Josef Böhm, Dr. Hubert Langlotz

Simulation von Zufallsversuchen auf dem TI-89/92

Teil 1 Grundlagen

Pädagogische Gesichtspunkte

Darstellung(en)	Untersuchung	WB ←→ BB	Dokumentation
	experimentell	WB → BB	Programme, Schülerheft

Technologie

Tabellen-kalkulation	Graphischer Taschenrechner	Computeralgebra-system	Dynamische Geometriesoft-ware
X	X	X	

Ziel und Beschreibung der Einheit

Ziel dieser Einheit ist es, den Schülerinnen und Schülern die Bedeutung von Simulationen in der Wahrscheinlichkeitsrechnung deutlich zu machen. Die Schülerinnen und Schüler sollen zunächst anhand einfacher Fragestellungen die Möglichkeiten des Daten - Matrix-Editors sowie den Aufbau kleiner Programme kennen lernen und in die Lage versetzt werden, verschiedene Probleme der Wahrscheinlichkeitsrechnung so zu modellieren, dass diese durch Simulationen experimentell bearbeitet werden können und damit auch Anregungen für die rechnerische Lösung des Problems gefunden werden können.

Rolle der Technologie

- Untersuchungswerkzeug
- Programmierwerkzeug
- Visualisierungswerkzeug

Notwendige Vorkenntnisse

Es wird davon ausgegangen, dass keine Vorkenntnisse vorhanden sind. Die Schüler sollten in der Lage sein, ein mathematisches Problem zu modellieren.

Dauer der Einheit

offen, abhängig von der Anzahl der genutzten Simulationen

Unterrichtsorganisation

Nachdem zunächst anhand einfacher Beispiele die Grundfunktionen zur Nutzung des Daten-Matrix-Editors durch den Lehrer bereitgestellt werden, sollte bei der Behandlung weiterer Problemstellungen von der Modellierung in jedem Fall nach Möglichkeiten einer Realsimulation gesucht werden, welche dann mit vertretbarer Anzahl von Versuchen auch tatsächlich durchgeführt werden sollten. Anschließend kann man dann zur Computersimulation übergehen, um noch mehr Datenmaterial zu erhalten, welches schließlich über Vermutungen zur Berechnung der theoretischen Wahrscheinlichkeiten führen sollte. Prinzipiell ist auch ein Weg von der Berechnung der Wahrscheinlichkeiten zur experimentellen Kontrolle durch eine Computersimulation möglich.

Ob der Schwerpunkt auf die Nutzung des Daten-Matrix-Editors oder auf das Erstellen kleiner Programme gelegt wird, sollte von der jeweiligen Zielsetzung und Klassensituation abhängig gemacht werden.

Aufgabenstellung

1. Bearbeiten Sie die folgenden Aufgaben und Beispiele, um sich mit den Grundfunktionen des Daten - Matrix - Editors und einfachen Programmierelementen vertraut zu machen!

Zufallsversuche lassen sich mit Hilfe von Zufallszahlen simulieren. Fast jeder Taschenrechner und fast jede höhere Programmiersprache besitzen einen integrierten (Pseudo-)-Zufallszahlengenerator.

Aufgabe 1:
Erzeugen Sie mit der Funktion `rand()` einzelne Zufallszahlen!
(mögliche Ergebnisse 0.94512 .... 0.0041)

Aufgabe 2:
Geben Sie nun `rand(1), rand(2), rand(6),..., rand(5.5), rand(0), rand(-10)` ein und beschreiben Sie die Wirkungsweise der Funktion `rand(a)`!

Aufgabe 3:
Erzeugen Sie zunächst eine Folge von 10 zufälligen Münzwürfen aus dem Gedächtnis nach Gutdünken, dann mit einer echten Münze und schließlich mit dem TI89/92 !
Nutzen Sie dazu die TI89/92-Funktion **seq**!

```
Bsp.: seq(n,n,1,10) {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

(Dieser Gedanke kann später wieder aufgegriffen werden, um die Problematik der "runs" zu diskutieren)
Mögliches Ergebnis des TI89/92: `{2 2 1 1 1 1 2 1 1 1}`
Hier bietet sich eine erste Diskussion zur "Güte" von Zufallszahlen an! Man kann auch ein erstes kleines Programm vorgeben und damit die Grundelemente eines TI89/92-Programms erläutern:

Beispiel 1:

Ein erstes kleines Programm: Zufallsregen zuza(n)

Wechseln Sie durch **[Apps]** in den Programm-Editor. Schreiben Sie in das Eingabefeld hinter Variable den neuen Programmnamen zuza und drücken Sie zweimal die Entertaste. Sie befinden sich im Programmfenster und können nun das Programm wie angegeben eingeben.
Hinweise: Vergessen Sie nicht, die Variable n im Programmkopf einzufügen. Eine Reihe von Befehlen lassen sich direkt über die Menüs **F2** und **F3** in den Programmtext einfügen.

`: zuza(n)`	Name des Programms und Übergabe der Versuchs-
`: Prgm`	anzahl n
`: Local i,x,y`	Im Programm benötigte Variablen werden lokalisiert
`: Clrdraw`	Löschen des Grafikbildschirms
`: For i,1,n`	Beginn der Schleife
`: rand()→x`	Erzeugung von Zufallszahlen zwischen 0 und 1
`: rand()→y`	
`: PtOn x,y`	Punkt mit den Koordinaten x, y wird gezeichnet
`: EndFor`	Schleifenende
`: EndPrgm`	

Bevor das Programm gestartet wird, stellen Sie bitte den Grafikmodus auf „Function“ und die Fenstereinstellungen auf
`xmin=0, xmax=1, xscl=1, ymin=0, ymax=1, xres=1`.

Interpretieren Sie das erhaltene Bild!

Beispiel 2: Nutzung des Daten-Matrix-Editors für das gleiche Problem

Öffnen Sie über `[APPS]` den Daten-Matrix-Editor und aktivieren Sie die Option `New` , und geben Sie der neuen Datavariable z.B. den Namen zz.
Nach dem zweimaligen Drücken der Eingabetaste befinden Sie sich im Daten-Editor.
Geben Sie im Feld c1 ein : `seq(rand(),t,1,200)`, dann im Feld c2: `seq(rand(2), t,1,200)` ein.
(Sollte keine Berechnung erfolgen, so müssen Sie den automatischen Berechnungsmodus einstellen: `F1 - 9 Auto-calculate` **on)**

Grafische Darstellung

Mittels **F2** gelangen Sie in das Plot Setup. Dort müssen Sie zunächst über F1 die Darstellungsart definieren. Für Plot Type wählen Sie `scatter`, für Mark die Option `Dot`, für x die Option `c1`, für y die Option `c2`. Bestätigen Sie alle Eingaben mit `Enter`.
Mit ♦ **F3** wird ein erster Versuch gestartet, eine grafische Darstellung zu erzielen. In der Regel erhält man ein unbefriedigendes Ergebnis(was von den Voreinstellungen im Grafikfenster abhängt!).
Stellen Sie für die Fenstereinstellungen `xmin=0, xmax=1, xscl=1, ymin=0, ymax=1, xres=1` ein.

Beispiel 3: Münzwürfe unter Nutzung des Daten-MatrixEditors

Es sollen n Würfe mit einer idealen Münze simuliert werden. Anschließend soll die relative Häufigkeit für Wappen grafisch dargestellt werden.

Öffnen Sie über `[APPS]` den Daten-Matrix-Editor , und geben Sie der neuen Datavariable z.B. den Namen `mwurf`.

Geben Sie im Feld c1 ein : `seq(n,n,1,100)`, dann im Feld c2: `seq(rand(2) -1, n, 1, 100)` . Weiterhin c3: `cumSum(c2)` und `c4=c3/c1`.

Interpretieren Sie die erhaltenen Ergebnisse in Spalte c3 und c4 !

Grafische Darstellung

Für Plot Type wählen Sie dieses mal `xyline`, für Mark die Option `Dot`, für x die Option `c1`, für y die Option `c4`.
Auch hier werden Sie in der Regel eine unbefriedigende Darstellung erhalten. Ändern Sie dies folgendermaßen ab:
Mit **F2 Zoom 9** aktivieren Sie `ZoomData`. Geben Sie weiterhin im y-Editor als Funktion `y1(x)=0.5` ein.

Interpretieren Sie nun die erhaltene Grafik!

Um einen weiteren Versuchsdurchlauf zu absolvieren, müssen Sie wieder in das aktuelle Daten-Matrix-Fenster wechseln**(`[APPS] 6 Current`)**. Sie erhalten eine neues Ergebnis, welches Sie wiederum zeichnen lassen können.

Beispiel 4: Würfeln - simuliert mit einem Programm

Schreiben Sie ein Programm, welches den n-fachen Wurf eines Würfels simuliert und die erzielten Ergebnisse im Ein-Ausgabebildschirm ausgibt!

```
:wurf1(n)
:Prgm
:Local i
:clrIO                    löscht den Ein- und Ausgabebildschirm
:newlist(n)→liste;        Erzeugung einer neuen Liste der Länge n
:For i,1,n
:rand(6)→liste[i]         das Ergebnis des i-ten Wurfes wird im
:EndFor                   Listenplatz i gespeichert
:Disp liste               Ausgabe der Liste im Ein- und Ausgabe-
:EndFor                   schirm
:EndPrgm
```

Wechseln Sie jetzt in den `HomeEditor` und geben Sie in der Eingabezeile `wurf1(10)` ein. Ergebnis:
Im Ein - und Ausgabefenster erscheint z.B. : `{2 5 4 6 4 5 5 2 6 4}`.

Wechseln Sie in zurück in den `HomeEditor` und geben Sie `liste` und anschließend `i` ein!
Versuchen Sie sich den Unterschied klar zu machen!
(Hinweis: Schauen Sie sich die dritte Programmzeile an!)

Aufgabe 4:

Ändern Sie das Programm so ab, dass man gleichzeitig k-mal den n-fachen Wurf durchführen kann!

Benennen Sie das Programm mit `wurf2(n,k)`! Nutzen Sie dazu die Kopierfunktion!

Beispiel 5: Erzeugung binomialverteilter Zufallsgrößen

Als Idee bietet sich an(vgl. B. Grabinger, Münster 1999):
Ziehen einer Zufallszahl x aus [0 ; 1], Wenn x< p, dann Erfolg, sonst Misserfolg.

In diesem Fall wird kein Programm, sondern eine Funktion geschrieben, da Funktionen auch als Bestandteil von Termen genutzt werden können.

```
bern(p)
Func
IF rand() < p Then     Bedingte Anweisung Wenn ... dann ... sonst
1
Else
0
EndIf
Endfunc
```

Aufgabe 5:

Erzeugen Sie eine binomialverteilte Zufallsgröße der Länge 10 mit der Trefferwahrscheinlichkeit von p= 0.5.

Mit einer weiteren Funktion bino(n,p) (Vgl. B. Grabinger Münster 1999) bzw. einfach unter Nutzung der Befehlsfolge sum(seq(bern(0.5),n,1,10)) lassen sich die Anzahl der Treffer bei binomialverteilten Zufallsvorgängen darstellen.

Definieren Sie sich die Funktion
`sum(seq(bern(p),n,1,10)) STO▸ bino(n,p)!`

Aufgabe 6:

Nutzen Sie die angegebenen Funktionen, um im Daten-MatrixEditor sowie grafisch Simulationen der Binomialverteilung zu erzeugen!

Beispiel 6: Geometrisch verteilte Zufallsgrößen

Ein betrunkener Seemann steht vor seiner Kajüte. In seiner Hosentasche hat er ein Schlüsselbund mit 5(s) gleichartigen Schlüsseln, von denen nur einer zu seiner Kajüte passt. Da er den richtigen Schlüssel nicht mehr identifizieren kann, zieht er immer blindlings einen aus dem Bund, ohne sich zu merken, welchen er schon versucht hat. Wie oft wird er wohl im Mittelprobieren müssen?

Das Beispielprogramm `sailor(s)` wird mit `sailor(5)` gestartet. Es gibt in Form einer Liste die vom Seemann gezogenen Schlüssel aus. Der passende Schlüssel ist der mit der Nummer 1.

```
sailor(s)
Prgm
Local a,i
ClrIO
{}→list                          Erzeugen einer leeren Liste
rand(s) →a
augment(list,{a})→list           Anhängen der Liste {a} an list
While a≠1                        Solange- Schleife
rand(s) →a
augment(list,{a})→list
EndWhile
Disp list
Disp "versuche"
Disp dim(list)                   Ausgabe der Anzahl der
EndPrgm                          Elemente der Liste list
```

Aufgabe7:

Schreiben Sie ein Programm, welches den Mittelwert der notwendigen Versuche ausgibt!

(Informieren Sie sich im Handbuch über weitere Listenfunktionen!)

Weitere Experimente finden sich in der Programmbibliothek in Teil 2.

Technische Hinweise

Die wichtigsten Funktionen zum Programmieren und für die Arbeit mit dem Daten-MatrixEditor seien hier genannt

Was willst Du erreichen	**Wie Du das machst**
Eine Zufallszahl erzeugen	`rand(), rand(7)`

Zwei Würfel werden 200 -mal geworfen und die Häufigkeit der auftretenden Summen soll dargestellt werden)
Nutzung des Daten - Matrix Editors)
	`c1: seq(rand(6),n,1,200)`
	`c2: seq(rand(6),n,1,200)`
	`c3: c1+c2`
	`F2 Plot Setup`
	`F1 Define`
	`Plot Type Histogramm`
	`x: c3`

Eine Operation in einem Programm n - mal wiederholen und das Ergebnis in einer Liste speichern	`For i,1,n`
	`rand(6) → a`
	`augment(list,{a})→list`
	`EndFor`

Fallunterscheidungen durchführen	`If a < b Then`
	`1`
	`Else`
	`0`
	`EndIf`

Sortieren einer Liste	`SortA list`

Teil 2: Simulationsprogramme

(Die vorliegende Programmsammlung ist vorerst auf dem TI-92 lauffähig, die Bildschirmausgaben müssen dem TI-89 angepasst werden).

Pädagogische Gesichtspunkte

Darstellung(en)	Untersuchung	WB ←→ BB	Dokumentation
graphisch numerisch tabellarisch	experimentell simuliert analytisch	BB → WB	Programme, Protokolle Graphiken

Technologie

Tabellen-kalkulation	Graphischer Taschenrechner	Computeralgebra-system	Dynamische Geometriesoft-ware
X	X	X	

Ziel und Beschreibung der Einheit

Die Schülerinnen und Schüler sollen mit der Möglichkeit vertraut werden, Zufallsexperimente mit dem Computer zu simulieren. Die Simulationsergebnisse können mit experimentell gemachten Erfahrungen verglichen werden, und dann als Anreiz dienen, die Wahrscheinlichkeiten auch analytisch zu berechnen. Graphische Darstellungen werden dazu angeboten. Einige bekannte Aufgaben aus der Literatur werden bearbeitet.

Rolle der Technologie

- Experimentierwerkzeug
- Programmierwerkzeug
- Visualisierungswerkzeug

Notwendige Vorkenntnisse

Es wird davon ausgegangen, dass die Grundkenntnisse aus Teil 1 vorhanden sind. Grundkenntnisse der Wahrscheinlichkeitsrechnung sind entweder bereits vorhanden oder werden parallel zu den Simulationen vermittelt.

Dauer der Einheit

offen, abhängig von der Anzahl der genutzten Simulationen

Unterrichtsorganisation

Gerade diese Untersuchungen eignen sich hervorragend zum Bearbeiten in Arbeitsgruppen. Es ist sicher nicht notwendig, dass alle Schülerinnen und Schüler alle angebotenen Simulationen selbst durchführen. Es ist zu empfehlen, dass einige wenige grundlegende Aufgaben gemeinsam unter Anleitung des Lehrers bearbeitet werden, da sich damit besonders viele Daten gewinnen lassen.
Dann sollten Arbeitsgruppen ihre Simulationen in geeigneter Form präsentieren.
Das Wunschziel könnte sein, dass die Schülerinnen und Schüler Zufallsexperimente (Glücksspiele, ...) genau analysieren und vielleicht zum Programmieren einfacher Simulationsmodelle angeleitet werden können.
Bei vielen Zufallsexperimenten ist zu empfehlen, dass sie mit einer sinnvollen und vertretbaren Anzahl von Versuchen tatsächlich durchgeführt werden (Münzen-, bzw. Würfelwurf, Ziehen von Spielsteinen aus einer "Urne", ...). Nur so kann ein Gefühl für das "Gesetz der großen Zahlen" entwickelt werden. Ganz wichtig ist die Erfahrung einer doch etwas längeren Versuchsserie (siehe die "Runs" aus Teil 1).

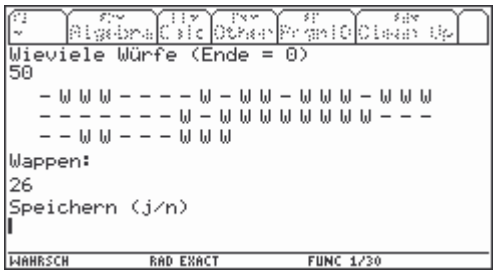
Auf sorgfältige Dokumentation (Strichlisten, ...) und graphische Darstellung ist zu achten. Neben händischer Aufbereitung kann auch auf verfügbare Tabellenkalkulationsprogramme, bzw. auf die grafischen Möglichkeiten der verfügbaren Taschenrechner zurückgegriffen werden.

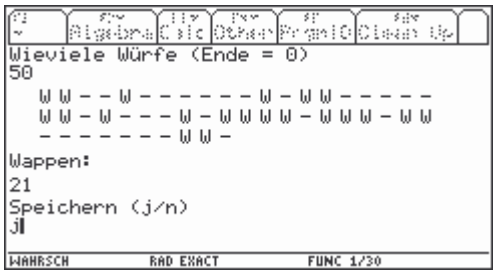
Aufgabenstellung

Zu den diskreten Verteilungen lassen sich leicht Zufallsexperimente erfinden. (Das kann bis zum Zählen von Autos an einer Tankstelle oder an einer bestimmten Kreuzung gehen).

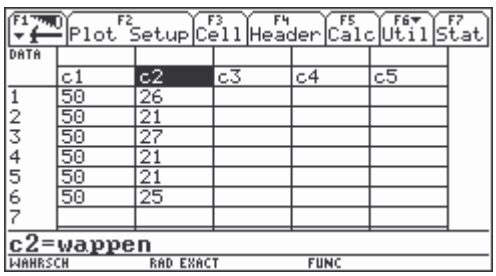
`muenze()`

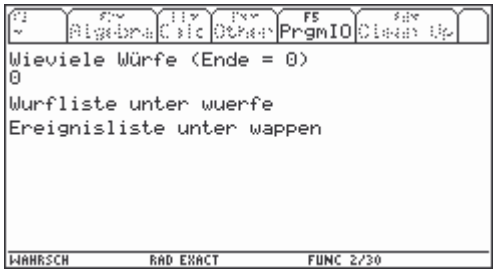
Das wohl einfachste Experiment ist das Werfen einer Münze. `muenze` ist ein einfaches Programm zu Simulieren des Münzwurfs mit anschließender - nicht automatisierter - eigenständiger Weiterbearbeitung der Ergebnislisten.



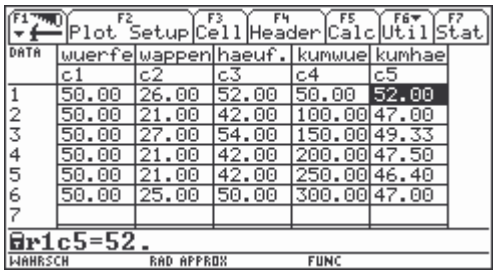


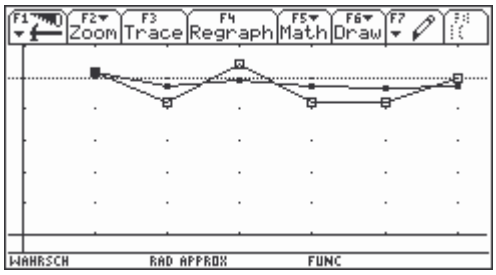
nach einer größeren Anzahl von Wurfserien .....





.... können die Daten in den Data-Editor übertragen werden. Sie werden in den Listen `wuerfe`, bzw. `wappen` abgelegt und dienen als Basis für die grafische Darstellung der Häufigkeiten und der kumulierten Häufigkeiten.





Die Boxes zeigen die Häufigkeiten für jeweils eine Serie, während die Squares die jeweils kumulierten Häufigkeiten angibt, die sich an den Wert 50% (punktiert) heranpirschen sollen.

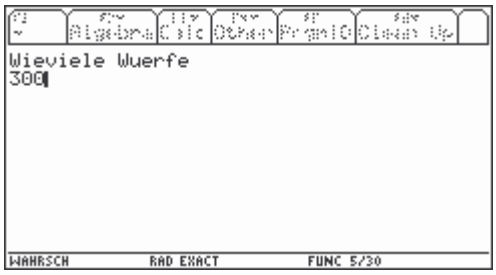
Wenn mehrere Gruppen arbeiten lassen sich alle Teilergebnisse kombinieren. Mehrer Versuchsläufe und Auswertungen zeigen die Konvergenz an das vermutete Ergebnis von 50% (hier punktiert dargestellt).

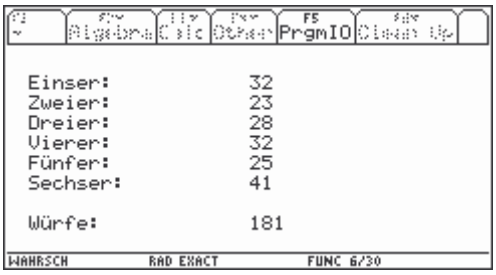
Aufgabe:

Verändere das Programm, dass eine "gefälschte" Münze mit 55% Wahrscheinlichkeit für WAPPEN simuliert wird.

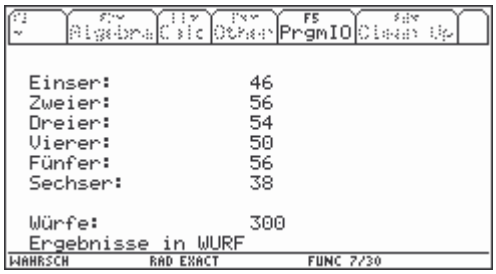
```
wuerfel()
```

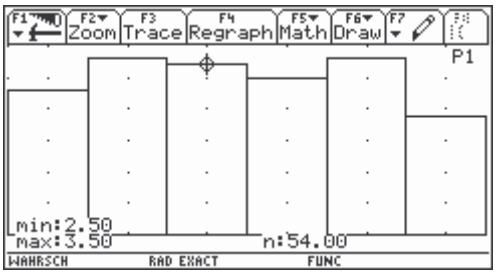
Bei der Einführung der Wahrscheinlichkeitstheorie kommt man sicher nicht um das "Glückspiel-urelement" Würfel herum. Die gewonnenen Daten können dazu verwendet werden, elementare Regeln für zusammengesetzte Wahrscheinlichkeiten empirisch zu erfahren und anschließend theoretisch zu belegen.





dies ist der Schirm während der Simulation.





Aufgaben:

Rechne die absoluten Häufigkeiten in relative um. Führe mehrere Simulationen durch und berechne die kumulierten Häufigkeiten.

Ermittle die Häufigkeiten (und dann die theoretischen Wahrscheinlichkeiten - oder umgekehrt) für die folgenden Ereignisse: Man wirft

1 oder 6,

mindestens die Augenzahl 4,

höchstens die Augenzahl 3,

eine gerade Augenzahl,

eine durch 3 teilbare Augenzahl,

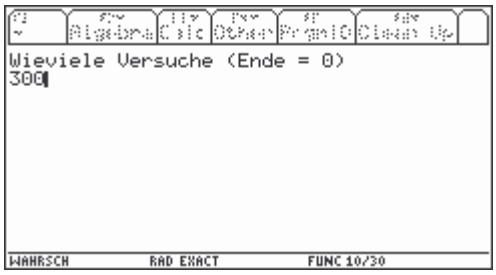
eine gerade oder eine durch 3 teilbare Augenzahl,

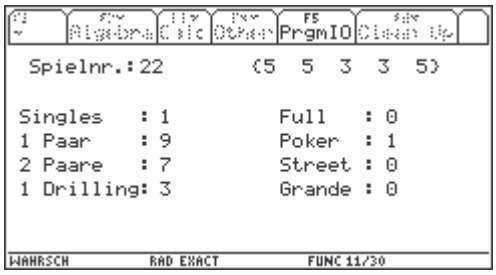
.....

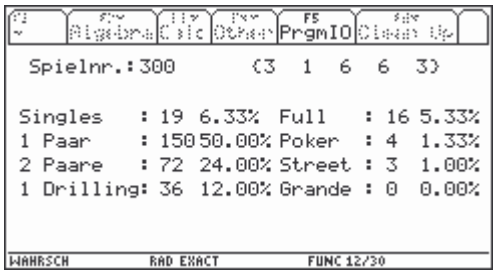
Weit verbreitet unter verschiedenen Namen wie "Kniffel" ist das "Würfelpokern". Ein geeignetes Programm lässt auch Würfel zu, die nicht 6 Seitenflächen aufweisen. Dann können wir vielleicht einmal "Tetraederpokern" oder auch "Oktaederpokern".....

`wpoker(s)`

Simuliert das Würfelpokern mit 5 "s"-seitigenWürfeln. `wpoker(6)` ergibt den folgenden Output:







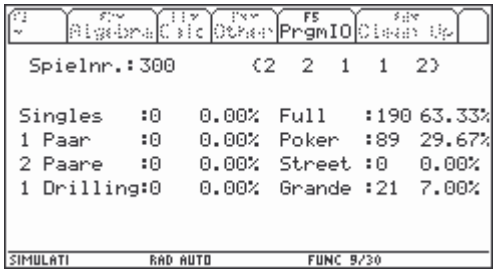
hier wurde gerade ein "Full House" erzielt.

Aufgaben:

Prüfe die simulierten Häufigkeiten mit den Mitteln der Wahrscheinlichkeitsrechnung.

Simuliere 300 Würfe mit einer Münze.

Simuliere das Werfen mit einem Tetraeder.



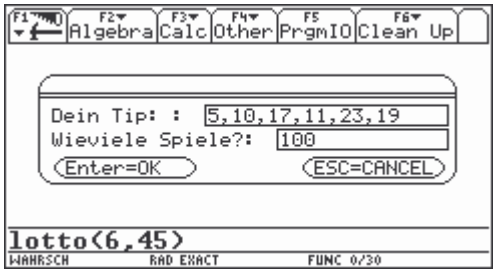
Das klassische Glücksspiel für den "kleinen Mann" ist wohl das Zahlenlotto. Verschiedene Varianten lassen sich mit `lotto` simulieren. Der Spieler gibt seinen Tipp ab und kann verfolgen, welcher Erfolg
ihm bei einer frei wählbaren Anzahl von Spielen beschieden ist.

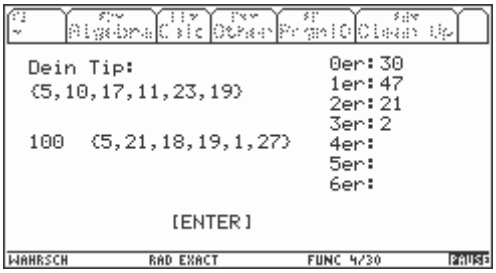
Die grafische Aufbereitung der Häufigkeit der einzelnen Nummern ist bereits im Programm integriert.

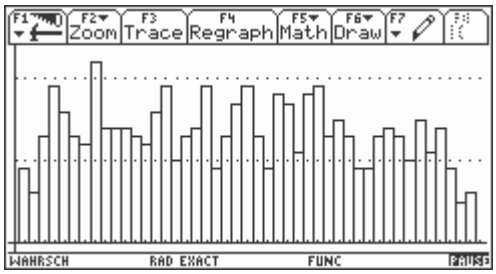
Als **Aufgabe** würde sich anbieten das Fußballtoto zu simulieren.

`lotto(klassengr, losgr)`

`lotto(6,45)` simuliert die österreichische Version "6 aus 45".

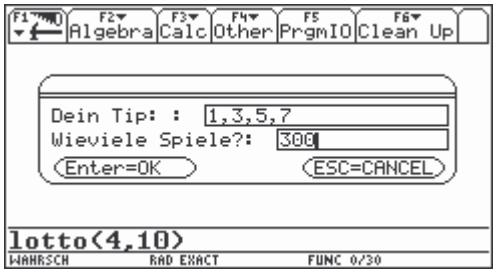


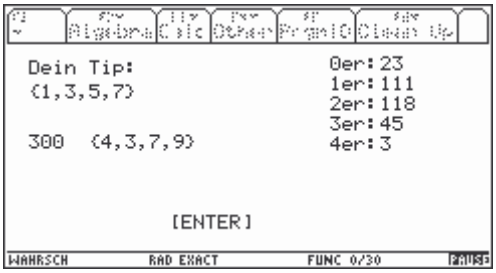


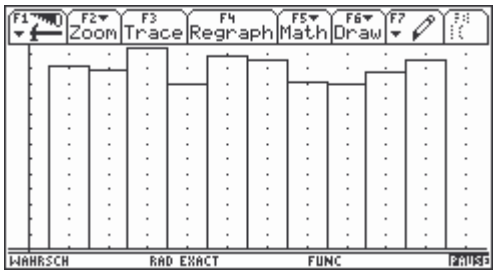


Hier wird die Häufigkeitsverteilung der Ziffern von 1 bis 45 für die 100 Versuche als Histogramm dargestellt.

`lotto(4,10)`



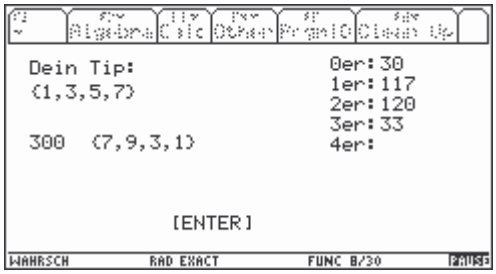


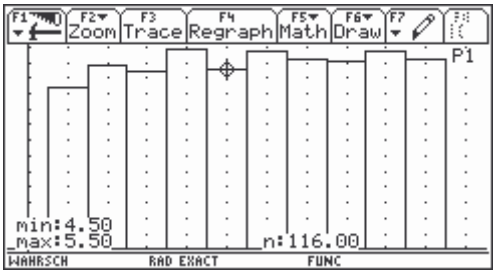


Die theoretischen Werte sind (gerundet):

22, 114, 129, 34, 1

Eine nächste Simulation ergibt andere Häufigkeiten. (über F3 - Trace können die absoluten Häufigkeiten der einzelnen Nummern abgefragt werden: der 5er trat 116 mal auf.

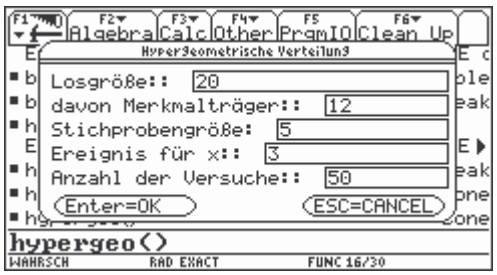


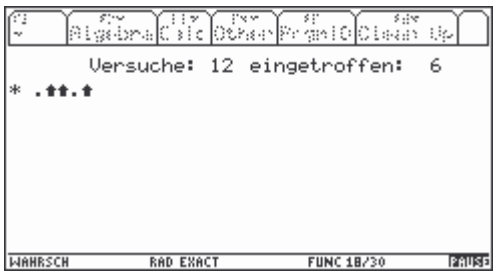


Die diskreten Verteilungen

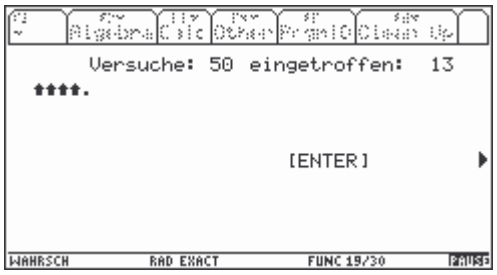
Die hypergeometrische Verteilung

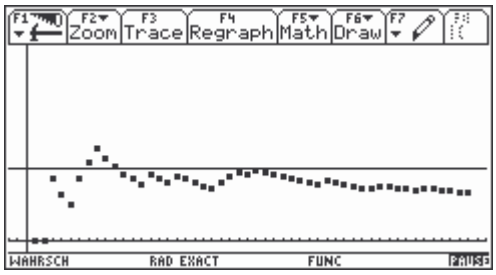
`hypergeo()`



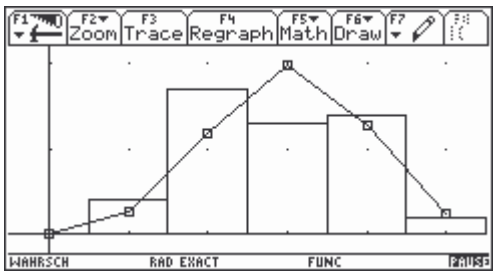


Nach der Eingabe der Daten wird ein beliebig ausgewähltes Ereignis besonders betrachtet. (Hier: 3 Merkmalträger in der Probe.) Alle Stichproben werden rasch angezeigt. Diejenigen, bei denen das Ereignis zutrifft mit einem Stern gekennzeichnet und mitgezählt.



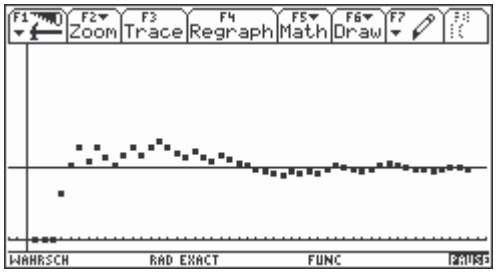


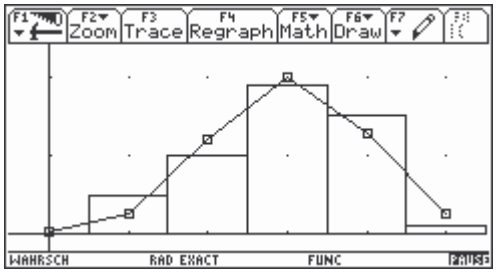
die kumulierten Häufigkeiten zusammen mit der theor. Wahrscheinlichkeit für das gewählte Ereignis.



Anschließende Grafik:

Hier sieht man das Histogramm der simulierten Häufigkeiten und dazu die theoretischen Häufigkeiten als Polygonzug.

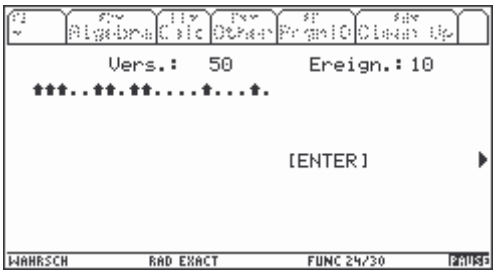


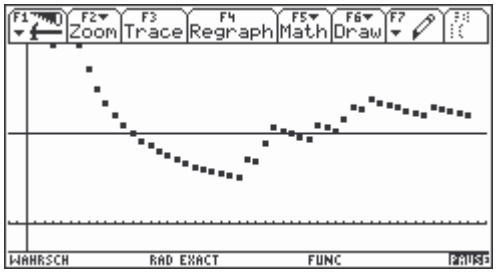


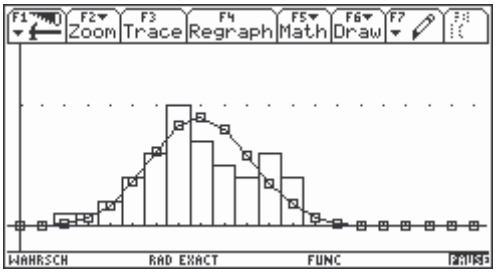
Der nächste Simulationslauf mit gleichen Daten liefert ein "viel schöneres Ergebnis".

Die Binomialverteilung

```
binomial()
```



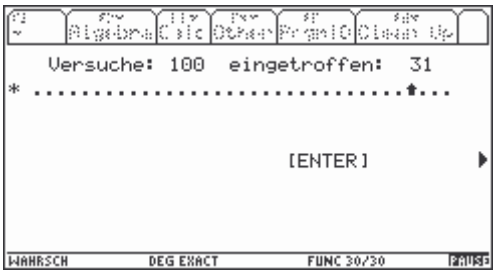


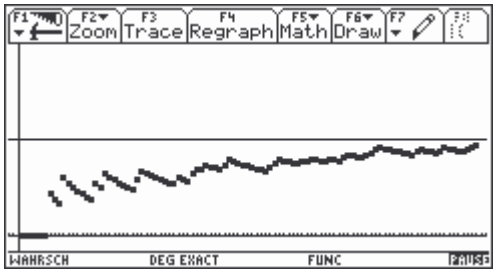


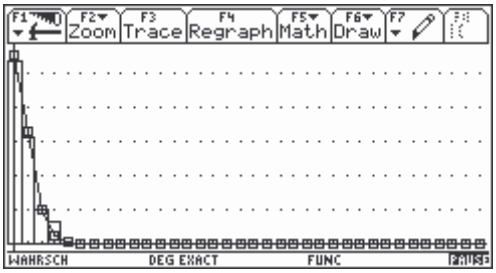
Hierzu ist wohl kein Kommentar notwendig!

Die Poisson Verteilung

```
poisson()
```







Wie erzeugt man Zufallszahlen, die einer vorgegebenen Verteilung gehorchen?
Wie testet man die Qualität dieser Zufallszahlen?

Es soll eine - beliebig lange - Folge von Zufallszahlen X ausgegeben werden, die der folgenden Verteilung gehorchen:

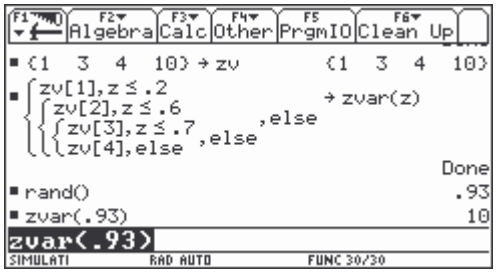
X_i	$X_1 = 1$	$X_2 = 3$	$X_3 = 4$	$X_4 = 10$
$p(X_i)$	0,20	0,40	0,10	0,30

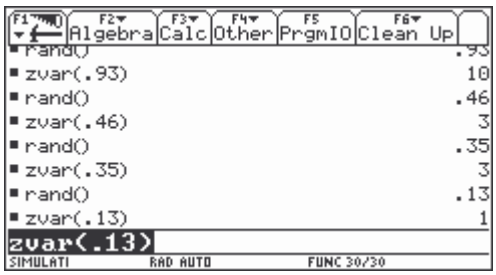
Wir erzeugen die Verteilungsfunktion:

x_i	X_1	$X_1 \cup X_2$	$X_1 \cup X_2 \cup X_3$	$X_1 \cup X_2 \cup X_3 \cup X_4$
$F(x_i)$	0,20	0,60	0,70	1,00

Mit `rand()` gibt der TI-92 eine (Pseudo)Zufallszahl z mit $0 < z < 1$ aus, von der wir vorerst annehmen, dass sie in diesem Intervall "gleichverteilt" ist.

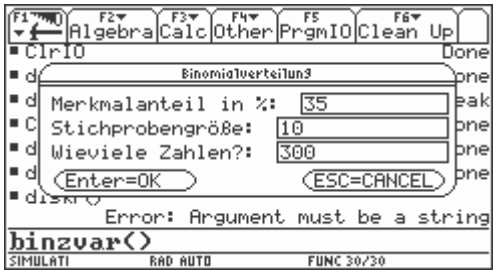
Ein geschachtelte Abfrage kann nun jeder dieser Zufallszahlen z eine der vier Zufallsvariablen zuordnen:

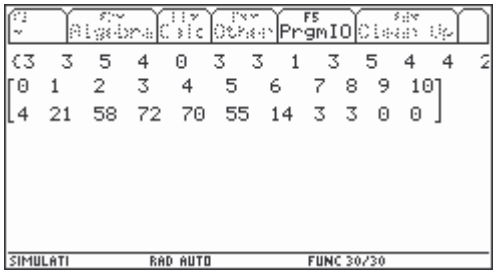




Im Home Screen wird mit einer `when(` - Konstruktion die Abfragetechnik dargestellt. Die Funktion zvar(z) erzeugt zu jeder Zufallszahl z die entsprechende Ausprägung der Zufallsvariablen x, die aus der Liste zv gewonnen wird.

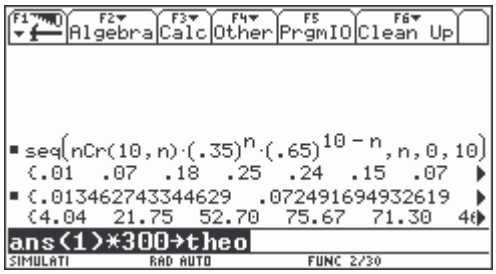
Diese Technik wird nun in einem - nicht allzu umfangreichen - Programm angewendet, mit dessen Hilfe sich binomial verteilte Zufallsvariable erzeugen lassen. Die so gewonnenen Listen eignen sich dann dazu, binomiale Zufallsprozesse zu simulieren.

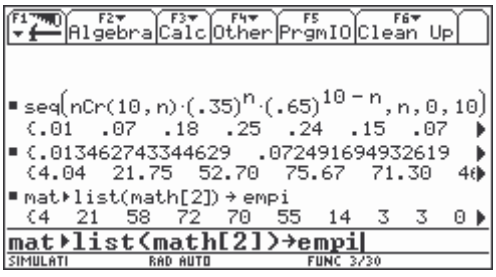




```
binzvar()
Prgm
Local p,n,z,lv,lw,zu,i
Dialog
Title
 "Binomialverteilung"
Request
 "Merkmalanteil in %",p
Request
 "Stichprobengroeße",n
Request
 "Wieviele Zahlen?",z
EndDlog
expr(p)/100→p
expr(n)→n
expr(z)→z
newList(n+1)→lw newL-
ist(n+1)→lv
newMat(2,n+1)→math
For i,1,n+1
i-1→math[1,i]
EndFor
{}→lz
For  i,0,n
nCr(n,i)*p^i*(1-p)^(n-i)
→lw[i+1]
Σ(lw[j],j,1,i+1)→lv[i+1]
EndFor
For j,1,z
rand()→zu
For  i,1,n+1
If zu≤lv[i] Then
  augment({i-1},lz)→lz
  math[2,i]+1→math[2,i]
  Goto neu
EndIf
EndFor
Lbl  neu
EndFor
Disp  lz
Disp  math
EndPrgm
```

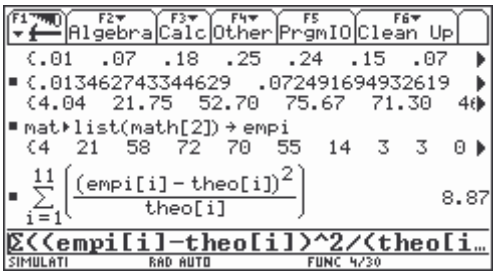
Zum Testen der Qualität dieser Zahlen führen wir den sogenannten χ^2-Test durch. Dazu müssen wir die Abweichungen der empirischen Häufigkeiten von den theoretischen betrachten. Die theoretischen Häufigkeiten lassen sich leicht mit der Binomialverteilung berechnen.





Unter dem Namen `math[2]` steht uns die Liste der absoluten Häufigkeiten der Zufallszahlen zur Verfügung! Wir wandeln diesen Zeilenvektor in die Liste `empi` um und berechnen dann das χ^2 nach der Formel:

$$\chi^2 = \sum_{i=1}^{11} \frac{(\text{empi}_i - \text{theo}_i)^2}{\text{theo}_i}$$



Mit diesem Wert testen wir die Hypothese H_o, dass die Zufallsvariable der binomischen Verteilung gehorcht. Große Werte für χ^2 machen diese Hypothese verdächtig. Aber was heißt schon groß?

Als eine Faustregel kann gelten: ist $\chi^2 < f + \sqrt{2f}$, dann besteht kein Anlass, an der Hypothese H_o zu zweifeln. Dabei ist f die "Anzahl der Freiheitsgrade", das ist die Anzahl der frei wählbaren Werte der Häufigkeiten, das sind 11 - 1 = 10, da die Summe mit 300 ja erreicht werden muss. (Mit 10 Häufigkeiten ist automatisch auch die 11. gegeben).

$f + \sqrt{2f} = 10 + \sqrt{20} = 14{,}47$; da 8,87 < 14,47 können wir mit ziemlich großer Sicherheit nicht ausschließen, dass die Zahlen der vermuteten Verteilung gehorchen. (Damit ist aber nicht bewiesen, dass sie es tun!!!)

Der χ^2-Test wird genauer mit Tabellen durchgeführt. Diese Behandlung sprengt aber den Rahmen und es wird auf die zahlreiche Literatur verwiesen.

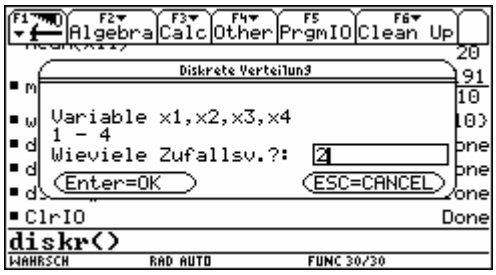
Mit dem nächsten Programm können Mittelwert und Varianz von Zufallsvariablen, die aus einer Linearkombination (oder einer anderen Zusammensetzung) von bis zu vier Zufallsvariablen empirisch untersucht werden.

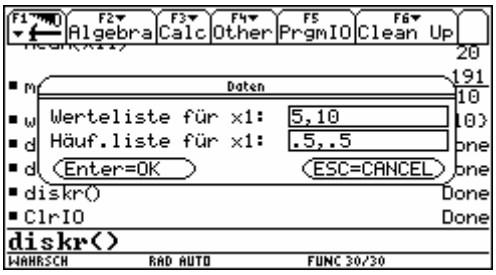
diskr()

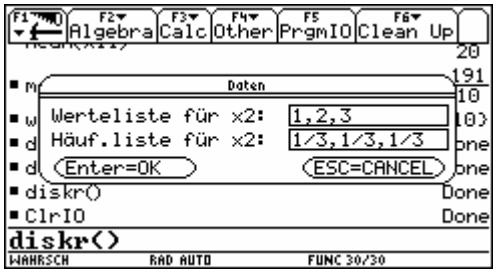
Bis zu vier Zufallsvariablen und deren Verteilungen werden eingegeben. Eine neue Zufallsvariable wird aus diesen durch Simulation einer Anzahl von Versuchen wird bestimmt und die Parameter dieser Verteilung werden ausgegeben (Mittelwert und Varianz).

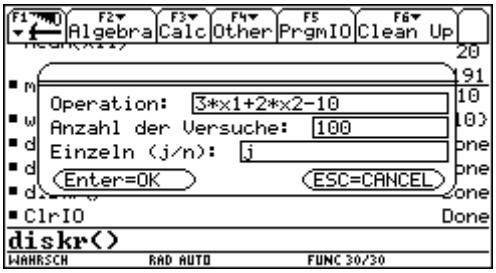
z.B. x1 = {5, 10} mit den Wahrscheinlichkeiten {0.5; 0.5}
x2 = {1,2,3} mit den Wahrscheinlichkeiten {1/3, 1/3, 1/3}

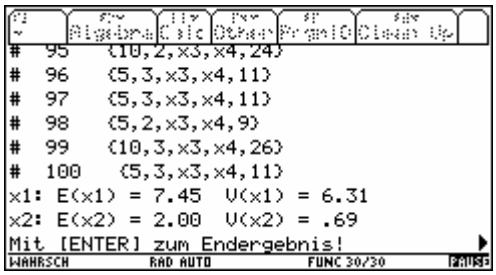
Gesucht ist die Verteilung der neuen Zufallsvariablen: 3*x1 + 2*x2 – 10

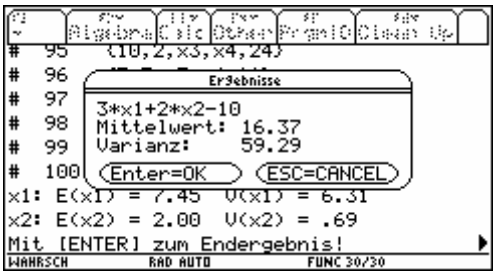






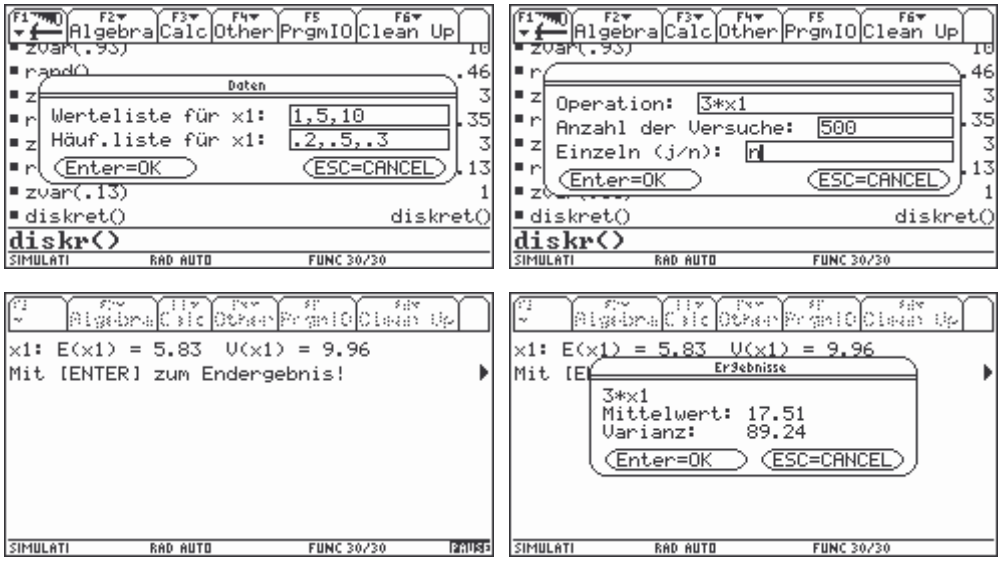




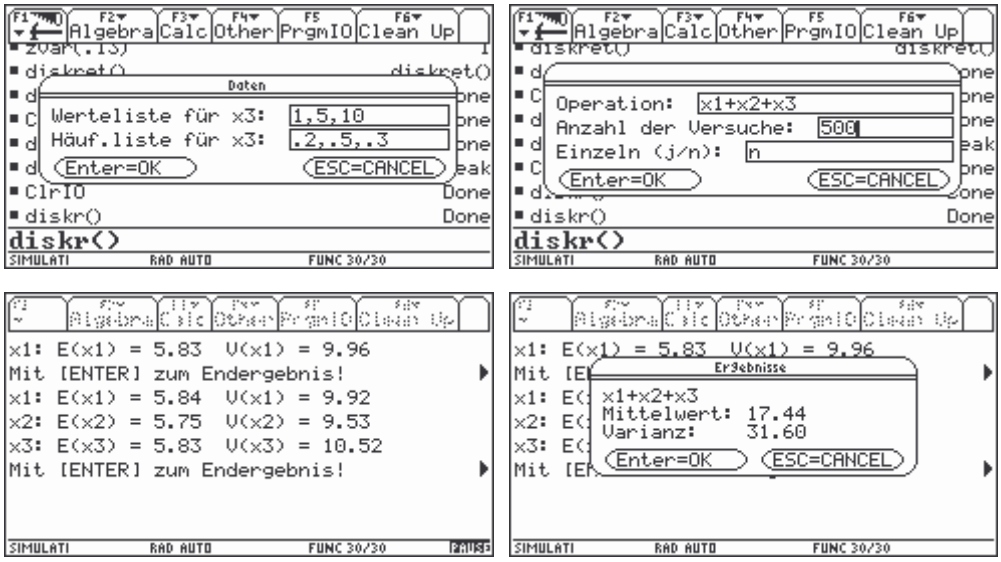


Es lässt sich auch der Unterschied von U = 3*X und V = X + X + X wirkungsvoll demonstrieren.

Für U wird eine Variable {1,5,10} mit den Wahrscheinlichkeiten {0.2, 0.5, 0.3} eingegeben, während für V diese Werte insgesamt dreimal für x1, x2 und x3 verwendet werden müssen.



Der Mittelwert von U = 3X sollte das Dreifache des Mittelwerts von X sein, die Varianz das Neunfache.



Der Mittelwert von V = X + X + X, wobei jedes X für eine eigenständige Ausprägung der Zufallsvariablen steht. sollte die Summe der Mittelwerte von X sein, die Varianz das **Drei**fache.

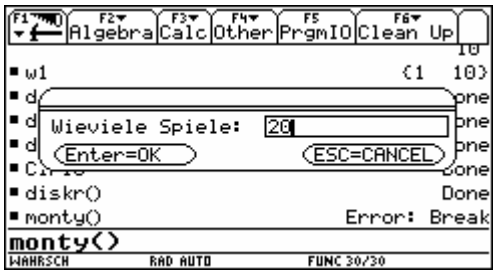
Wie lässt sich der Unterschied zum obigen Ergebnis plausibel erklären?

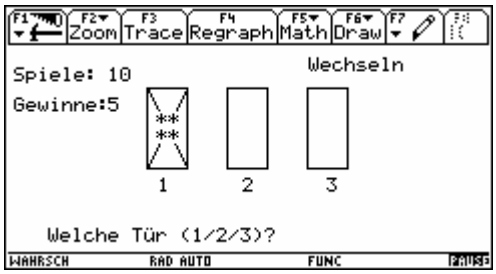
Aufgabe:

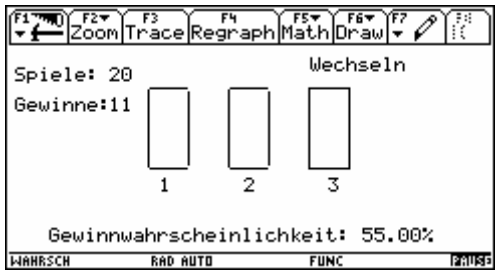
Bestimme Mittelwert und Varianz von U = 3X - 2Y und versuche den Zusammenhang zwischen Mittelwerten und Varianzen herzustellen.
Wie wirkt sich die Addition einer Konstanten zu einer Zufallsvariablen auf Mittelwert und Varianz aus?

Zum berühmten "**Ziegenproblem**" (siehe Papier von Benno Grabinger) könnte man auch eine grafische Simultion anbieten. Der Name `monty` rührt von einer amerikanischen Gameshow mit dem Showmaster Monty Hall her, der dieses Spiel berühmt gemacht hat. Im zitierten Papier findet sich auch eine genaue Beschreibung des Spiels und eine analytische Ableitung des hier empirisch gefundenen Ergebnisses.

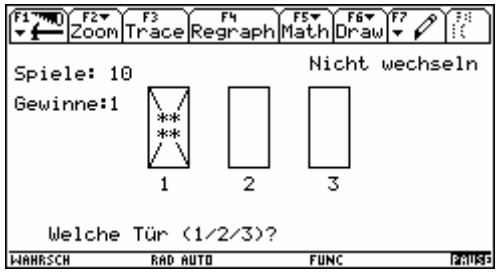
```
monty()
```







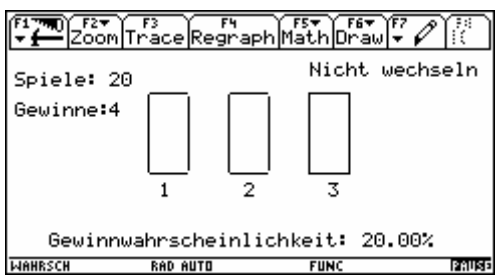
Hier wurde eben kein Gewinn erzielt! (Der Spieler wählte 1, Monty öffnete 2 oder 3, der Spieler **wechselte** zu 3, bzw 2, der Gewinn war aber hinter Tür 1.)



Hier wurde der Gewinn erreicht!

(Der Spieler wählte wiederum Tür 1, Monty öffnete die leere Tür 2 oder 3, der Spieler blieb bei Tür 1, hinter der auch der Gewinn auf ihn wartete).

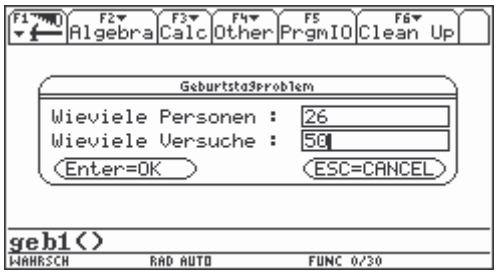
Die Simulation zeigt deutlich, welche Strategie die günstigere zu sein scheint.

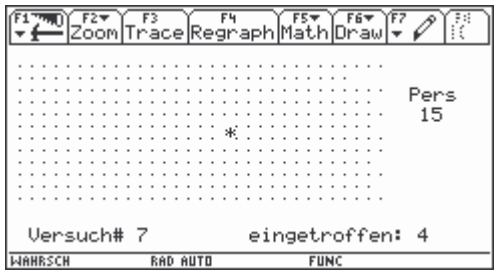


Ein Problem, das in keinem Wahrscheinlichkeitstheorielehrbuch fehlen darf, ist das

Geburtstagsproblem

`geb()`





Wie groß ist die W., dass unter 26 SchülerInnen einer Klassen mindestens 2 den gleichen Geburtstag haben. Die Pünktchen stell das Jahr mit 365 Tagen dar. Nach 6 Versuchen ist das Ereignis 4 mal eingetroffen. Im 7. Versuch hat Person 15 gerade einen „Geburtstagspartner“ gefunden. (Es waren dann insgesamt 22).

Aufgaben:

Spiele die Simulation für die Klasse durch. (Vergleich mit der Wirklichkeit, bestimme auch die theoretische Wahrscheinlichkeit!)

Simulation für den ganzen Jahrgang einer Schule. (Vergleich mit der Wirklichkeit, bestimme auch die theoretische Wahrscheinlichkeit!)

Wie sieht es mit den Personen des Lehrkörpers an der Schule aus?

Die Idee und Grundlagen zur folgenden umfangreichen Untersuchung stammt von Wilfried Rohm, AMMU-Ausendung 4, Mai 1994. Weitere Grundlagen finden sich u.a. in Arthur Engel, Wahrscheinlichkeitsrechnung und Statistik, Bd 1 und Bd 2, Klett Studienbücher.

Hier wäre dringend zu raten, längere Zufallsfolgen durch einen Münzwurf tatsächlich zu erzeugen und auszuwerten.

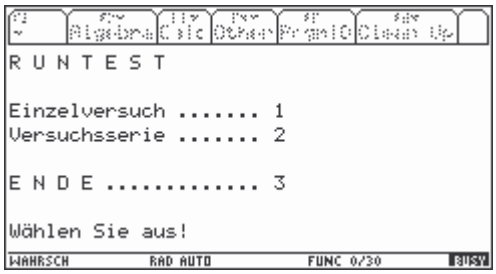
Es hat sich auch bewährt, einige Schüler eine Zufallsfolge von (Kopf/Adler) nur hinschreiben zu lassen, sie erzeugen erfahrungsgemäß zu viele Runs. Der Lehrer kann allen jenen, deren Runanzahl außerhalb von $\frac{n+1}{2} \pm \sigma = \frac{n+1}{2} \pm \sqrt{\frac{n-1}{4}}$ liegt, mit ca 70% Sicherheit auf den Kopf zusagen, dass sie nicht ehrlich geworfen haben.

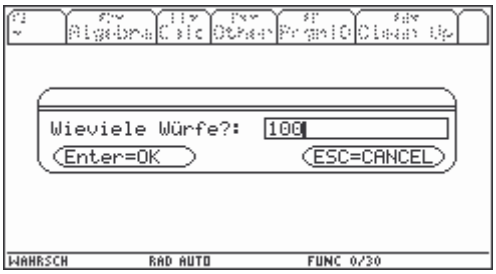
Siehe dazu Auswertung 2:RUNS im Rahmen der nächsten Simulation.

Der Runtest

Unter einem Run in einer Folge 00001001101000111 versteht man die Anzahl von Ketten gleicher Zeichen. Hier gibt es 8 Runs.
In einer zufälligen Folge von Bits gibt es eine Menge interessanter Untersuchungen. Derartige Folgen werden mit `runtest()` simuliert. Die mittlere Anzahl eines Zeichens und die mittlere Runanzahl sind nur zwei der interessanten Ergebnisse.

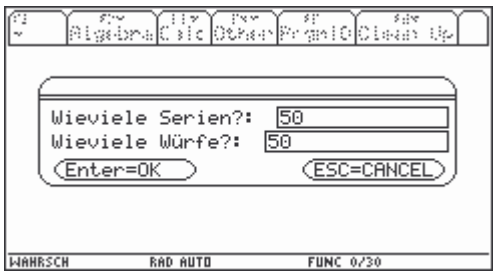
`runtest()`

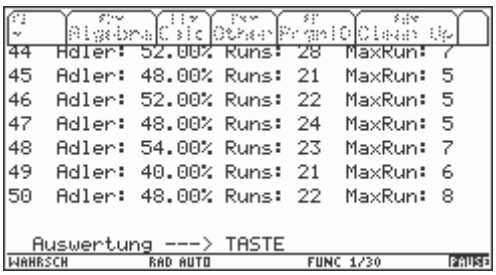




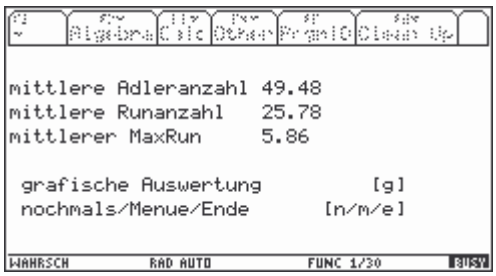
Ein Versuch mit hundert Würfen einer Münze:

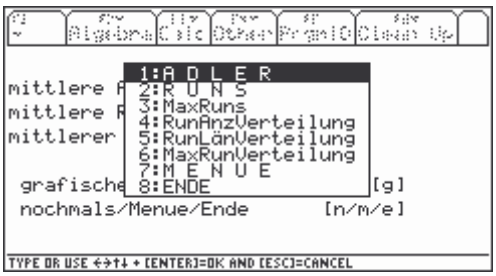
Anschließend eine Serie von 50 mal 50 Würfen und deren Auswertung nach verschiedenen Kriterien:





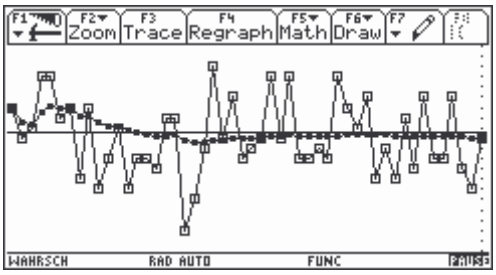
so sieht´s unterwegs aus.





`1:ADLER`

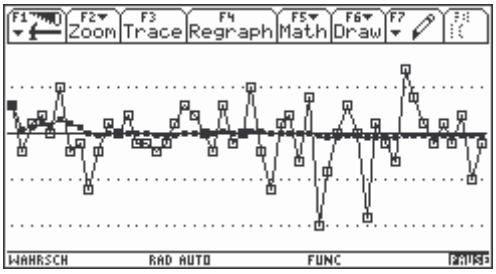
stark oszilierend die jeweiligen Ergebnisse pro Versuch und schließlich konvergierend die kumulierten Häufigkeiten.



`2:RUNS`

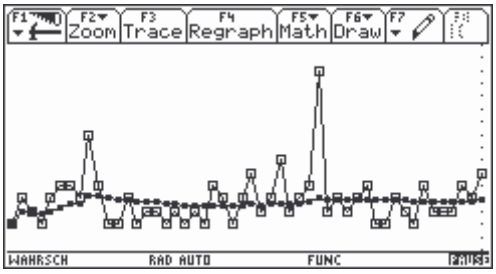
Die mittlere Anzahl der Runs konvergiert gegen 25,5.
Der theoretische Wert beträgt

$$E(R) = \frac{n+1}{2}$$



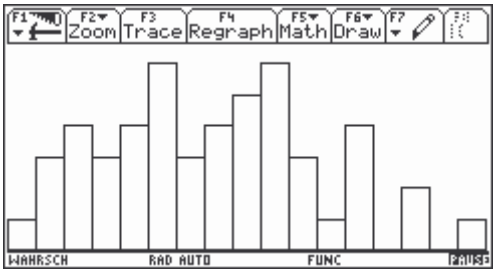
`3:maxRuns`

zeigt die längsten Runs pro Versuch und wie deren
Durchschnittswert entsteht, der offensichtlich
wieder konvergiert.



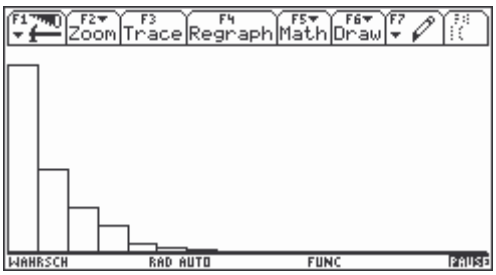
`4:RunAnzVerteilung`

zeigt die Häufigkeit der Anzahlen der Runs in
einer Versuchsserie. (Verteilung??)



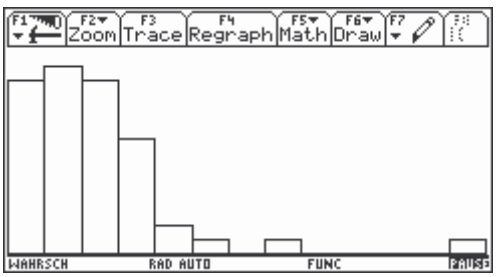
`5:RunLaenVerteilung`

gibt ein Histogramm für die Häufigkeiten der Runlängen in allen Versuchen (Runs der Länge 1, der Länge 2, usw.)
Die Balkenhöhen scheinen sich zu halbieren??!!
und schließlich zeigt



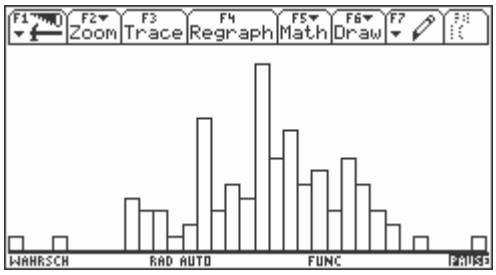
`6:MaxRunVerteilung`

die Häufigkeit der längsten Runs.
(Beschriftung könnte dann per Hand durchgeführt werden!!)



Bei hundert Versuchen zu je hundert Würfen werden bekannte Wahrscheinlichkeitsverteilungen („Glockenform“) schon eher erreicht.

runAnzVerteilung



und MaxRunVerteilung