anzzahlen Pseudo-Zufallszahlen
durchgeführt; Ausgabe: Chiquadrat-Wert.}
var i1,i2 : integer;
    anzelementeinklasse : array[1..anzklassen] of integer;

begin
  chiquadrat := 0;
  for i1 := 1 to anzklassen do
    anzelementeinklasse[i1] := 0;

    {Einteilung der Pseudo-Zufallszahlen in anzklassen Klassen}
    for i1 := 1 to anzzahlen do
      for i2 := 1 to anzklassen do
        if (zufallszahl[i1] > (i2-1)/anzklassen) and (zufallszahl[i1] <= i2/anzklassen)
then
          anzelementeinklasse[i2] := anzelementeinklasse[i2] + 1;

        {Berechnung des Chiquadrat-Wertes}
        for i1 := 1 to anzklassen do
          chiquadrat := chiquadrat + (anzelementeinklasse[i1]-
anzzahlen/anzklassen)2/(anzzahlen/anzklassen);

        {Warnmeldung}
        if chiquadrat > obergrenze then
          writeln('Chiquadrat-Wert (' ,chiquadrat:1:1,') > Obergrenze. Obergrenze muss neu
gewählt werden. ');
        end;
      end;
    end;

  { _____ }

function fakultaet(x:integer):real;
var i1 : integer;
    f : real;

begin
  f := 1;
  for i1 := 1 to x do
    f := f*i1;
  fakultaet := f;
end;

{ _____ }

function cqn(x:real;dx:cqn:real):real; {Näherungsfunktion für die Verteilung der
Chiquadrat-Werte}
var i1 : integer;
    b,cqnl : real;

begin
  cqnl := 0;

```

```

b := 2^((anzklassen-1)/2)*fakultaet(((anzklassen-1) div 2)-1);
for il := 1 to round(x/dxcqn) do
  cqnl := cqnl+(il*dxcqn)^((anzklassen-1)/2-1)*exp(-(il*dxcqn)/2)/b*dxcqn;
cqnl := cqnl;
end;

{_____}

procedure sortmaxcqt; {Die Chiquadrat-Werte werden ihrer Grösse nach aufsteigend geordnet.}
var   il,i2,imax : integer;
       temp       : real;

begin
  for il := anzcqttests downto 1 do
    begin
      imax := 1;
      for i2 := 2 to il do
        if chiquadratlist[i2] > chiquadratlist[imax] then
          imax := i2;
        temp := chiquadratlist[imax];
        chiquadratlist[imax] := chiquadratlist[i1];
        chiquadratlist[i1] := temp;
      end;
    end;
end;

{_____}

procedure kstestcqt; {Ein Kolmogorow-Smirnow-Test wird für anzcqttests Chiquadrat-Werte durchgeführt, dabei wird nur die Verteilung für Werte zwischen 0 und obergrenze betrachtet; Ausgabe: empirische Verteilungsfunktion Fn, K+anzcqttests, K-anzcqttests und bei welchem getesteten Wert sie auftraten.}
var   il,i2 : integer;
       K      : array[1..obergrenze*anzcqttests div tt] of real;

begin
  sortmaxcqt;

  {Berechnung der empirischen Verteilungsfunktion Fn}
  for il := 1 to obergrenze*anzcqttests div tt do
    Fn[il] := 0;
  for il := 1 to obergrenze*anzcqttests div tt do
    for i2 := 1 to anzcqttests do
      if chiquadratlist[i2] <= il/(anzcqttests div tt) then
        Fn[il] := i2/anzcqttests;

  {Berechnung von Fn(x)-F(x)}
  for il := 1 to obergrenze*anzcqttests div tt do
    K[il] := Fn[il]-cqnl(il/(anzcqttests div tt),dxcqnb);

  {Ermittlung von K+anzcqttests und K-anzcqttests}
  posmaxcqt := K[1];
  negmaxcqt := K[1];
  iposmaxcqt := 1;
  inegmaxcqt := 1;
  for il := 2 to obergrenze*anzcqttests div tt do
    begin
      if K[il] > posmaxcqt then
        begin
          posmaxcqt := K[il];
          iposmaxcqt := il;
        end;
      if K[il] < negmaxcqt then
        begin
          negmaxcqt := K[il];
          inegmaxcqt := il;
        end;
    end;
  posmaxcqt := sqrt(anzcqttests)*posmaxcqt;
  negmaxcqt := sqrt(anzcqttests)*negmaxcqt*(-1);
end;

```

```

{ _____ Hauptprogramm _____ }

begin
  {Textfenster}
  textfensterbez := 'Chiquadrat-Test';
  showText(textfensterbez);
  clearText;
  setTextRect(0,300,1020,650);
  textFont(1);
  textSize(12);

  writeln('Es werden für jeweils ',anzzahlen,' Zahlen lange Pseudo-Zufallszahlenfolgen
',anzcqtests,' Chiquadrat-Tests mit ',anzklassen,' Klassen durchgeführt.');
```

{Durchführung der anzcqtests Chiquadrat-Tests}

```

seed1temp := seed1begin;
seed2temp := seed2begin;
for il := 1 to anzcqtests do
  begin
    zahlenaufnahme;
    cqtest;
    chiquadratlist[il] := chiquadrat;
    seed1temp := seedlend;
    seed2temp := seed2end;
  end;

{Durchführung des Kolmogorow-Smirnow-Tests der Chiquadrat-Werte}
kstestcq;

writeln(' ');
writeln('Ergebnisse des Kolmogorow-Smirnow-Tests der ',anzcqtests,' Chiquadrat-
Werte:');
writeln('Maximum der positiven Abweichungen (genormt: ',posmaxcq:4:4,' ) bei
',iposmaxcq/(anzcqtests div tt):3:3, '. ');
writeln('Maximum der negativen Abweichungen (genormt: ',negmaxcq:4:4,' ) bei
',inegmaxcq/(anzcqtests div tt):3:3, '. ');

{ _____ Graphik _____ }

{Graphikfenster}
drawfensterbez := 'Verteilung der Chiquadrat-Werte';
showDrawing(drawfensterbez);
clearDrawing;
setDrawingRect(0,0,510,300);

{Achsen und Beschriftung}
uRGBBlack;
uSetXAchse(-0.04*obergrenze,1.04*obergrenze);
uSetYAchse(-0.1,1.2);
uDrawKSXY(10,0.1);
uScaleKSXY(50,1,0,0,times,9);

uPenUp;
uSetPos(0.8*obergrenze,0.04);
writeDraw('Chiquadrat-Werte');
uSetPos(0.02*obergrenze,1);
writeDraw('kumulative Häufigkeit');
```

{Näherungsfunktion}

```

for il := 1 to obergrenze*anzcqtests div tt do
  uLine(0,(il-1)/(anzcqtests div tt),cqn((il-1)/(anzcqtests div
tt),dxcqng),il/(anzcqtests div tt),cqn(il/(anzcqtests div tt),dxcqng));

{empirische Verteilungsfunktion der Chiquadrat-Werte}
uRGBColor(1,0,0);
for il := 1 to obergrenze*anzcqtests div tt do
  uLine(0,(il-1)/(anzcqtests div tt),Fn[il],il/(anzcqtests div tt),Fn[il]);

{Abweichungen}
uRGBColor(0,1,0.4);
uLine(0,inegmaxcq/(anzcqtests div tt),Fn[inegmaxcq],inegmaxcq/(anzcqtests div
tt),cqn(inegmaxcq/(anzcqtests div tt),dxcqng));
uLine(0,iposmaxcq/(anzcqtests div tt),Fn[iposmaxcq],iposmaxcq/(anzcqtests div
tt),cqn(iposmaxcq/(anzcqtests div tt),dxcqng));

```

```
    uRGBColor(0,0.6,1);
    uLine(0,inegmaxcqt/(anzcqttests div tt),0,inegmaxcqt/(anzcqttests div
tt),Fn[inegmaxcqt]);
    uLine(0,iposmaxcqt/(anzcqttests div tt),0,iposmaxcqt/(anzcqttests div
tt),cqn(iposmaxcqt/(anzcqttests div tt),dxcqng));

    writeln(' ');
    writeln('Graphik:');
    writeln('rot: empirische Verteilung der Chiquadrat-Werte');
    writeln('schwarz: Näherungsfunktion für die erwartete Verteilung');
end.
```

3.3. Run-Test

3.3.1. Verfahren

Es gibt viele verschiedene Arten, einen Run und das Verfahren eines Run-Tests zu beschreiben. Bei einem Run-Test, wie ich ihn beschreiben werde, wird das Monotonieverhalten, das heisst das Auftreten von monoton wachsenden oder fallenden Blöcken innerhalb der gesamten zu testenden Zahlenfolge, betrachtet – somit ist der Run-Test ein Unabhängigkeitstest.¹⁹

Allgemein kann ein Run definiert werden als „eine Folge von identischen Ereignissen. Sie wird durch eine neue Klasse von Ereignissen beendet“. Dabei ist die Länge des Runs „die Anzahl von [...] Ereignissen, die in einem Run auftreten.“²⁰

In einem „Run up“-Test liegt die Aufmerksamkeit beim Ereignis „ $X_j \leq X_{j+1}$ “, wobei X_j und X_{j+1} zwei Pseudo-Zufallszahlen der Folge X_1, X_2, \dots, X_n sind. Ein Run wird also durch das Ereignis „ $X_j > X_{j+1}$ “ oder dadurch, dass der Index $j = n$ ist, abgeschlossen. (Ein Beispiel: Die Folge 4, 5, 5, 9, 7, 8 wird diesen Regeln entsprechend in folgende Runs unterteilt: 4, 5, 5, 9 bilden einen „Run up“ der Länge vier, während der Block 7, 8 einen „Run up“ der Länge zwei darstellt.) Ist ein Run abgeschlossen, wird seine Länge ermittelt und um den wievielten Run der betreffenden Länge es sich in der betrachteten Zahlenfolge gehandelt hat.

In einem „Run down“-Test wird das Ereignis „ $X_j \geq X_{j+1}$ “ betrachtet, ein Run also durch das Ereignis „ $X_j < X_{j+1}$ “ oder durch $j = n$ abgeschlossen.

Man kann natürlich berechnen, mit welcher Wahrscheinlichkeit wie viele Runs einer bestimmten Länge in einer Zufallszahlenfolge auftreten. Die beobachtete Anzahl Runs einer bestimmten Länge lässt somit eine Aussage über die Hypothese, es könne sich bei der Zahlenfolge um eine Folge von Zufallszahlen handeln, zu.

3.3.2. Umsetzung

Knuth schreibt, dass, wenn die jeweils erste Zahl X_{j+1} , die einem gerade bei der vorhergehenden Zahl X_j abgeschlossenen Run folgt, nicht beachtet wird, sondern der neue Run erst bei X_{j+2} beginnt, die Runlängen voneinander unabhängig sind. Deshalb kann mit den Werten, die angeben, wie viele Runs einer Länge beobachtet wurden, ein χ^2 -Test durchgeführt werden.²¹ Mein Ziel ist es darum nun, zu berechnen, wie viele „Runs up“ oder „Runs down“ einer bestimmten Länge bei Zufallszahlenfolgen einer bestimmten Länge zu erwarten sind.

Knuth gibt an, dass, werden die Runs so gezählt wie gerade beschrieben, Runs der Länge r in einer Zufallszahlenfolge beliebiger Länge mit der Wahrscheinlichkeit $P = r/(r+1)!$ auftauchen.²² Ich werde diese Formel nicht beweisen, aber ich will für „Runs up“ der Länge eins und zwei zeigen, wie man auf einem anderen Weg – dieser lässt sich vermutlich auch auf alle anderen Längen anwenden – zu den gleichen Werten kommt, wie sie die Formel von Knuth für diese Längen angibt:

Ein „Run up“ hat dann die Länge eins, wenn der betrachteten Zahl eine kleinere Zahl folgt, es gilt $X_{k+1} < X_k$.

Man geht nun folgendermassen vor: Man unterteilt den Bereich von null bis eins, in den die Zufallszahlen fallen können, in insgesamt m Bereiche der Breite Δx , deren untere Grenze jeweils x_i ist, wobei $i = 1, 2, \dots, m$. Dann betrachtet man die Bedingungen für einen „Run up“ der Länge eins bezüglich dieser Bereiche, man versucht also, diese Bedingungen in Abhängigkeit von x_i und Δx zu bestimmen. Sobald sie bekannt sind und man damit auch die Wahrscheinlichkeit für ihre Erfüllung für jeden Bereich berechnen kann, muss man nur noch diese Wahrscheinlichkeiten, die für die einzelnen Bereiche gelten, aufsummieren, um die Wahrscheinlichkeit eines „Run up“ der Länge eins für den ganzen Bereich zu kennen.

Die Bedingungen für einen „Run up“ der Länge eins lassen sich nun bezüglich x_i und Δx wie folgt bestimmen: Für den Wert X_{k+1} muss $X_{k+1} < x_i$ gelten, für den Wert X_k dagegen $x_i < X_k < x_i + \Delta x$: X_{k+1} muss sich unterhalb des Bereiches der Breite Δx , der bei x_i beginnt, befinden, während X_k innerhalb dieses Bereiches sein muss. Die Wahrscheinlichkeiten dafür sind $P(X_{k+1} < x_i) = x_i$ und $P(x_i < X_k < x_i + \Delta x) = \Delta x$. Da die beiden Ereignisse voneinander unabhängig sind, gilt

$$P(X_{k+1} < x_i \cap x_i < X_k < x_i + \Delta x) = P(X_{k+1} < x_i) \cdot P(x_i < X_k < x_i + \Delta x) = x_i \cdot \Delta x.$$

Es mag scheinen, die Bedingung $x_i < X_k < x_i + \Delta x$ könne durch die Bedingung $X_k > x_i$ ersetzt werden, auch wenn X_k nicht kleiner sei als $x_i + \Delta x$, erfülle dies die Bedingung für einen „Run up“ der Länge eins. Diese Vereinfachung ist aber nicht zulässig, denn wenn sich auf diese Weise $X_k > x_i + \Delta x$ ergäbe, hätte die Bedingung $X_{k+1} < x_i$ wenig Sinn, da in diesem Fall auch die Bedingung $X_{k+1} < x_i + \Delta x$ für einen „Run up“ der Länge eins genügen würde.

Argumentiert man so, ist es aber auch nicht zulässig, dass der Bereich oberhalb x_i eine gewisse Breite Δx haben soll: Falls beispielsweise $x_i + \Delta x/2 < X_k < x_i + \Delta x$ gelten würde, sich also X_k in der oberen Hälfte des betroffenen Bereiches der Breite Δx befände, müsste für X_{k+1} nur noch $X_{k+1} < x_i + \Delta x/2$ gelten anstatt $X_{k+1} < x_i$. Aufgrund dieser Überlegung verkleinert man die Breite Δx , man lässt Δx gegen null gehen.

Somit hat man die Wahrscheinlichkeiten für das Auftauchen eines „Run up“ der Länge eins in Abhängigkeit von x_i und Δx bestimmt. Man summiert nun die Wahrscheinlichkeiten für die verschiedenen Bereiche, die verschiedenen Werte von x_i auf.

$$P(X_k > X_{k+1}) \approx \sum_{i=1}^m x_i \cdot \Delta x$$

Berücksichtigt man $\Delta x \rightarrow 0$, womit auch $m \rightarrow \infty$, wird diese Summe zu einem Integral. Man erhält

$$P(X_k > X_{k+1}) = \int_0^1 x \cdot dx = 1/2,$$

was gleich $1/(1+1)!$ ist, also die Formel Knuths bestätigt.

Das gleiche Verfahren auf einen „Run up“ der Länge zwei anzuwenden, ist ein wenig schwieriger, doch lässt sich dann daraus leicht auf andere Runlängen schliessen.

Die Bedingungen für einen „Run up“ der Länge zwei sind $X_k \leq X_{k+1}$ und $X_{k+1} > X_{k+2}$. In einem ersten Schritt geht es darum, die Bedingung $X_{k+1} > X_{k+2}$ zu erfüllen, allerdings

muss dabei auch auf die Bedingung $X_k \leq X_{k+1}$ Rücksicht genommen werden. Wieder unterteilt man einen Bereich in kleinere Bereiche der Breite Δx , deren untere Grenze jeweils x_i ist, wobei $i = 1, 2, \dots, n$. Doch diesmal unterteilt man nur einen Bereich von y , einem konstanten Wert zwischen null und eins, bis eins, da $X_{k+1} > y$ erfüllt werden soll, somit wird also $X_k \leq X_{k+1}$ berücksichtigt.

Es müssen nun zuerst einmal, in Abhängigkeit von x_i und Δx angegeben, folgende Bedingungen für einen „Run up“ der Länge zwei gelten: Es muss, ähnlich wie bei einem „Run up“ der Länge eins zum Abschluss des Runs $X_{k+2} < x_i$ und $x_i < X_{k+1} < x_i + \Delta x$ sein. Es ist

$$P(X_{k+2} < x_i \cap x_i < X_{k+1} < x_i + \Delta x) = P(X_{k+2} < x_i) \cdot P(x_i < X_{k+1} < x_i + \Delta x) = x_i \cdot \Delta x.$$

Aufgrund der gleichen Überlegungen wie oben summiert man diese Wahrscheinlichkeiten auf und lässt dabei wiederum Δx gegen null gehen, womit $n \rightarrow \infty$ einhergeht. Auf diese Weise erhält man

$$P(y < X_{k+1} \cap X_{k+1} > X_{k+2}) = \int_y^1 x \cdot dx.$$

Man unterteilt nun den Bereich von null bis eins in Bereiche der Breite Δy , die jeweils die Untergrenze y_j haben, wobei $j = 1, 2, \dots, m$. Die Bedingungen für einen „Run up“ der Länge zwei sind erfüllt, wenn sich X_k im gewünschten Bereich aufhält: Es muss $y_j < X_k < y_j + \Delta y$ gelten. Dies scheint widersprüchlich zu sein: X_k muss sich zwischen y_j und $y_j + \Delta y$ aufhalten, während es für X_{k+1} genügt, nur grösser als y_j zu sein. Man erwartet aber, X_{k+1} müsse nicht nur grösser als y_j sondern auch grösser als $y_j + \Delta y$ sein. Doch dieser Widerspruch verschwindet, wenn man den Bereich, in welchem sich X_k aufhalten muss, schrittweise verkleinert, wenn man Δy gegen null gehen lässt, da bekanntlich $\lim_{\Delta y \rightarrow 0} y_j + \Delta y = y_j$. Damit ist also

$$P(y_j < X_{k+1} \cap X_{k+1} > X_{k+2}) \cdot P(y_j < X_k < y_j + \Delta y) = P(y_j < X_{k+1} \cap X_{k+1} > X_{k+2}) \cdot \Delta y.$$

Für die Wahrscheinlichkeit eines „Run up“ der Länge zwei gilt somit:

$$P(X_k \leq X_{k+1} > X_{k+2}) \approx \sum_{j=1}^m P(y_j < X_{k+1} \cap X_{k+1} > X_{k+2}) \cdot \Delta y = \sum_{j=1}^m \int_{y_j}^1 x \cdot dx \cdot \Delta y$$

Man lässt auch hier aufgrund der gleichen Überlegungen wie im Fall eines „Run up“ der Länge eins wieder Δy gegen null gehen, womit auch $m \rightarrow \infty$. Es werden also die Wahrscheinlichkeiten für alle Bereiche von null bis eins aufsummiert, man erhält

$$P(X_k \leq X_{k+1} > X_{k+2}) = \int_0^1 \int_y^1 x \cdot dx \cdot dy = 1/3,$$

was gleich $2/(2+1)!$ ist. Entsprechend dazu gilt beispielsweise auch für einen „Run up“ der Länge drei

$$P(X_k \leq X_{k+1} \leq X_{k+2} > X_{k+3}) = \int_0^1 \int_y^1 \int_z^1 x \cdot dx \cdot dy \cdot dz = 1/8,$$

was wiederum gleich $3/(3+1)!$ ist.

Wie schon geschrieben, lässt sich dieses Vorgehen vermutlich auf alle anderen Runlängen übertragen, ich sehe keinen Grund, der dagegenspricht. Für die gezeigten Beispiele erhält man damit jedenfalls die gleichen Werte wie mit der von Knuth angegebenen Formel

$P = r/(r+1)!$ für die Wahrscheinlichkeit des Auftauchens von Runs der Länge r in einer Zufallszahlenfolge beliebiger Länge.

Ich vermute nun, dass sich der Erwartungswert der Länge eines Runs – die einem Run folgende Zufallszahl zähle ich dabei hier auch zu dieser Länge – für eine Zufallszahlenfolge der Länge n mit Hilfe von Knuths Formel wie folgt berechnen lässt:

$$E(\text{Länge eines Runs}) = \sum_{r=1}^n \frac{r}{(r+1)!} (r+1) = \sum_{r=1}^n \frac{1}{(r-1)!} = \sum_{x=0}^{n-1} \frac{1}{x!}$$

Für ein grosses n ist dieser Wert näherungsweise gleich der eulerschen Zahl e .²³

Der Erwartungswert für die Anzahl „Runs up“ oder „Runs down“ in einer Zufallszahlenfolge beträgt demnach näherungsweise n/e , da die Zufallszahlenfolge in genau so viele Runs der erwarteten Länge e – die folgende, nicht zum Run gehörende Zufallszahl darin immer mitgezählt – unterteilt werden kann. Der Erwartungswert für die Anzahl „Runs up“ oder „Runs down“ der Länge r in einer Zufallszahlenfolge der Länge n beträgt demnach

$$E(\text{Anzahl Runs der Länge } r) = \frac{n}{e} \cdot \frac{r}{(r+1)!}.$$

Im Test werden nun also die „Runs up“ und „Runs down“ gezählt, dann wird bei einem anschliessenden χ^2 -Test die beobachtete Anzahl der Runs mit bestimmten Runlängen mit der für jede Runlänge berechneten erwarteten Anzahl der Runs mit bestimmten Runlängen verglichen.

Da hier die Kategorien aus Runs gleicher Länge gebildet werden und nach den Erwartungswerten die meisten Werte in den Kategorien der kürzesten Runlängen zu finden sein werden – man erhält einen Erwartungswert von $n/e \cdot 1/2 \approx 0.184 \cdot n$ Runs der Länge eins, $n/e \cdot 1/3 \approx 0.123 \cdot n$ der Länge zwei und beispielsweise nur noch $n/e \cdot 1/144 \approx 0.00255 \cdot n$ Runs der Länge fünf – müssen dabei die letzten Kategorien so in einer Kategorie zusammengefasst werden, dass die Faustregel, n so zu wählen, dass in jeder Kategorie erwartungsgemäss mindestens fünf Werte sein sollten, erfüllt ist. Da die Erwartungswerte für immer längere Runs schnell sehr klein werden, genügt es, vom gegenwärtigen n ausgehend zu berechnen, ab welcher Runlänge die Erwartungswerte kleiner als fünf Runs sind und ab dann die Kategorien mit der letzten Kategorie, in der man noch mehr als fünf Runs erwarten kann, zusammenzufassen.

Eigentlich müsste man in die so entstehende letzte Kategorie alle Erwartungswerte bis zu dem für einen Run der Länge n miteinbeziehen, doch dies macht keinen Sinn, da diese Erwartungswerte sehr klein sind, praktischerweise wird man die Berechnung also bei einer bestimmten Runlänge abbrechen.

Wie gerade schon geschrieben, werden mit grosser Wahrscheinlichkeit die meisten Werte in den Kategorien der kürzeren Runlängen zu finden sein. Deshalb werden Abweichungen von der Erwartung in diesen grösseren Kategorien auch eher ausgeglichen werden als diejenigen in den kleineren Kategorien der längeren Runlängen.

Ich werde aufgrund dieser Überlegungen den Test der Uniform-Funktion für verschiedene Werte von n , nämlich für $n = 10^2, 10^3, 10^4, 10^5, 10^6, 10^7, 10^8$ durchführen. Dabei beginne ich bei jedem Test wieder mit den Startwerten von `seed1 = 12345` und `seed2 = 67890`.

3.3.3. Nullhypothese

Ich werde bei jedem der Tests einen Wert für χ^2 erhalten, der bei ν Freiheitsgraden eine bestimmte Wahrscheinlichkeit besitzt, bei zufällig verteilten Werten aufzutreten, die Anzahl der Freiheitsgrade ν ist dabei wie oben gleich der Anzahl der Klassen minus eins. Ich werde hier einige Werte für χ^2 in Abhängigkeit des Freiheitsgrades und der Wahrscheinlichkeit, dass ein Wert kleiner oder gleich dem jeweiligen χ^2 -Wert ist, aufführen.

$P(\chi^2_\nu \leq x)$		0.01	0.05	0.25	0.5	0.75	0.95	0.99
x	$\nu = 1$	0.00016	0.00393	0.1015	0.4549	1.323	3.841	6.635
	$\nu = 2$	0.02010	0.1026	0.5754	1.386	2.773	5.991	9.210
	$\nu = 3$	0.1148	0.3518	1.213	2.366	4.108	7.815	11.34
	$\nu = 4$	0.2971	0.7107	1.923	3.357	5.385	9.488	13.28
	$\nu = 5$	0.5543	1.1455	2.675	4.351	6.626	11.07	15.09
	$\nu = 6$	0.8721	1.635	3.455	5.348	7.841	12.59	16.81
	$\nu = 7$	1.239	2.167	4.255	6.346	9.037	14.07	18.48
	$\nu = 8$	1.646	2.733	5.071	7.344	10.22	15.51	20.09
	$\nu = 9$	2.088	3.325	5.899	8.343	11.39	16.92	21.67
	$\nu = 10$	2.558	3.940	6.737	9.342	12.55	18.31	23.21

Tabelle 2: Ausgewählte Punkte verschiedener χ^2 -Verteilungen mit jeweils ν Freiheitsgraden²⁴

Die Nullhypothese H_0 lautet: Die Verteilung der Prüfvariablen χ^2 entspricht der Verteilung, die durch die Dichtefunktion

$$f(x) = \frac{x^{(l-1)} e^{-\frac{x}{2}}}{2^l \Gamma(l)},$$

beschrieben wird, wobei $x \geq 0$, $l = \nu/2$ und ν die Anzahl der Freiheitsgrade ist.

Jeder Test wird einzeln behandelt, der Stichprobenumfang ist somit jeweils nur $n = 1$. Als Irrtumswahrscheinlichkeit wähle ich hier $\alpha = 0.02$. Da $P(\chi^2 \leq \chi^2_{\nu, p \leq 0.01}) \leq 0.01$ und $P(\chi^2 \geq \chi^2_{\nu, p \geq 0.99}) \leq 0.01$ ist, gilt für den Ablehnungsbereich $K = [0; \chi^2_{p \leq 0.01, \nu} [\cup] \chi^2_{p \geq 0.99, \nu}; \infty)$ (siehe Tabelle 2).

3.3.4. Programm

```
program runtest;
```

```
const anzzahlen = 50; anzzgesamt = 100000 {Insgesamt wird eine anzzgesamt lange Pseudo-
Zufallszahlenfolge getestet, weil aber bei einem "array[1..x] of real" x < 6554 sein
muss, geschieht dies in einer Schlaufe, in der jeweils anzzahlen lange Zahlenfolgen
getestet werden. anzzahlen sollte deshalb Teiler von anzzgesamt sein. Ist ein Run länger
als anzzahlen, erscheint eine Warnmeldung.}; maxrunle = 30 {Eigentlich müsste sogar der
Erwartungswert für einen Run der Länge anzzgesamt bei der Berechnung des Chiquadrat-
Wertes miteinbezogen werden, doch wird im Programm die Berechnung willkürlich bei
maxrunle abgebrochen, da die Erwartungswerte für lange Runs verschwindend klein sind. Es
muss maxrunle < anzzahlen sein.}; seedlbegin = 12345; seed2begin = 67890;
```

```
var endrun {endrun ist erfüllt, wenn am Ende des vorherigen Schlaufendurchlaufs gerade
ein Run beendet wurde, die erste Zahl des aktuellen Durchlaufs wird in diesem Fall nicht
in die Berechnungen miteinbezogen.}: boolean;
```

```

anzruns,il,maxrunl {längste beobachtete Runlänge},runl {gegenwärtige
Runlänge},seedlend,seed2end,seed1temp,seed2temp : integer;
chiquadrat : real;
textfensterbez : string;
anzbrunsdl {Anzahl beobachtete Runs der Länge i}: array[1..anzzahlen] of integer;
zufallszahl : array[1..anzzahlen+1] of real;

{
}

procedure zahlenaufnahme;
var   il,k,s1,s2,z : integer;

begin
  s1 := seed1temp;
  s2 := seed2temp;

  {Um die einzelnen Schlaufendurchläufe miteinander zu verknüpfen, muss die
  (anzzahlen+1).Pseudo-Zufallszahl jeweils schon bekannt sein, seedlend und seed2end aber
  müssen abgelesen werden, wenn il = anzzahlen ist.}
  for il := 1 to anzzahlen+1 do
    begin
      k := s1 div 53668;
      s1 := 40014 * (s1 - k * 53668) - k * 12211;
      if s1 < 0 then s1 := s1 + 2147483563;

      k := s2 div 52774;
      s2 := 40692 * (s2 - k * 52774) - k * 3791;
      if s2 < 0 then s2 := s2 + 2147483399;

      z := s1 - s2;
      if z < 1 then z := z + 2147483562;
      zufallszahl[il] := z * 4.656613059555*10(-10);

      if il = anzzahlen then
        begin
          seedlend := s1;
          seed2end := s2;
        end;
    end;
end;

{
}

procedure runtest(ausrichtung:string;ende:boolean); {Je nach ausrichtung ('up' oder
'down') wird ein "Run up"- oder ein "Run down"-Test für die anzzahlen lange Pseudo-
Zufallszahlenfolge durchgeführt; Ausgabe: beobachtete Anzahl Runs, Anzahl beobachtete
Runs der Länge i, längste beobachtete Runlänge.}
var   il,i2 : integer;
      differenz : real;

begin
  {Falls die Bedingung ende erfüllt ist, Bildung einer zusätzlichen Zahl, so dass der
  allerletzte Run gezählt wird, auch wenn er nicht abgeschlossen ist.}
  if ende = true then
    if ausrichtung = 'up' then
      zufallszahl[anzzahlen+1] := -1
    else
      zufallszahl[anzzahlen+1] := 2;

  {Ermittlung der Anzahl beobachteter Runs der Länge i und der längsten beobachteten
  Runlänge (Falls am Ende des letzten Schlaufendurchlaufs gerade ein Run beendet wurde -
  Bedingung endrun ist erfüllt - wird die erste Pseudo-Zufallszahl des aktuellen
  Durchlaufs nicht in die Berechnungen miteinbezogen.)}
  if endrun = true then
    il := 3
  else
    il := 2;
  while il < anzzahlen + 2 do
    begin
      if ausrichtung = 'up' then
        differenz := zufallszahl[il-1]-zufallszahl[il]
      else
        differenz := zufallszahl[il]-zufallszahl[il-1];
    end;
  end;
end;

```

```

    if differenz > 0 then
    begin
        anzruns := anzruns + 1;
        if runl > maxrunl then
            maxrunl := runl;
        for i2 := 1 to anzzahlen do
            if runl = i2 then
                anzbrunsdl[i2] := anzbrunsdl[i2] + 1;

                {Warnmeldung}
                if runl > anzzahlen then
                begin
                    writeln('Runlänge (' ,runl,') > anzzahlen. anzzahlen muss neu gewählt
werden. ');
                    readln;
                end;

                runl := 1;

                {Durch die Erhöhung von i1 wird die dem Run folgende Pseudo-Zufallszahl nicht
beachtet. Handelt es sich bei dieser folgenden Pseudo-Zufallszahl um die erste Zahl des
nächsten Schlaufendurchlaufs, ist die Bedingung endrun erfüllt.}
                il := il + 1;
                if il = anzzahlen + 2 then
                    endrun := true
                else
                    endrun := false;
                end
            else
                runl := runl + 1;

                il := il + 1;
            end;
        end;
    }

function fakultaet(x:integer):real;
var   il : integer;
      f : real;

begin
    f := 1;
    for il := 1 to x do
        f := f*il;
    fakultaet := f;
end;

}

function runsdle(x:integer):real; {Funktion des Erwartungswertes der Anzahl Runs der
Länge x in einer anzzgesamt langen Zufallszahlenfolge}
begin
    runsdle := anzzgesamt/(exp(1))*(x/fakultaet(x+1))
end;

}

procedure cqb; {Berechnung des Chiquadrat-Wertes, dabei werden die Klassen, die auf die
Klasse g-1 folgen, wie eine einzige Klasse behandelt.}
var   g,il,zanzbrunsdl {zusammengefasste Anzahl beobachtete Runs der Längen g bis
maxrunle}: integer;
      zrunsdle {zusammengefasste Erwartungswerte der Anzahl Runs der Längen g bis
maxrunle}: real;

begin
    chiquadrat := 0;
    zanzbrunsdl := 0;
    zrunsdle := 0;

    {Bestimmung von g}
    il := 1;
    repeat
        il := il + 1;

```

```

until runsdle(il) < 5;
g := il-1;
writeln('Chiquadrat-Test mit ',g,' Klassen:');

{Berechnung des Chiquadrat-Wertes}
for il := 1 to g-1 do
  chiquadrat := chiquadrat + (anzbrunsdl[il]-runsdle(il))^2/(runsdle(il));
for il := g to maxrunle do
  zanzbrunsdl := zanzbrunsdl + anzbrunsdl[il];
for il := g to maxrunle do
  zrunsdle := zrunsdle + runsdle(il);
chiquadrat := chiquadrat + (zanzbrunsdl-zrunsdle)^2/zrunsdle;
end;

{_____Hauptprogramm_____}

begin
  {Textfenster}
  textfensterbez := 'Run-Test';
  showText(textfensterbez);
  clearText;
  setTextRect(0,0,510,650);
  textFont(1);
  textSize(12);
  writeln('Es werden für eine ', anzzgesamt, ' Zahlen lange Pseudo-Zufallszahlenfolge
ein "Run up"- und ein "Run down"-Test durchgeführt.');
```

```

  {Durchführung des "Run up"-Tests}
  for il := 1 to anzzahlen do
    anzbrunsdl[il] := 0;
  anzruns := 0;
  endrun := false;
  maxrunl := 0;
  runl := 1;
  seed1temp := seed1begin;
  seed2temp := seed2begin;
  for il := 1 to anzzgesamt div anzzahlen do
    begin
      zahlenaufnahme;
      if il = anzzgesamt div anzzahlen then
        runtest('up',true)
      else
        runtest('up',false);
      seed1temp := seedlend;
      seed2temp := seed2end;
    end;

  writeln(' ');
  writeln('Ergebnisse des "Run up"-Tests der ',anzgesamt,' Pseudo-Zufallszahlen:');
  writeln('Anzahl Runs: ',anzruns:10);
  for il := 1 to maxrunl do
    writeln('Anzahl Runs der Länge ',il,': ',anzbrunsdl[il]:10);

  {Berechnung des Chiquadrat-Wertes}
  cqpb;

  writeln('Chiquadrat der Verteilung der Runs: ',chiquadrat:4:4);

  {Durchführung des "Run down"-Tests}
  for il := 1 to anzzahlen do
    anzbrunsdl[il] := 0;
  anzruns := 0;
  endrun := false;
  maxrunl := 0;
  runl := 1;
  seed1temp := seed1begin;
  seed2temp := seed2begin;
  for il := 1 to anzzgesamt div anzzahlen do
    begin
      zahlenaufnahme;
      if il = anzzgesamt div anzzahlen then
        runtest('down',true)
      else
        runtest('down',false);

```

```

        seed1temp := seed1end;
        seed2temp := seed2end;
    end;

    writeln(' ');
    writeln('Ergebnisse des "Run down"-Tests der ',anzzgesamt,' Pseudo-Zufallszahlen:');
    writeln('Anzahl Runs: ',anzruns:10);
    for il := 1 to maxrunl do
        writeln('Anzahl Runs der Länge ',il,': ',anzbrunsdl[il]:10);

        {Berechnung des Chiquadrat-Wertes}
        cqb;

    writeln('Chiquadrat der Verteilung der Runs: ',chiquadrat:4:4);
end.

```

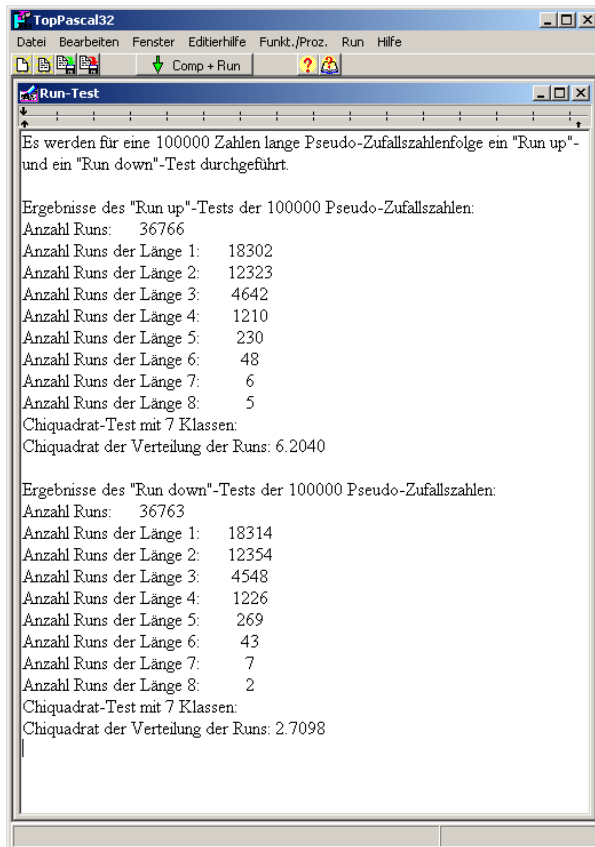


Abbildung 3: Ausgabe des beschriebenen Programms runtest

3.4 Poker-Test

3.4.1. Verfahren

Der Poker-Test betrachtet die Häufigkeit des Auftretens bestimmter Kombinationen in der zu testenden Zahlenfolge, er ist deshalb ein Unabhängigkeitstest.²⁵

Ich werde hier nicht das Verfahren des „klassischen Poker-Tests“ beschreiben und durchführen, sondern ein etwas vereinfachtes Verfahren, das leichter programmierbar ist, die Ergebnisse bleiben dabei Knuth zufolge immer noch aussagekräftig genug.

Die zu testenden n Zahlen werden, ohne ihre Reihenfolge zu verändern, entsprechend ihres Betrages einem von d Kartentypen zugeteilt. Ich werde die Zahl X_i – auch hier eine rationale Zahl zwischen null und eins – dem Kartentyp $m = 1, 2, \dots, d$ zuteilen, falls

$$\frac{m-1}{d} < X_i \leq \frac{m}{d}.$$

Bei $d = 10$ Kartentypen würde beispielsweise die Folge 0.043, 0.201, 0.494, 0.873, 0.523 zur Folge 1, 3, 5, 9, 6. Es entsteht also auf diese Weise eine Folge von Kartentypen Y_1, Y_2, \dots, Y_n , bei der jedes Glied nur einen von d verschiedenen Werten annehmen kann.

Die Folge der Werte Y_1, Y_2, \dots, Y_n wird nun in $n/5$ 5er Gruppen unterteilt, für alle j von null bis $n/5 - 1$ wird $\{Y_{5j+1}, Y_{5j+2}, \dots, Y_{5j+5}\}$ gebildet. Dann wird jedes Tupel einem der folgenden Ereignisse zugeteilt:

A_1 : „Alle im Tupel vorkommenden Kartentypen sind gleich.“

A_2 : „Es kommen zwei verschiedene Kartentypen vor.“

A_3 : „Es kommen drei verschiedene Kartentypen vor.“

A_4 : „Es kommen vier verschiedene Kartentypen vor.“

A_5 : „Alle Kartentypen sind verschieden.“

Auch hier lässt sich berechnen, welche Kombinationen in einer Zufallszahlenfolge wie häufig auftreten, womit man wiederum etwas über die Hypothese, es könne sich bei der Zahlenfolge um eine Folge von Zufallszahlen handeln, aussagen kann.

3.4.2. Umsetzung

Ich habe vor, mit der beobachteten Anzahl der Tupel der jeweiligen Kombination dem Vorschlag Knuths folgend einen χ^2 -Test durchzuführen, wobei es natürlich fünf Klassen geben wird. Dazu muss ich wissen, mit welcher Wahrscheinlichkeit in einem Tupel welche Kombination vorkommt, mit welcher Wahrscheinlichkeit wie viele verschiedene Kartentypen vorkommen.

Knuth behauptet Folgendes: Betrachtet man bei einem Test Tupel, die aus jeweils k Werten bestehen – beim von mir beschriebenen und durchgeführten Test ist $k = 5$ – von denen jeder einem von d Kartentypen zugeteilt wurde, taucht bei einem beliebigen Tupel das Ereignis „Es kommen r verschiedene Kartentypen vor.“ mit der Wahrscheinlichkeit

$$P(A_r) = \frac{1}{d^k} \frac{d!}{(d-r)!} S(k, r)$$

auf,²⁶ wobei $S(k, r)$ eine Stirling-Zahl zweiter Art ist, die man mit Hilfe der Rekursion $S(n+1, m) = S(n, m-1) + m \cdot S(n, m)$ aus $S(0, 0) = 1$ und $S(n, 0) = S(0, m) = 0$ für alle $n, m \in \mathbb{N}$ berechnen kann.²⁷

Diese Wahrscheinlichkeit erklärt sich folgendermassen: Insgesamt kann man aus d verschiedenen Elementen d^k verschiedene k lange Tupel bilden, es gibt also d^k verschiedene gleichwahrscheinliche Möglichkeiten, ein Tupel zu bilden, daher die Division durch d^k .

Nun muss noch angegeben werden, bei wie vielen der d^k möglichen Tupel sich das Ereignis „Es kommen r verschiedene Kartentypen vor.“ beobachten lässt, dies geschieht durch das Produkt von $d!/(d-r)!$ und $S(k, r)$:

$S(k, r)$ gibt „die Anzahl der unterschiedlichen Möglichkeiten, eine Menge“ mit k Elementen „in r nichtleere Teilmengen zu zerlegen“ an.²⁸ Kommen r verschiedene Kartentypen in einem Tupel vor, das aus k Werten – oder Karten – besteht, gibt es demnach $S(k, r)$ Möglichkeiten, die r verschiedenen Kartentypen den k Karten zuzuordnen, gewissermassen entscheidet hier der Kartentyp einer Karte, in welche der r Teilmengen sie eingeteilt wird.

Die r Kartentypen, die im Tupel vorkommen, müssen aus den d möglichen Kartentypen ausgewählt werden. Da man beachten muss, in welcher Reihenfolge sie gezogen werden, in welcher Reihenfolge sie also im Tupel vorkommen, gibt es dazu $d!/(d-r)!$ Möglichkeiten.

Somit gibt das Produkt von $d!/(d-r)!$ und $S(k, r)$ an, wie viele verschiedene Wege es gibt, das Ereignis „Es kommen r verschiedene Kartentypen vor.“ anzutreffen.

Ich lege für den Test der von der `Uniform`-Funktion generierten Pseudo-Zufallszahlenfolgen die Anzahl der verschiedenen Kartentypen willkürlich $k = 10$ fest. Da ich mit den erhaltenen Werten einen χ^2 -Test durchführen will, muss ich der Faustregel, n so zu wählen, dass in jeder Kategorie erwartungsgemäss mindestens fünf Werte sein sollten, folgen. Deshalb werde ich, da ich $k = 10$ gewählt habe, jeweils eine Folge der Länge $n = 500000$ betrachten, denn in diesem Fall tritt das Ereignis „Alle im Tupel vorkommenden Kartentypen sind gleich.“ mit

$$P(A_1) = \frac{1}{10^5} \frac{10!}{(10-1)!} S(5, 1) = \frac{1}{10^4}$$

auf, somit sollten erwartungsgemäss 10 der 100000 5er Tupel in die erste und kleinste Kategorie fallen. (Die übrigen Wahrscheinlichkeiten sind $P(A_2) = 0.0135$, $P(A_3) = 0.18$, $P(A_4) = 0.504$ und $P(A_5) = 0.3024$.)

Auch dieses Testverfahren werde ich, beginnend mit `seed1 = 12345` und `seed2 = 67890`, zwanzigmal durchführen, es werden auf diese Weise zehn Millionen aufeinanderfolgende Pseudo-Zufallszahlen untersucht.

3.4.3. Nullhypothese

Ich werde letztendlich einen Wert für χ^2 erhalten, der bei ν Freiheitsgraden eine bestimmte Wahrscheinlichkeit besitzt, bei zufällig verteilten Werten aufzutreten, die Anzahl der Freiheitsgrade ist dabei $\nu = 4$.

Die Nullhypothese H_0 lautet: Die Verteilung der Prüfvariablen χ^2 entspricht der Verteilung, die durch die Dichtefunktion

$$f(x) = \frac{x^{(l-1)} e^{-\frac{x}{2}}}{2^l \Gamma(l)}$$

beschrieben wird, wobei $x \geq 0$, $l = \nu/2$ und bei fünf Klassen $\nu = 4$ die Anzahl der Freiheitsgrade ist.

H_0 wird anhand der Anzahl der χ^2 -Werte geprüft, die ausserhalb des Bereiches $[0.297;13.28]$ liegen, in dem sie, falls H_0 zutreffen würde, mit einer Wahrscheinlichkeit von 0.98 zu finden wären (siehe Tabelle 2). Da ich als Irrtumswahrscheinlichkeit $\alpha = 6\%$ wähle, darf maximal eine der zwanzig Stichproben ausserhalb des Bereiches $[0.297;13.28]$ liegen, denn es ist

$$\sum_{k=2}^{20} \binom{20}{k} \cdot (0.02)^k \cdot (1 - 0.02)^{20-k} = 0.0599 \leq \alpha.$$

Der Ablehnungsbereich ist somit $K = [2;20]$.

3.4.4. Programm

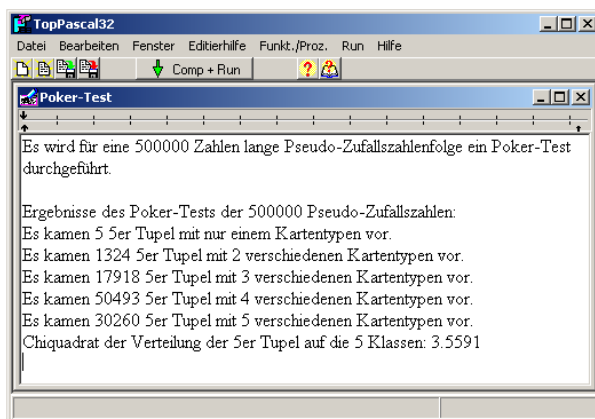


Abbildung 4: Ausgabe des beschriebenen Programms pokertest

```
program pokertest;
```

```
const anzkarten = 5 {Ein Tupel wird jeweils aus anzkarten Karten gebildet.}; anzktypen =
10 {Anzahl verschiedene Kartentypen, es muss anzktypen > anzkarten sein.}; anzzahlen =
100; anzzgesamt = 500000 {Insgesamt wird eine anzzgesamt lange Pseudo-Zufallszahlenfolge
getestet, doch auch hier muss dies in einer Schlaufe geschehen, in der jeweils anzzahlen
lange Zahlenfolgen getestet werden. anzzahlen muss deshalb Teiler von anzzgesamt sein.
Zusätzlich muss anzzahlen durch anzkarten teilbar sein.}; seedlbegin = 12345; seed2begin
= 67890;
```

```
var il, seedlend, seed2end, seedltemp, seed2temp : integer;
    chiquadrat : real;
    textfensterbez : string;
    anzbtupel {Anzahl beobachtete Tupel der Art i, wobei i angibt, wieviele
verschiedene Kartentypen im Tupel vorkamen.}: array[1..anzkarten] of integer;
    zufallszahl : array[1..anzzahlen] of real;
```

```
{ _____ }
```

```
procedure zahlenaufnahme;
var il, z, k, s1, s2 : integer;
```

```
begin
```

```

s1 := seed1temp;
s2 := seed2temp;
for i1 := 1 to anzzahlen do
  begin
    k := s1 div 53668;
    s1 := 40014 * (s1 - k * 53668) - k * 12211;
    if s1 < 0 then s1 := s1 + 2147483563;

    k := s2 div 52774;
    s2 := 40692 * (s2 - k * 52774) - k * 3791;
    if s2 < 0 then s2 := s2 + 2147483399;

    z := s1 - s2;
    if z < 1 then z := z + 2147483562;
    zufallszahl[i1] := z * 4.656613059555*10(-10);
  end;
seedlend := s1;
seed2end := s2;
end;
}

procedure pokertest; {Ein Poker-Test wird für anzzahlen Pseudo-Zufallszahlen
durchgeführt; Ausgabe: beobachtete Anzahl Tupel der Art i, i gibt an, wieviele
verschiedene Kartentypen im Tupel vorkamen.}
var  anzbktypen {Anzahl beobachtete Kartentypen},i1,i2,i3 : integer;
     anzbktyp {Anzahl beobachtete Karten des Kartentyps i}: array[1..anzktypen] of
integer;

begin
  i1 := 1;
  while i1 < anzzahlen do
    begin
      for i2 := 1 to anzktypen do
        anzbktyp[i2] := 0;
        anzbktypen := 0;

        {Zuteilung von jeweils anzkarten Pseudo-Zufallszahlen zu den anzkartentyp
Kartentypen}
        for i2 := 0 to anzkarten-1 do
          begin
            for i3 := 1 to anzktypen do
              if (zufallszahl[i2+i1] > (i3-1)/anzktypen) and (zufallszahl[i2+i1] <=
i3/anzktypen) then
                anzbktyp[i3] := anzbktyp[i3] + 1;
            end;

            {Zuteilung des Tupels zur Art i}
            for i2 := 1 to anzktypen do
              if anzbktyp[i2] > 0 then
                anzbktypen := anzbktypen + 1;
            for i2 := 1 to anzkarten do
              if anzbktypen = i2 then
                anzbtupel[i2] := anzbtupel[i2]+1;

            i1 := i1 + anzkarten;
          end;
        end;
      end;
    }

function stirlingz2(r,j : integer):real;
var  s : array[0..anzkarten,0..anzktypen] of real;
     i1,i2 : integer;

begin
  s[0,0] := 1;
  for i1 := 1 to r do
    s[i1,0] := 0;
  for i1 := 1 to j do
    s[0,i1] := 0;
  for i2 := 1 to j do
    for i1 := 1 to r do
      s[i1,i2] := s[i1-1,i2-1]+i2*s[i1-1,i2];

```

```

    stirlingz2 := s[r,j];
end;

{_____}

function anztupele(i:integer):real; {Funktion des Erwartungswertes der Anzahl Tupel der
Art i bei anzzgesamt/anzkarten Tupeln}
const k = anzkarten; d = anzktypen;
var   il: integer;
      dpr {dpr ist am Ende dpr = d*(d-1)*...*(d-i+1)}: real;

begin
    dpr := d;
    for il := 2 to i do
        dpr := dpr*(d-il+1);

    anztupele := (dpr/(d^k)*stirlingz2(k,i))*anzzgesamt/anzkarten;
end;

{_____Hauptprogramm_____}

begin
    {Textfenster}
    textfensterbez := 'Poker-Test';
    showText(textfensterbez);
    clearText;
    setTextRect(0,0,510,270);
    textFont(1);
    textSize(12);
    writeln('Es wird für eine ', anzzgesamt, ' Zahlen lange Pseudo-Zufallszahlenfolge ein
Poker-Test durchgeführt.');
```

```

    {Durchführung des Poker-Tests}
    for il := 1 to anzkarten do
        anzbtupel[il] := 0;
        seed1temp := seed1begin;
        seed2temp := seed2begin;
        for io := 1 to anzzgesamt div anzzahlen do
            begin
                zahlenaufnahme;
                pokertest;
                seed1temp := seed1end;
                seed2temp := seed2end;
            end;

    {Berechnung des Chiquadrat-Wertes}
    chiquadrat := 0;
    for il := 1 to anzkarten do
        chiquadrat := chiquadrat + (anzbtupel[il]-anztupele(il))^2/(anztupele(il));

    writeln(' ');
    writeln('Ergebnisse des Poker-Tests der ',anzzgesamt,' Pseudo-Zufallszahlen:');
    if anzbtupel[1] = 1 then
        writeln('Es kam ein ',anzkarten,'er Tupel mit nur einem Kartentypen vor.')
```

```

    else
        writeln('Es kamen ',anzbtupel[1]:1,' ',anzkarten,'er Tupel mit nur einem Kartentypen
vor.');
```

```

    for il := 2 to anzkarten do
        if anzbtupel[il] = 1 then
            writeln('Es kam ein ',anzkarten,'er Tupel mit ',il,' verschiedenen Kartentypen
vor.')
```

```

        else
            writeln('Es kamen ',anzbtupel[il]:1,' ',anzkarten,'er Tupel mit ',il,'
verschiedenen Kartentypen vor.');
```

```

        writeln('Chiquadrat der Verteilung der ',anzkarten,'er Tupel auf die ',anzkarten,'
Klassen: ', chiquadrat:4:4);
end.
```

4. Resultate der Tests der `uniform`-Funktion

4.1. Kolmogorow-Smirnow-Test

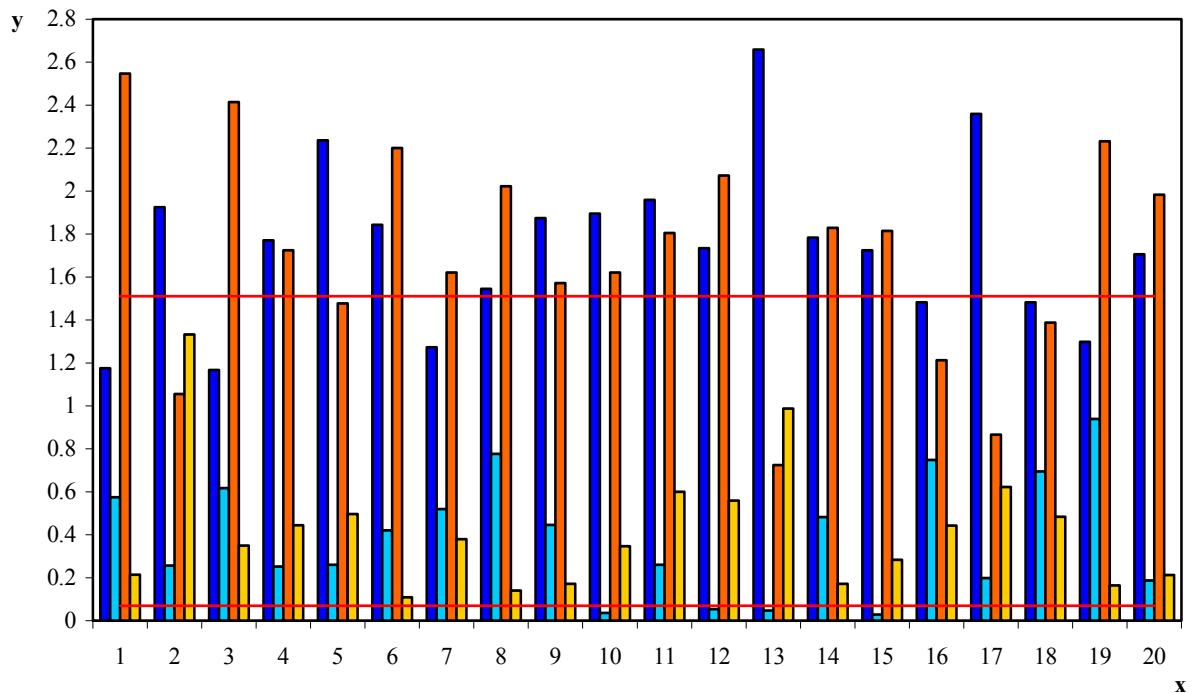


Abbildung 5: Die Resultate der Kolmogorow-Smirnow-Tests

Legende:

x-Achse: Test-Nummer

y-Achse: Betrag der K_{1000}^{\pm}

rot: die Grenzen des Bereiches $[0.07;1.51]$

blau bzw. hellblau: die K_{1000}^+ bzw. K_{1000}^- zweiter Ebene der Verteilung der K_{1000}^+ erster Ebene

orange bzw. gelb: die K_{1000}^+ bzw. K_{1000}^- zweiter Ebene der Verteilung der K_{1000}^- erster Ebene

	Test der K_{1000}^+ erster Ebene		Test der K_{1000}^- erster Ebene		seed1 und seed2 zu Beginn des Test	
	K_{1000}^+ zweiter Ebene	K_{1000}^- zweiter Ebene	K_{1000}^+ zweiter Ebene	K_{1000}^- zweiter Ebene	seed1	seed2
1	1.1743	0.5741	2.5463	0.2138	12345	67890
2	1.9237	0.2573	1.0545	1.3318	826277612	155873079
3	1.1667	0.6174	2.4150	0.3498	945483033	852888922
4	1.7694	0.2517	1.7241	0.4438	471594436	1441293558
5	2.2361	0.2591	1.4771	0.4962	1276049815	1814724314
6	1.8421	0.4197	2.2005	0.1082	1429921728	1991135797
7	1.2729	0.5197	1.6208	0.3790	146709890	2089824147
8	1.5444	0.7758	2.0217	0.1398	1546622763	1488560821
9	1.8737	0.4463	1.5720	0.1719	838783829	1576131487
10	1.8940	0.0365	1.6208	0.3465	1804473592	90356765
11	1.9584	0.2591	1.8044	0.5996	1466792609	818836409
12	1.7339	0.0538	2.0721	0.5595	1939218517	829316573
13	2.6582	0.0471	0.7239	0.9881	2086947860	185575333
14	1.7825	0.4811	1.8288	0.1719	1852847542	1036872026
15	1.7241	0.0291	1.8141	0.2839	1857779877	188292981
16	1.4811	0.7482	1.2126	0.4421	59495091	1895357525
17	2.3586	0.1977	0.8654	0.6227	717590490	1723384975
18	1.4811	0.6944	1.3880	0.4830	1884302608	813113349
19	1.2984	0.9379	2.2321	0.1642	384473123	1697832033
20	1.7061	0.1873	1.9831	0.2119	1230894902	1646154228

Tabelle 3: Die Resultate der Kolmogorow-Smirnow-Tests

4.2. χ^2 -Test

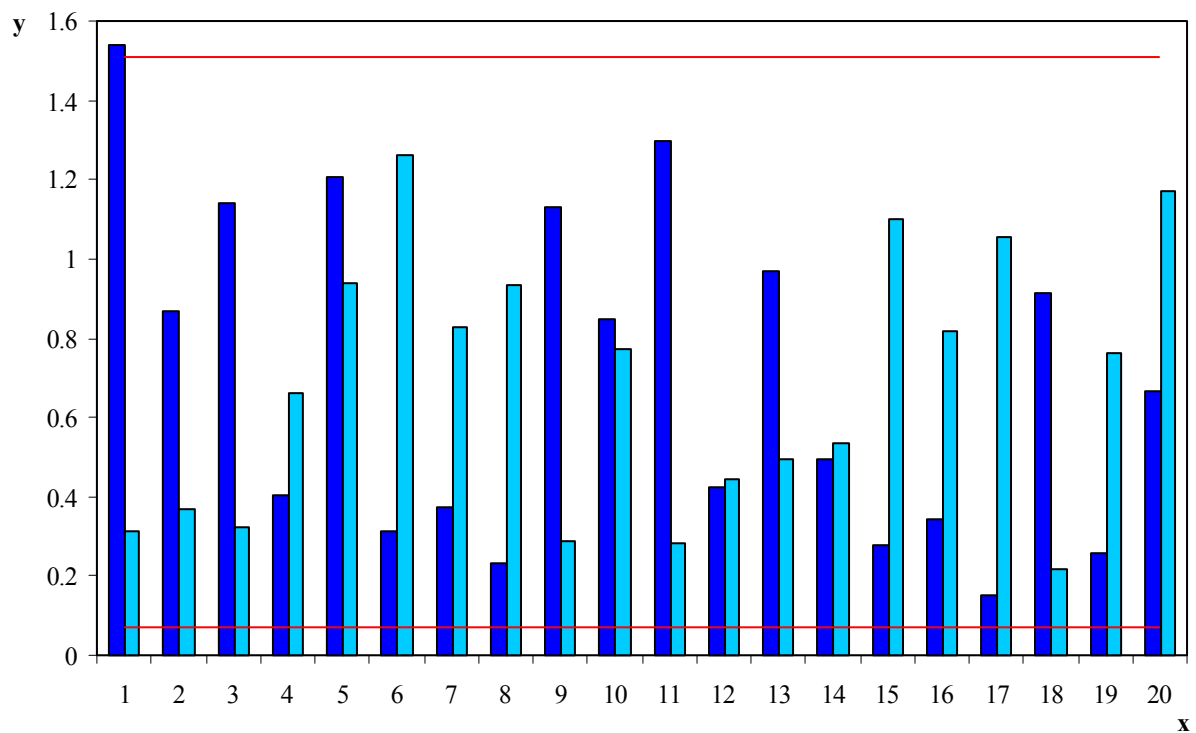


Abbildung 6: Die Resultate der χ^2 -Tests

Legende:

x-Achse: Test-Nummer

y-Achse: Betrag der K_{1000}^{\pm}

rot: die Grenzen des Bereiches [0.07;1.51]

blau bzw. hellblau: die K_{1000}^+ bzw. K_{1000}^- der Verteilung der χ^2 -Werte

	Verteilung der χ^2 -Werte		seed1 und seed2 zu Beginn des Test	
	K_{1000}^+	K_{1000}^-	seed1	seed2
1	0.3133	1.5379	12345	67890
2	0.3669	0.8676	826277612	155873079
3	0.3233	1.1409	945483033	852888922
4	0.6634	0.4049	471594436	1441293558
5	0.9372	1.2041	1276049815	1814724314
6	1.2608	0.3144	1429921728	1991135797
7	0.8281	0.3741	146709890	2089824147
8	0.9349	0.2317	1546622763	1488560821
9	0.2896	1.1315	838783829	1576131487
10	0.7704	0.8469	1804473592	90356765
11	0.2829	1.2989	1466792609	818836409
12	0.4428	0.4260	1939218517	829316573
13	0.4938	0.9697	2086947860	185575333
14	0.5375	0.4943	1852847542	1036872026
15	1.1005	0.2782	1857779877	188292981
16	0.8159	0.3444	59495091	1895357525
17	1.0570	0.1490	717590490	1723384975
18	0.2182	0.9132	1884302608	813113349
19	0.7600	0.2569	384473123	1697832033
20	1.1697	0.6659	1230894902	1646154228

Tabelle 4: Die Resultate der χ^2 -Tests

4.3. Run-Test

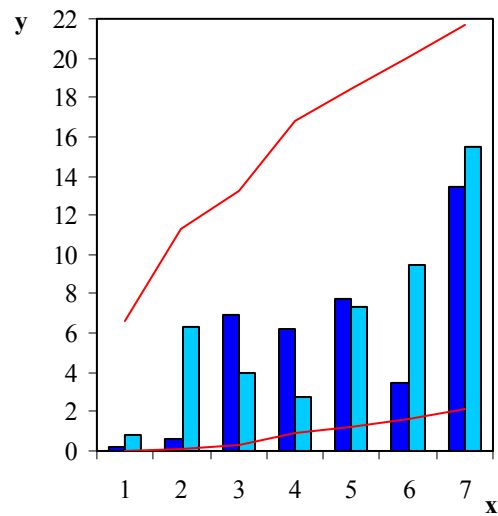


Abbildung 7: Die Resultate der „Run up“- und „Run down“-Tests

Legende:

x-Achse: Test-Nummer

y-Achse: Betrag der χ^2 -Werte

rot: die Grenzen des Ablehnungsbereiches

blau bzw. hellblau: die χ^2 -Werte der Verteilung der „Runs up“ bzw. „Runs down“

	Anzahl getestete Zahlen	χ^2 der „Runs up“-Verteilung	χ^2 der „Runs down“-Verteilung	Anzahl Klassen bei den χ^2 -Tests
1	10^2	0.2459	0.7665	2
2	10^3	0.6178	6.3635	4
3	10^4	6.9303	3.9405	5
4	10^5	6.2040	2.7098	7
5	10^6	7.7050	7.2917	8
6	10^7	3.4805	9.4245	9
7	10^8	13.4017	15.4417	10

Tabelle 5: Die Resultate der „Run up“- und „Run down“-Tests; der Test begann jedes Mal mit seed1 = 12345 und seed2 = 67890

4.4. Poker-Test

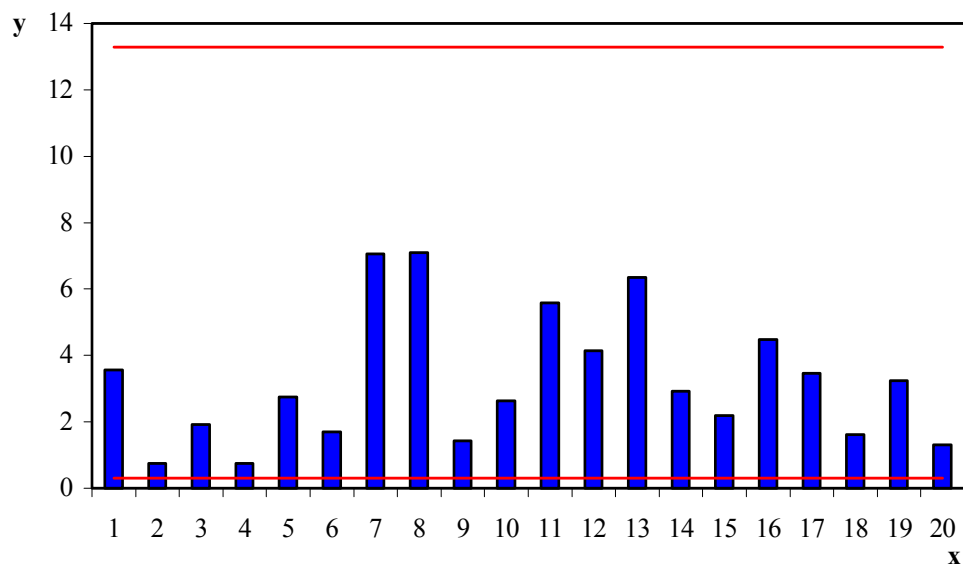


Abbildung 8: Die Resultate der Poker-Tests

Legende:

x-Achse: Test-Nummer

y-Achse: Betrag der χ^2 -Werte

rot: die Grenzen des Bereiches [0.297;13.28]

blau: die χ^2 -Werte der Verteilung der verschiedenen Tupel-Arten

	χ^2 der Verteilung der Tupel	seed1 und seed2 zu Beginn des Test	
		seed1	seed2
1	3.5591	12345	67890
2	0.7488	192293598	844120704
3	1.9233	826277612	155873079
4	0.7537	117008272	410889866
5	2.7421	945483033	852888922
6	1.7006	1734277810	1784083903
7	7.0605	471594436	1441293558
8	7.0912	104200650	198730628
9	1.4263	1276049815	1814724314
10	2.6275	551686257	167940402
11	5.5761	1429921728	1991135797
12	4.1480	1022333740	1185414552
13	6.3439	146709890	2089824147
14	2.9222	332186987	1606810798
15	2.1932	1546622763	1488560821
16	4.4865	952065911	1025018670
17	3.4553	838783829	1576131487
18	1.6150	1260035272	708674595
19	3.2438	1804473592	90356765
20	1.2989	1738374447	343183376

Tabelle 6: Die Resultate der Poker-Tests

5. Diskussion

Die unter 3.1.3. aufgestellte Nullhypothese H_0 muss aufgrund der Test-Resultate verworfen werden. Nur gerade die Verteilung der K_{1000}^- zweiter Ebene der Verteilung der K_{1000}^- erster Ebene widerspricht der Nullhypothese nicht, alle Werte liegen dort im Bereich $[0.07;1.51]$, die drei anderen Verteilungen aber widerlegen H_0 eindeutig: Bei der Verteilung der K_{1000}^\pm zweiter Ebene der Verteilung der K_{1000}^+ erster Ebene zählt man vierzehn K_{1000}^+ -Werte und vier K_{1000}^- -Werte, die nicht im Bereich $[0.07;1.51]$ liegen, bei der Verteilung der K_{1000}^+ zweiter Ebene der Verteilung der K_{1000}^- erster Ebene sind es ebenfalls vierzehn Werte, die nicht im Bereich $[0.07;1.51]$ liegen. Bei einer Irrtumswahrscheinlichkeit von $\alpha = 0.06$ dürfte es jeweils maximal einer sein.

Die unter 3.2.3. aufgestellte Nullhypothese H_0 kann aufgrund der Test-Resultate nicht verworfen werden, denn es gibt nur einen einzigen K_{1000}^- -Wert, der im Bereich $[0.07;1.51]$ liegt.

Auch die unter 3.3.3. aufgestellte Nullhypothese H_0 kann aufgrund der Test-Resultate nicht verworfen werden, kein einziger der χ^2 -Werte liegt im Ablehnungsbereich.

Die unter 3.4.3. aufgestellte Hypothese H_0 kann aufgrund der Test-Resultate ebenfalls nicht widerlegt werden, hier lag kein einziger der zwanzig χ^2 -Werte ausserhalb des Bereiches $[0.297;13.28]$.

Die `Uniform`-Funktion besteht also den χ^2 -Test, den „Run up“-Test, den „Run down“-Test sowie den Pokertest, versagt jedoch beim Kolmogorow-Smirnow-Test eindeutig. Es hat den Anschein, als ob alle Tests relativ hart zu bestehen sein sollten, da die jeweiligen Verfahren immer noch entweder mit einem Kolmogorow-Smirnow-Test oder einem χ^2 -Test kombiniert wurden, bei drei der Test wurden sogar noch die Resultate dieser Tests wiederum einem Test unterzogen. Trotzdem bestand die `Uniform`-Funktion drei der Tests und man könnte sagen, ein Test sei zumindest teilweise bestanden.

L'Ecuyer führte mit der `Uniform`-Funktion ebenfalls einige Tests durch, wobei er mehrere Milliarden Pseudo-Zufallszahlen testete. Ihm zufolge bestand die `Uniform`-Funktion verschiedene Versionen des χ^2 -Anpassungstests, des „Serial“- , „Gap“- , Poker-, „Coupon's collector“- , Permutations- und des „Collision“-Tests sowie je eine Version des „Maximum-of-t“-Tests und des „Runs up“-Tests. Alle Tests wiederholte er bis zu zehntausend Mal und wandte auf die Resultate am Ende jeweils einen Kolmogorow-Smirnow-Test an.²⁹ Die `Uniform`-Funktion scheint also Pseudo-Zufallszahlen vergleichsweise hoher Qualität zu generieren. Ich kann deshalb die Möglichkeit nicht ausschliessen, dass sich in der Umsetzung des Kolmogorow-Smirnow-Tests ein Fehler eingeschlichen hat. Ich konnte jedoch keinen finden. Soweit ich weiss, führte L'Ecuyer selbst leider keinen Kolmogorow-Smirnow-Test „mit zwei Ebenen“, wie ich ihn hier vorstellte, durch.

Ich folgere somit, dass die Hypothese, es handle sich bei den von der `Uniform`-Funktion generierten Zahlenfolgen um Zufallszahlenfolgen, aufgrund der Testresultate verworfen werden muss.

6. Anmerkungen

- ¹ zum TI-92 im Allgemeinen vgl. TI-Handbuch
- ² vgl. TI-Handbuch, S.432
- ³ vgl. L'Ecuyer, S. 747
- ⁴ vgl. TI-Handbuch, S.432
- ⁵ vgl. Knuth, S.41f.; vgl. Schulze, S.3
- ⁶ vgl. Knuth, S.48f.
- ⁷ zum gesamten beschriebenen Testverfahren vgl. Knuth, S. 48f.
- ⁸ Knuth, S.50
- ⁹ vgl. Knuth, S.50, S.58 und S.559
- ¹⁰ vgl. Knuth, S.51
- ¹¹ vgl. Knuth, S.51
- ¹² vgl. Knuth, S.51
- ¹³ zum gesamten beschriebenen Testverfahren vgl. Knuth, S. 42f.
- ¹⁴ vgl. Knuth, S.43
- ¹⁵ vgl. Knuth, S.44
- ¹⁶ vgl. Knuth, S.53
- ¹⁷ vgl. Duden, S. 83f.
- ¹⁸ vgl. Duden, S. 197f.
- ¹⁹ zum gesamten beschriebenen Testverfahren vgl. Knuth, S. 66f., S.77 und S.564
- ²⁰ Schulze, S.18
- ²¹ vgl. Knuth, S. 77
- ²² vgl. Knuth, S.77 und S.564
- ²³ vgl. Duden, S.147
- ²⁴ vgl. Knuth, S.44
- ²⁵ zum gesamten beschriebenen Testverfahren vgl. Knuth, S. 63f.
- ²⁶ vgl. Knuth, S.64
- ²⁷ vgl. Duden, S. 585
- ²⁸ Duden, S. 584f.
- ²⁹ vgl. L'Ecuyer, S. 748f.

7. Literaturverzeichnis

Knuth, Donald Erwin: The Art of Computer Programming, Volume 2, Seminumerical Algorithms, Third Edition. Munich, Addison-Wesley, 199

Meyers Lexikonredaktion: Duden, Rechnen und Mathematik, 6., überarbeitete Auflage. Mannheim, Dudenverlag, 2000

Texas Instruments: TI-92 Handbuch. © 1995,1995 Texas Instruments Incorporated

Pierre L'Ecuyer: Efficient and Portable Combined Random Number Generators. Communications of the ACM, Vol. 31, Number 6, Juni 1988, S.742-749, 774

Dr. Ing-Habil. Thomas Schulze: Simulation II,
http://www-wi.cs.uni-magdeburg.de/lehre/ss99/simu/skript/zz_test.pdf, 5.12.2002