Programmieren mit TI-*Nspire*-CAS

Josef Böhm

bk teachware Schriftenreihe Nr. SR−60, ISBN 978−3−901769−81−8
eMail kontakt@bk−teachware.com

Die TI-*Nspire*-Dateien zu den Abschnitten können von der Adresse `http://shop.bk-teachware.com/BKT-G60` bezogen werden.

Vorwort

Vor geraumer Zeit habe ich in dieser Serie von CAS-orientierten Büchern ein Buch zum *Programmieren mit Derive* veröffentlicht. Diesem Buch war ein relativ großer Erfolg innerhalb der deutschsprachigen *Derive*-Gemeinde beschieden.

Zum Programmieren mit dem TI-92 bzw. Voyage 200 gibt es auch einige sehr gut brauchbare Materialien. Leider wird dem Programmieren nur mehr sehr wenig Bedeutung im täglichen Gebrauch in der Schule zugemessen, da es für (fast) alle Zwecke zum Teil ausgezeichnete fertige Software gibt.

Dabei wird aber nicht beachtet, dass das Umsetzen eines Algorithmus in eine – welche auch immer – Programmiersprache einen sehr wichtigen Aspekt des Mathematikunterrichts beleuchtet, nämlich das Umsetzen eines an sich bekannten Prozesses in eine andere Sprache. Dazu ist es aber notwendig, diesen Prozess – und sei er noch so einfach – ganz genau zu analysieren, inklusive von Sonderfällen, die oft gar nicht beachtet werden („weil das eh nicht passiert, wenn man´s mit der Hand rechnet!“).

Das Arbeiten mit Schleifen und Abfragen, die Berücksichtigung von Plausibilitätskontrollen, eine benutzerfreundliche Eingabe und eine klare Ausgabe der Ergebnisse sind wichtige Punkte bei der Erstellung eines ordentlichen Programms.

Das Programmieren mit dem TI-*Nspire* war bis zur Version 1.2 eine sehr mühsame Sache, da vor allem das Editieren des Programms – auch mit der PC-Version nicht recht bequem von statten ging. Ab der Version 1.3 wird ein sehr handlicher Editor geboten, mit dem sich flott arbeiten und testen lässt.

Alles, was in diesem Büchlein mit der PC-Version gezeigt wird, lässt sich vollkommen identisch auch am Handheld durchführen. Alle Programme/Funktionen können über das *Link*-Programm von einem Medium auf das andere übertragen werden.

Als weitere Neuerung gibt es nun die Möglichkeit, eigene Programmbibliotheken zu erstellen, die den Zugriff auf wichtige Hilfsprogramme (zB zur Statistik, Analysis, Geometrie usw.) aus allen Dokumenten erlaubt.

Der Vorrat an Befehlen ist zwar eine erweiterte Fassung der TI-89/TI-92/Titanium/ Voyage 200/ Befehle aber es fehlt die Möglichkeit, die Grafikseite (die *Graphs & Geometry*-Applikation) zu programmieren. Über Umwege lässt sich aber doch Einiges erreichen. Als größeren Mangel empfinde ich, dass in der aktuellen Version 1.3 eine interaktive Eingabe über selbst erstellte Eingabemasken und/oder –menüs nicht möglich ist.

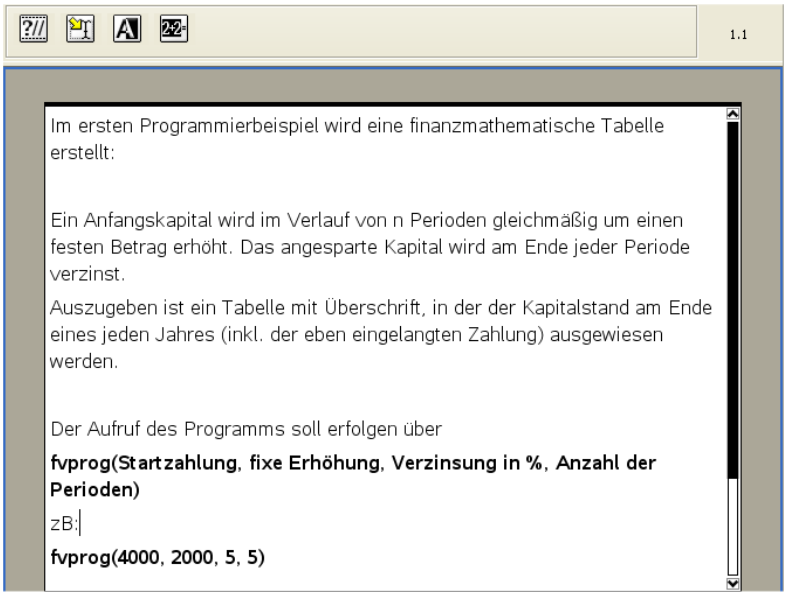
Aber auch ohne diese genannten Möglichkeiten bietet TI-*Nspire* sehr viele Gestaltungsmöglichkeiten und es ist zu hoffen, dass dieses Büchlein zum Nachmachen und zum eigenen Programmieren anregt. Der Leser sollte mit dem elementaren Umgang mit TI-*Nspire* vertraut sein, dann wird der Einstieg in das „Abenteuer Programmieren“ kein Problem darstellen.

Josef Böhm

(Beachten Sie bitte die Download-Adresse beim Inhaltsverzeichnis!)

1 Anstelle einer Einführung: die Finanzen

Stilgerecht wird das erste Problem in den *Notes* vorgestellt.



Zu diesen möglichen Ausgaben wollen wir kommen:

fvprog(4000,2000,5,5)

"Periode"	"Kapitalstand"
0.00	4000.00
1.00	6200.00
2.00	8510.00
3.00	10935.50
4.00	13482.28
5.00	16156.39

Fertig

Ausgabe des Programms

fval(4000,2000,5,5)

"Periode"	"Kapitalstand"
0.	4000.
1.	6200.
2.	8510.
3.	10935.5
4.	13482.3
5.	16156.4

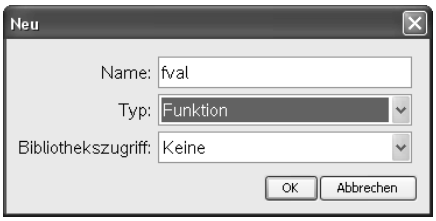
Ausgabe der Funktion

Wenn im Folgenden von Programmen gesprochen wird, sind vorerst immer Funktionen und Programme gemeint. Auf die Unterschiede zwischen diesen beiden Arten von Routinen, die aus einer Sammlung von Einzelbefehlen bestehen, wird an gegebener Stelle hingewiesen.

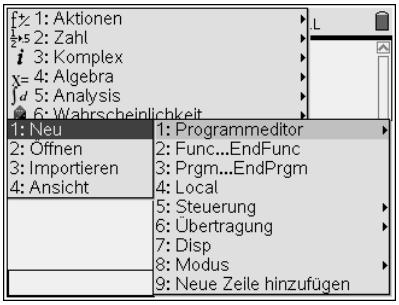
In der aktuellen TI-*Nspire* Version 1.3 ist ein komfortabler Programmeditor eingebaut. Eine weitere Neuerung besteht darin, dass Programmbibliotheken eingerichtet werden können, in denen Hilfsprogramme abgelegt werden. Diese Programme scheinen – wahlweise – auch im Funktionskatalog auf und können aus allen Applikationen aufgerufen werden. Hier gibt es aber schon einen wesentlichen Unterschied zwischen Funktionen und Programmen: nur Funktionen lassen sich in anderen Applikationen einsetzen (wie zB auch in *Lists & Spread-*

sheet, *Data & Statistics*, ja sogar in den *Notes*), während Programme nur im *Calculator* ausgeführt werden können. Manche Befehle lassen sich nur in Programmen einsetzen. Im Allgemeinen ist man mit Funktionen flexibler, weil das Ergebnis als „Funktionswert“ direkt weiter verwendet werden kann.

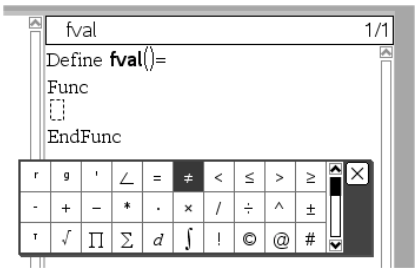
Der Editor kann als eigene Applikation sofort „eingefügt“ oder aus dem Calculator über die entsprechende Schaltfläche aufgerufen werden:

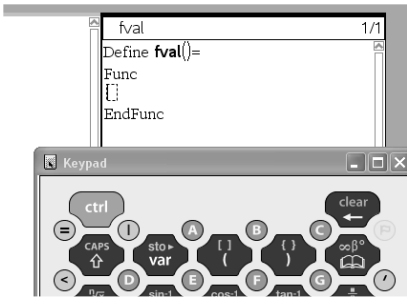


Als *Typ* können wir zwischen Funktion und Programm wählen. Den Bibliothekszugriff werden wir beim ersten Beispiel nicht gleich besprechen. Dass dies am „Taschenrechner“ genau so funktioniert, sehen Sie an den nächsten Bildern.



Nun öffnet sich das Fenster des Programmeditors und wir können wie in einem gewöhnlichen Texteditor den Programmcode eingeben. Für allfällige Sonderzeichen öffnen Sie entweder die Liste aller Symbole oder blenden - am PC - die Taschenrechnertastatur ein.

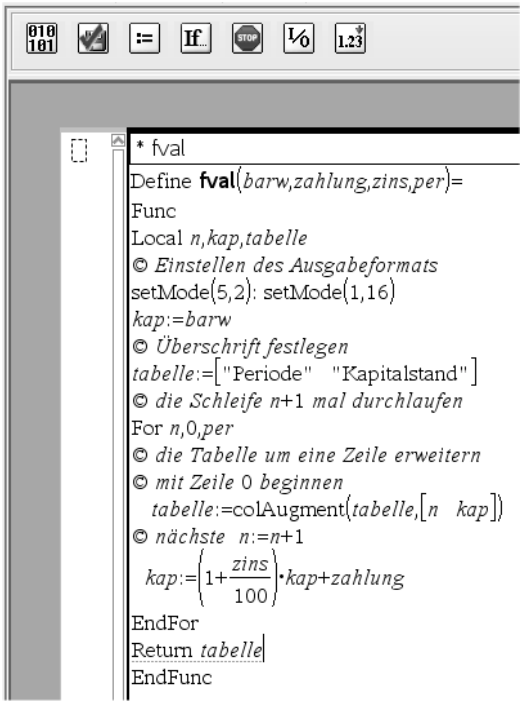




Ich vergrößere das Editorfenster und beginne mein Programm (hier meine Funktion) zu schreiben. Das Sternchen vor dem Programmnamen (*fval) zeigt an, dass diese Fassung noch nicht zwischengespeichert wurde und daher auch nicht evaluiert werden kann. Erst die Aktivierung des Häkchens in der Menüleiste prüft die Syntax und zwischenspeichert das Programm. Dann kann es im Calculatorfenster aufgerufen und getestet bzw. verwendet werden.

Ich klicke auf das Häkchen und bestätige den Menüpunkt

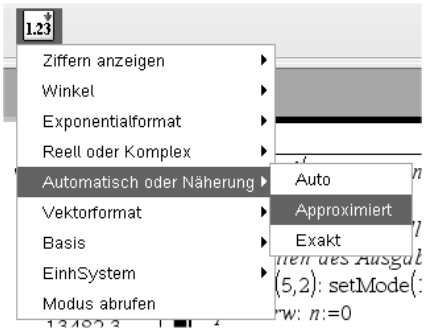
In diesem Fall wird kein Fehler gemeldet – was noch lange keine Garantie dafür darstellt, dass alles in Ordnung ist und dass das Programm genau das macht, was wir wollen. Reine syntaktische Fehler werden jedoch angezeigt.
Ein häufiger Fehler ist es, nach einer Änderung auf das Kontrollhäkchen zu vergessen. Die endgültige Speicherung erfolgt aber erst mit dem kompletten Dokument.

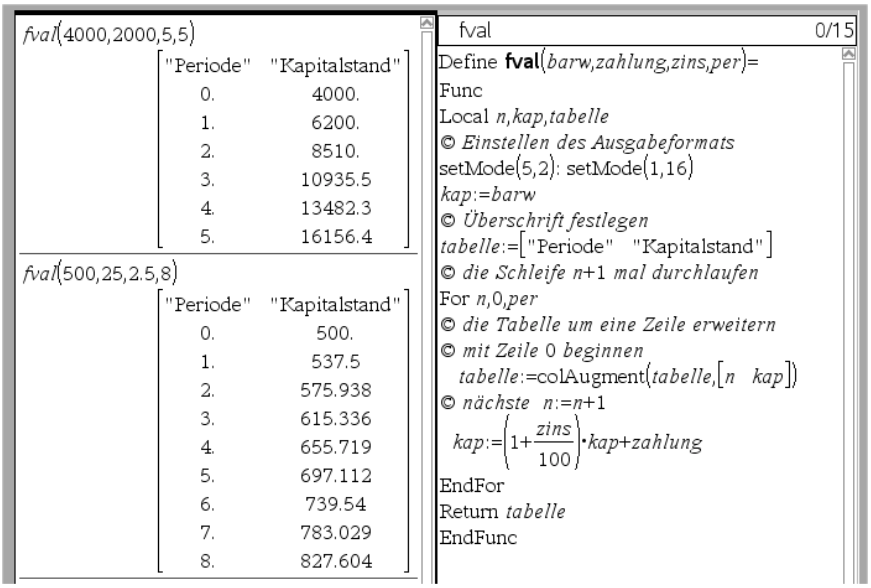


Nun können wir die Funktion testen. Dazu wird das Calculatorfenster auf normale Größe aufgezogen und dort die Funktion mit geeigneten Parametern aufgerufen (nächste Seite).

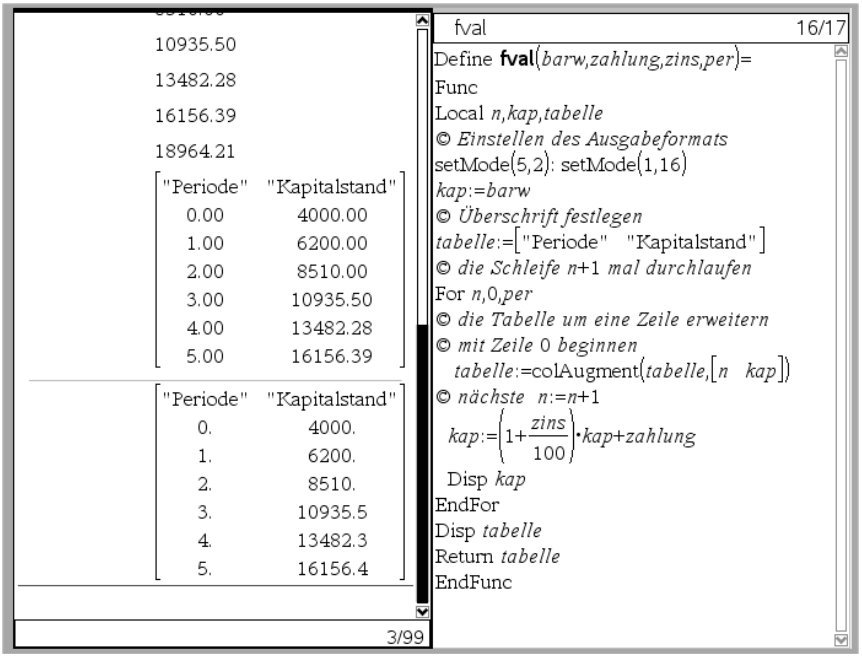
Wir sehen, dass alles bestens funktioniert. Die Zeilen, die mit einem © - erreichbar über 010 101 - eingeleitet werden, sind Kommentarzeilen, die vor allem der Programmdokumentation dienen sollen. Ich möchte gleich hier auf drei wichtige Zeilen aufmerksam machen.

Einstellungen für Berechnungs- und Ausgabeparameter werden in **setMode()**-Anweisungen festgelegt. Im Referenzteil kann man alle Möglichkeiten nachschlagen. Einfacher ist es, aus dem Editor die 1.23-Schaltfläche aufzurufen, dann öffnet sich ein Menü. Wenn Sie nun zB die gezeigte Wahl treffen, wird automatisch die Anweisung **setMode(5,2)** in den Programmcode eingefügt. **setMode(16,1)** legt das Format FIX 2 fest.





Leider scheint die **setMode**-Anweisung hier nichts zu bewirken. Oder doch? Ich baue in die zählergesteuerte **For−EndFor**-Schleife und nach ihr eine **Disp**-Anweisung ein, um das Zwischenergebnis *kap* und das Endergebnis *tabelle* ausgeben zu lassen. (Derartige Ausgaben von Zwischenergebnissen sind vor allem für eine allfällige Fehlersuche sehr gut geeignet.) Vor der Zeile **EndFor** füge ich also **Disp** *kap* und nach ihr **Disp** *tabelle* ein und rufe **fval(4000,2000,5,5)** nochmals auf.



Sie erkennen, was passiert: innerhalb der Funktion wird mit der durch **setMode** bestimmten Einstellung gearbeitet. Die Ausgabe des Funktionswertes selbst erfolgt aber mit den in den Dokumenteinstellungen festgelegten Formaten. Um mit dieser Funktion die Ausgabe mit zwei Dezimalstellen zu erzwingen bleibt uns also nichts anderes übrig, als die Dokumenteinstellungen im Datei-Menü entsprechend festzulegen.

Sie werden sich möglicherweise fragen, warum wir denn nicht nur die **Disp**-Anweisung stehen lassen und die **Return**-Anweisung weglassen? Gute Frage, aber ich lade Sie ein, das selbst zu probieren. Sie brauchen nur vor das **return** ein Kommentarzeichen einzufügen und die Funktion nochmals aufzurufen.

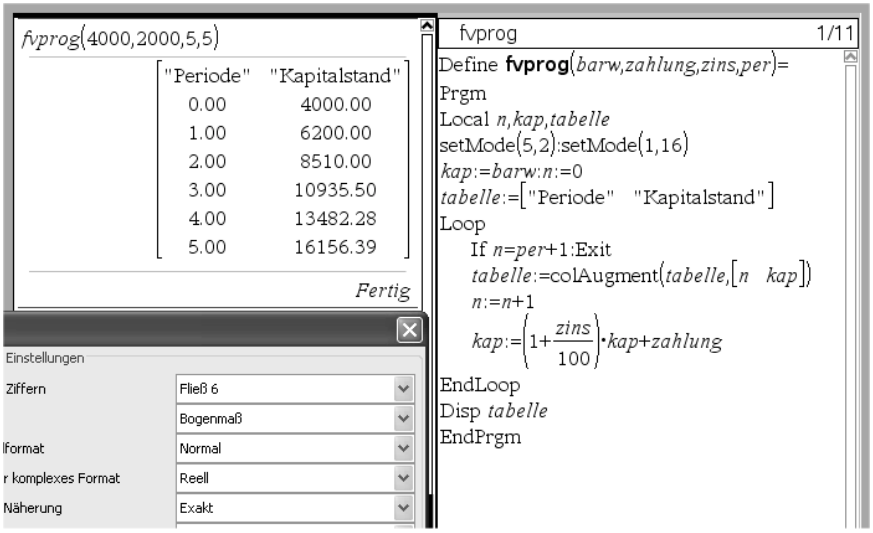
Ich werde in diesem Kapitel auf **setMode** nochmals zurückkommen!

Mein zweiter Hinweis an dieser Stelle gilt der Anweisung **colAugment**, die es ermöglicht, Zeilen an eine bestehende Matrix anzufügen. Das war bisher beim V200 bzw. TI-92 nicht möglich. In der vorletzten Zeile würde auch *tabelle* anstelle von **Return** *tabelle* reichen.

Der dritte Hinweis ist aber der wichtigste:

> Alle auftretenden Variablen – inklusive der Schleifenzähler, wie hier das *n* – müssen in Funktionen als lokale Variable am Beginn der Funktion als lokale Variable definiert werden. Sonst handeln Sie sich eine Fehlermeldung ein.

Nun will ich noch ganz rasch die Aufgabe in Form eines Programms behandeln:



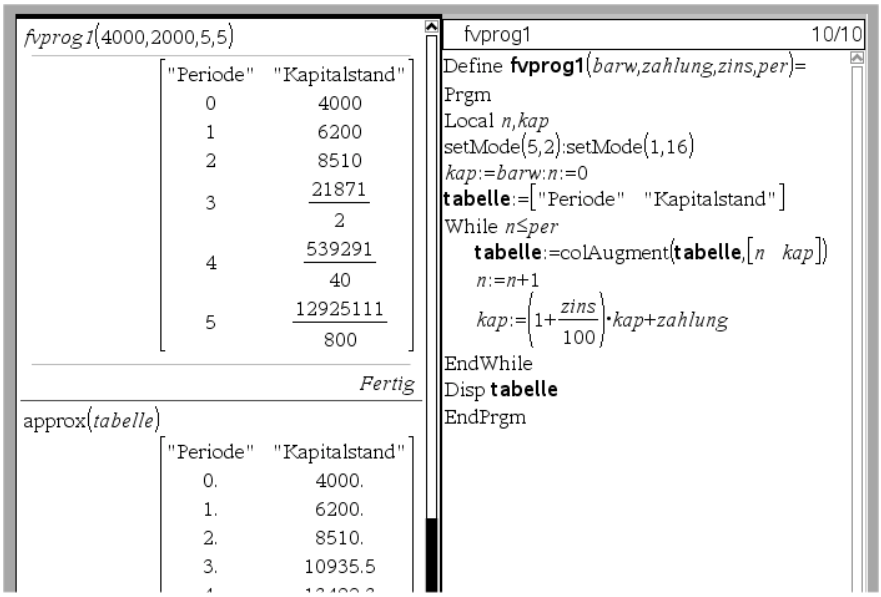
Finden Sie die Unterschiede, wobei ich hier nicht die andere Methode meine, die Schleife zur Berechnung der jeweils neuen Kapitalien und der entsprechenden Tabellenzeile zu definieren. Hier sehen Sie die Verwendung einer **Loop−EndLoop**-Schleife.

Da hier nur auf **Disp** zur Anzeige des Ergebnisses zurückgegriffen werden muss, wird tatsächlich die Ausgabe mit zwei Dezimalstellen durch die beiden **setMode**-Anweisungen erzwungen. Ich habe die generellen Dokumenteinstellungen eingeblendet.

Grundsätzlich könnten im Programm *n*, *kap,* und *tabelle* auch globale Variable werden, aber dann bleiben sie auch außerhalb des Programms bestehen. Das ist manchmal – vor allem bei der Fehlersuche – nützlich, aber sonst können diese Variablen nur bei anderen Problemen im gleichen Dokument stören. *tabelle* wäre möglicherweise auch als globale Variable sinnvoll, denn dann könnte man mit der *tabelle* weiter arbeiten. Wir werden dies gleich tun und demonstrieren, dass *tabelle* tatsächlich global ist. Im Calculator sehen Sie, dass mit *tabelle* das im Programm gespeicherte Ergebnis aufgerufen werden kann. Und damit können wir es auch in anderen Applikationen ansprechen. Zur Ausgabe über das Programm ist allerdings unbedingt die **Disp**-Anweisung nötig.

Weiters sollte Ihnen die Vollzugsmeldung *Fertig* auffallen, die bei Funktionen nicht auftritt, denn dort ist die Ausgabe ein Funktionswert.

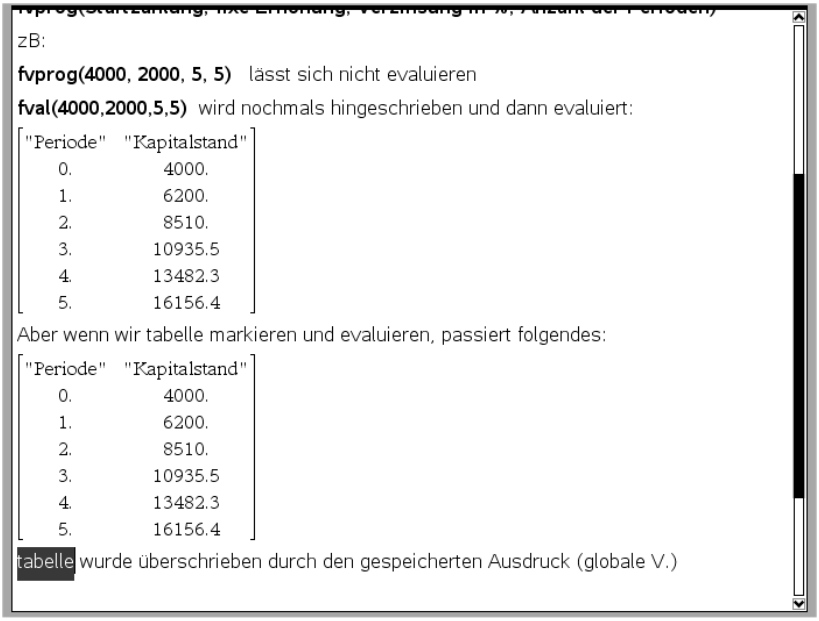
Ich ändere das Programm minimal, indem ich die Variable *tabelle* aus der Liste der lokalen Variablen entferne und das Programm nochmals laufen lasse. Außerdem möchte ich Ihnen eine dritte Möglichkeit zur Programmierung einer Schleife zeigen: die **While**−**EndWhile**-Konstruktion:



Auf der einen Seite müssen wir in diesem Fall wieder auf die – innerhalb des Programms definierte - Ausgabeform verzichten, aber andererseits steht uns der Wert der *tabelle* auch nach Durchführung des Programms zur weiteren Verfügung.

Schließlich sehen Sie die **tabelle** im Programm fett dargestellt. Das ist ein Hinweis darauf, dass es sich um eine bereits gespeicherte – weil globale – Variable handelt. Auch beim nochmaligen Öffnen des Codes der Funktion wird die **tabelle** fett auftreten.

Zum Schluss will ich noch die Portabilität der Ergebnisse am Beispiel der *Notes* demonstrieren.



Die Evaluation erfolgt, indem man die Funktion oder die Variable markiert und anschließend die Schaltfläche  im Menü der Notes anklickt. Mit einem Programm ist dies nicht möglich!

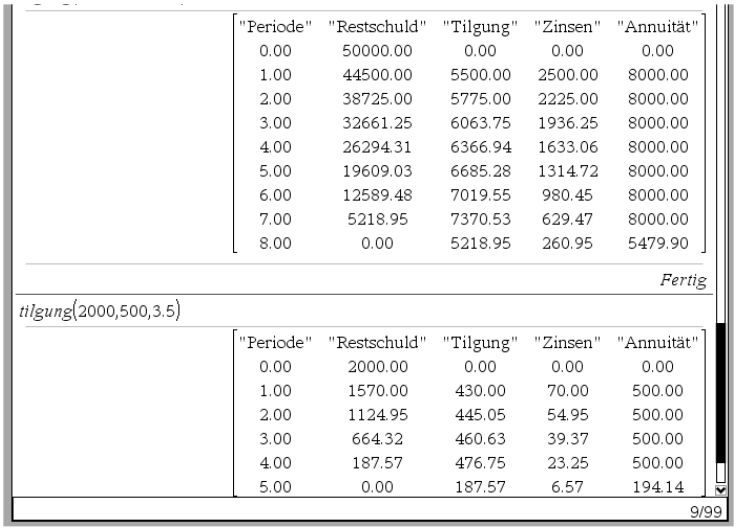
Übungsaufgaben:

Erstellen Sie eine Funktion zur Ausgabe einer Zinseszinstabelle.

Schreiben Sie ein Programm/eine Funktion, die den Verlauf einer Schuldtilgung in einer Tabelle darstellt.

Im ersten Aufruf wird eine Schuld von 50000 € durch Annuitäten von 8000 € getilgt, wobei Zinsen in der Höhe von 5% berechnet werden. Die Annuitäten werden erst zur Bedeckung der Zinsen und mit dem verbleibenden Rest zur Verminderung der jeweiligen Restschuld verwendet.

Die Ausgabe des Ergebnisses etwa so aussehen, wie auf der nächsten Seite gezeigt wird.



"Periode"	"Restschuld"	"Tilgung"	"Zinsen"	"Annuität"
0.00	50000.00	0.00	0.00	0.00
1.00	44500.00	5500.00	2500.00	8000.00
2.00	38725.00	5775.00	2225.00	8000.00
3.00	32661.25	6063.75	1936.25	8000.00
4.00	26294.31	6366.94	1633.06	8000.00
5.00	19609.03	6685.28	1314.72	8000.00
6.00	12589.48	7019.55	980.45	8000.00
7.00	5218.95	7370.53	629.47	8000.00
8.00	0.00	5218.95	260.95	5479.90

Fertig

tilgung(2000,500,3.5)

"Periode"	"Restschuld"	"Tilgung"	"Zinsen"	"Annuität"
0.00	2000.00	0.00	0.00	0.00
1.00	1570.00	430.00	70.00	500.00
2.00	1124.95	445.05	54.95	500.00
3.00	664.32	460.63	39.37	500.00
4.00	187.57	476.75	23.25	500.00
5.00	0.00	187.57	6.57	194.14

9/99

Bei Kenntnissen der Finanzmathematik können Sie auch einen Tilgungsplan entwerfen, der entsteht, wenn die Höhe der Schuld, die Verzinsung und die Laufzeit gegeben sind. In diesem Fall wäre zuerst die – gleich bleibende – Höhe der Annuität innerhalb des Programms zu berechnen.

Zur Abwechslung: Versuchen Sie bitte, die eine oder andere Form eines Tilgungsplans in der *Lists & Spreadsheet*-Applikation zu realisieren.

	A	B	C	D	E	F
◆						
1	Periode	Restschuld	Tilgung	Zinsen	Annuität	Zinsfuß
2	0.00	50000.00	0.00	0.00	0.00	5.00
3	1.00	44500.00	5500.00	2500.00	8000.00	
4	2.00	38725.00	5775.00	2225.00	8000.00	
5	3.00	32661.25	6063.75	1936.25	8000.00	
6	4.00	26294.31	6366.94	1633.06	8000.00	
7	5.00	19609.03	6685.28	1314.72	8000.00	
8	6.00	12589.48	7019.55	980.45	8000.00	
9	7.00	5218.95	7370.53	629.47	8000.00	
10	8.00		5218.95	260.95	5479.90	

2 Über eine Sutra zur ersten Bibliothek

Im schon genannten Büchlein *Programmieren in Derive* habe ich eine alte indische Weisheit aus der Mathematik der Veden für ein Einführungsprogramm verwendet. Diese Sutra lautet:

Durch Addition – Durch Subtraktion:

> Jede positive ganze Zahl > 2 lässt sich als Differenz zweier Quadratzahlen m^2 und n^2 darstellen. Wenn $x = a \cdot b$, dann ergeben sich m und n als $m = \frac{a+b}{2}$ *und* $n = \frac{a-b}{2}$.

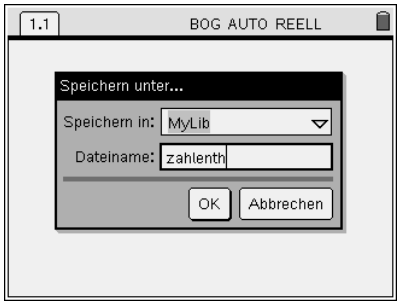
Die Aufgabe besteht darin, für jede Zahl x alle möglichen ganzzahligen Paare (m,n) in einer geeigneten Form auszugeben. Dazu benötigen wir vorerst alle Teiler von x. *Derive* bietet uns eine Funktion DIVISORS(n), die alle positiven Teiler in einer Liste ausgibt. Diese Funktion ist Teil einer Zusatzdatei, die eine Menge von zahlentheoretischen Funktionen enthält. Sie wurde in ihren wesentlichen Teilen von Johann Wiesenbauer zusammengestellt.

Wir wollen eine derartige Zusatzdatei (= Bibliothek) für den TI-*Nspire* anlegen. Wir betrachten die Funktion von Johann Wiesenbauer näher:

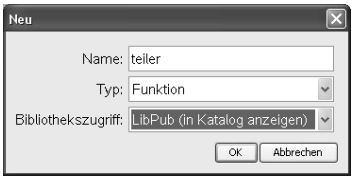
```
DIVISORS(n) := SORT(VECTOR(∏(u_), u_, {[1]}·∏(VECTOR(MAP_LIST([v_1^k_], k_,
{0, ..., v_2}), v_, FACTORS(n)))))
```

Das sieht reichlich kompliziert aus. Selbst, wenn es uns gelingt, diesen Einzeiler zu entschlüsseln, stehen wir vor dem nächsten Problem, denn auch die Funktionen `FACTORS` und `MAP_LIST` haben wir nicht zur Verfügung. Auf `FACTORS` wollen wir nachher noch zu sprechen kommen. Wir erzeugen uns Funktion teiler(n) für den Einsatz mit *Nspire* selbst (möglicherweise nicht so elegant und auch nicht effizient, aber die Unterschiede in den Rechenzeiten machen sich erst bei großen Zahlen deutlich bemerkbar).

Schritt 1: Wir öffnen ein neues *Nspire*-Dokument und speichern es sofort im Verzeichnis `Eigene Dateien\TI-Nspire\mylib` als zahlenth.tns. In diese Bibliothek legen wir die Funktion teiler(n) und alle zukünftigen Funktionen, die zu diesem Themenkreis gehören ab. Am PC geschieht dies in der üblichen Art und Weise und am Taschenrechner über das Werkzeugsymbol, wie rechts gezeigt wird.



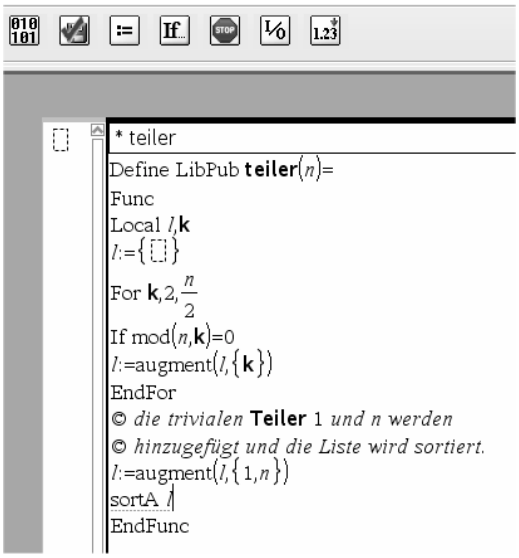
Schritt 2: Die Funktion wird als eine allgemein zugängige Funktion definiert und damit scheint sie nachher auch im Funktionskatalog auf.



Schritt 3: Das Programm – in diesem Fall die Funktion – wird editiert und kann auch schrittweise im Rechenfenster getestet werden. Man darf allerdings nicht vergessen, nach jeder Änderung über das grüne Häkchen die Letztfassung zu speichern.

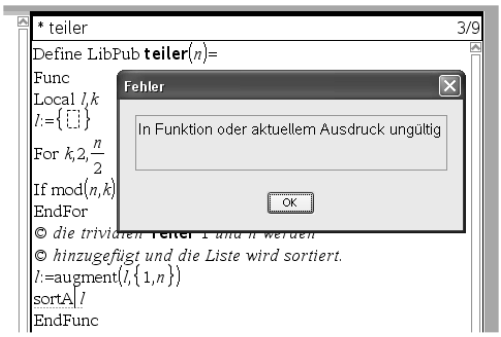
Diese Speicherung gilt aber nur im Rahmen der Sitzung. Es empfiehlt sich, in regelmäßigen Abständen die ganze Datei zu sichern. Damit ist man im Falle eines Programmabsturzes, der mit einem Rechnerabsturz verbunden sein kann, auf der sicheren Seite.

Das hochgestellte Sternchen vor dem Programm-(Funktions-)namen zeigt, dass die letzte Änderung noch nicht gespeichert wurde. Wir machen das nun und testen die Funktion teiler.



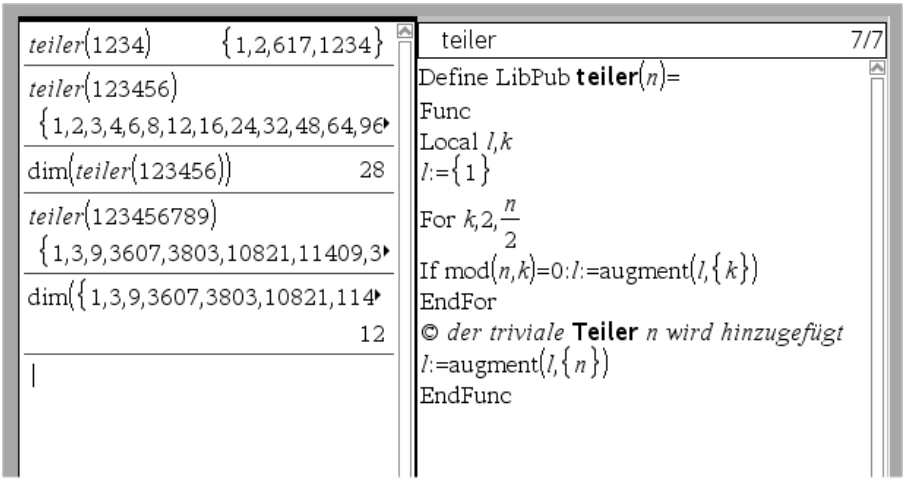
(Die Kommentarzeichen fügen Sie über die Editor-Schaltfläche ein.)

Schritt 4: Die Funktion muss fallweise verbessert werden. Hier zeigt die Syntaxprüfung einen Fehler und die Schreibmarke steht an der Stelle, wo er zu finden ist.



Die Sortierroutine kann offensichtlich in einer Funktion nicht eingesetzt werden. (Die sortierte Liste wird nicht zurückgegeben, sondern nur unter ihrem Namen gespeichert. Das passt nicht ins Konzept einer Funktion!)

Dann machen wir es eben anders:

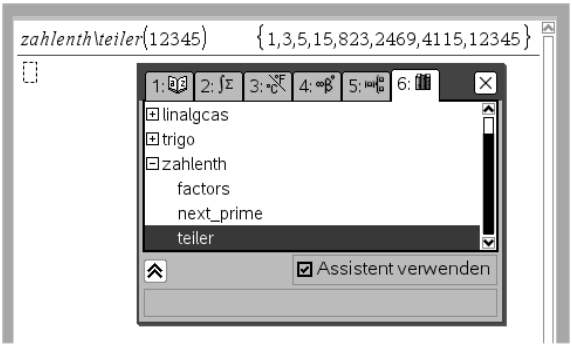


Es fällt auf, dass das Wort **teiler** – auch im Kommentar – fett gedruckt erscheint. Damit wird angezeigt, dass **teiler** eine gespeicherte Größe (Funktion, Programm, Konstante, Liste etc) ist.

Ein Vergleich mit dem Ergebnis von *Derive* macht uns sicher, dass unser Programm richtig ist – wenngleich nicht so elegant und rasch.

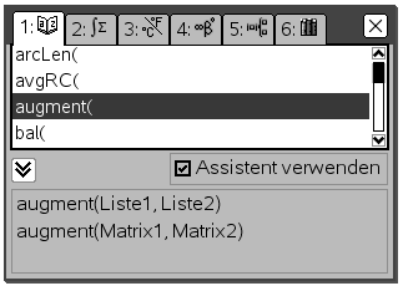
```
DIVISORS(123456789)

[1, 3, 9, 3607, 3803, 10821, 11409, 32463, 34227, 13717421, 41152263,
  123456789]

DIM(DIVISORS(123456789)) = 12
```

teiler kann nun in jedem Dokument eingesetzt werden. Da es als LibPub definiert wurde, wird es auch im Katalog angezeigt. Die Bibliothek zahlenth enthält – vorerst – nur die Funktion teiler. Der Aufruf erfolgt entweder über den Katalog oder editiert als zahlenth\teiler.



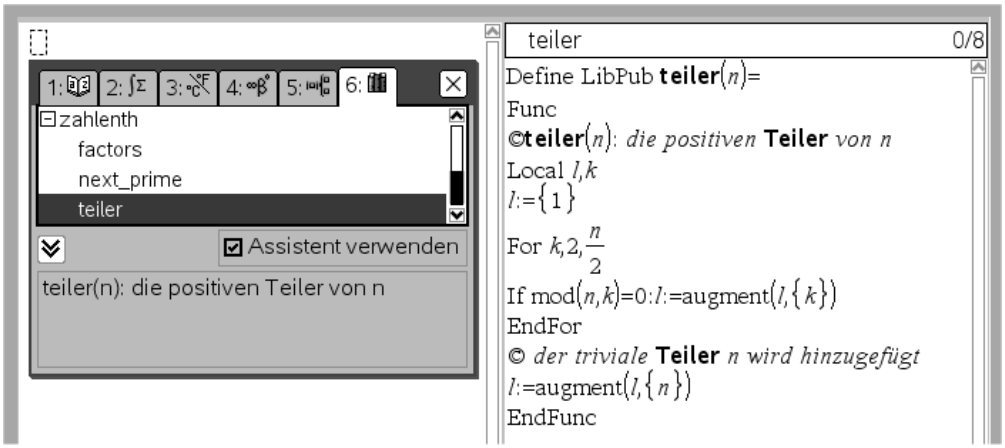
Insgesamt stehen drei Bibliotheken zur Verfügung. Die Bibliothek zahlenth enthält hier bereits drei Programme/Funktionen.

Es ist Ihnen sicherlich schon aufgefallen, dass beim Aufruf einer Funktion aus dem Katalog die Funktionen zumeist noch einen hilfreichen Kommentar zur richtigen Eingabe der Parameter anzeigen.



Ein derartiger Kommentar kann zu allen Bibliotheksprogrammen erstellt werden, und zwar ganz einfach:

Öffnen Sie das Programm nochmals und fügen Sie als erste Programmzeile jenen Kommentar ein, den Sie gerne als (Kurz-)Beschreibung sehen würden.



Speichern Sie die neue Fassung und frischen Sie die Bibliothek(en) wieder über einen Menüpunkt des -Menüs im Calculator. oder über einen Extrabutton in der Menüleiste auf.

Wenn Sie nun wieder auf die Bibliothek in Ihrem Katalog zurückgreifen, dann sollten Sie die Beschreibung vorfinden.

Da wir nun leicht zu den Teilern der Zahl *x*, die in die Differenz von ganzzahligen Quadraten zerlegt werden soll, gelangen, können wir uns dem eigentlichen Problem zuwenden.

Im nächsten Bildschirmabdruck sehen sie fast den kompletten Programmcode – der Schluss ist unten angefügt – mit einigen Testläufe und zusätzlichen Erläuterungen in den *Notes* (links unten).

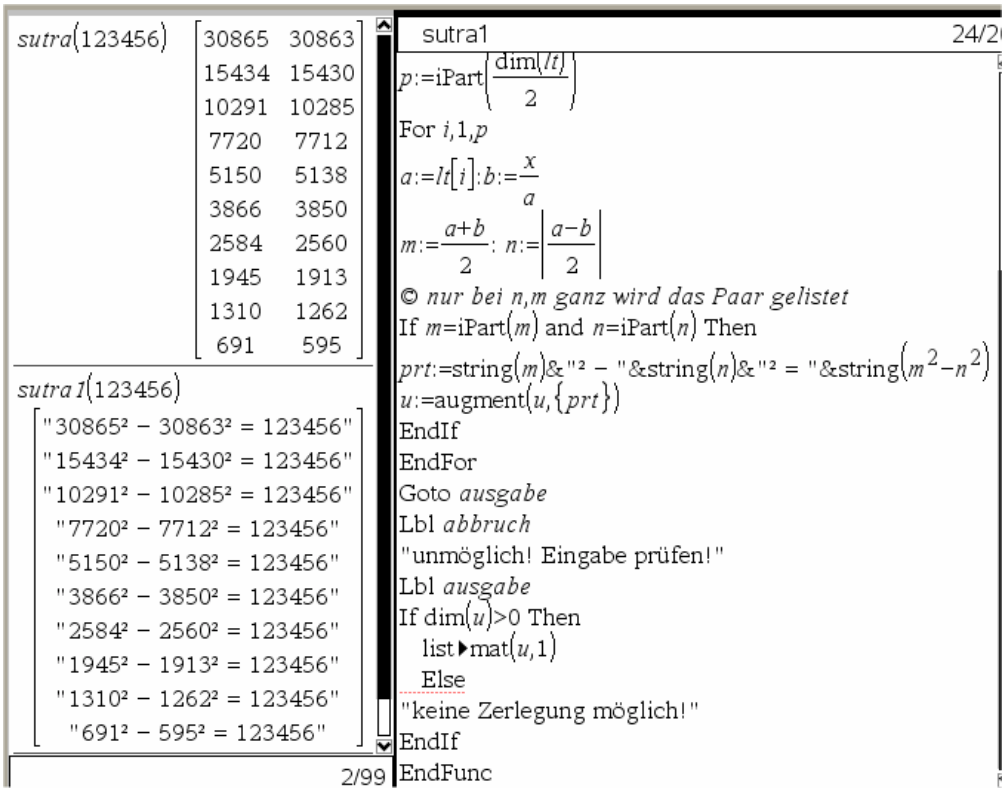
Das Problem ist gelöst – bei großen Zahlen können schon beträchtliche Rechenzeiten entstehen – `DIVISORS` in *Derive* beweist da schon seine Überlegenheit, aber teiler haben wir uns immerhin selbst „gestrickt".

> Wichtiger Hinweis: Überlange Rechnungen oder fehlerhafte Programme, die zu Endlosschleifen führen, können Sie abbrechen, indem Sie einige Sekunden auf die Pause-Taste drücken.

Programme sollen sich auch „verkaufen", d.h. über eine benützerfreundliche Eingabe verfügen und eine ordentliche Ausgabe der Resultate anbieten. Bei der Eingabe können wir (noch) nicht mehr machen, als den Programmaufruf mit den notwendigen Parametern ordentlich kommentieren (zB in beigefügten *Notes*) und Plausibilitätskontrollen einbauen. Leider ist derzeit noch keine Interaktion über einen Dialog zur Eingabe der Daten möglich. Doch bei

der Ausgabe haben wir doch einige Gestaltungsmöglichkeiten. So wäre es doch schön, wenn die Antwort für **sutra(1100)** etwa so aussehen würde.

```
sutra1(1100)
                    ["276² − 274² = 1100"
                     "60² − 50² = 1100"
                     "36² − 14² = 1100"]
```



Sie erkennen leicht den Unterschied: Die Quadratdifferenz wurde mit Hilfe von Zeichenkettenoperationen zusammen gefügt. **string(x)** wandelt die Zahl *x* in eine Zeichenkette um, die dann mit dem „&“ mit anderen Zeichenketten (unter “) verbunden wird. Das „^{2}”-Zeichen holen Sie sich aus dem Katalog aller vorrätigen *Nspire*-Symbole . Vergessen Sie nicht, *prt* auch unter die lokalen Variablen aufzunehmen. Wenn Sie die Fehlermeldung *In Funktion oder aktuellem Ausdruck ungültig* angezeigt bekommen, dann überprüfen Sie bitte zuerst, ob alle Variablen als „local” deklariert sind.

Wir sind in unserer Funktion teiler ohne explizite Primfaktorzerlegung ausgekommen. Ich will aber trotzdem - wie vorhin angekündigt - die Funktion `FACTORS(n)` behandeln, die in

Derive die Zerlegung in Primfaktoren in einer Form ausgibt, die diese Faktoren leicht weiter verwenden lassen:

$$\text{FACTOR}(720) = 2^4 \cdot 3^2 \cdot 5$$

$$\text{FACTORS}(720) = \begin{bmatrix} 2 & 4 \\ 3 & 2 \\ 5 & 1 \end{bmatrix}$$

Die `FACTORS`-Routine kann nicht eingesehen werden. So habe ich in *Derive* eine eigene geschrieben, die sicherlich nicht so elegant ist, wie die von Albert Rich oder Johann Wiesenbauer. Ich werde diese Funktion dazu benützen, um zu demonstrieren, wie man ein Programm aus einer anderen „Programmiersprache" in ein *TI-Nspire*-Programm umsetzen kann. Dieses Programm werden wir auch in unsere Bibliothek zahlenth aufnehmen.

Hier ist mein *Derive*-Programm `faktoren(n)` mit drei Testläufen:

```
faktoren(n, k, m, r) :=
  Prog
    m := []
    k := 2
    Loop
      If n = 1
         RETURN m
      r := 0
      Loop
        If MOD(n, k) = 0
           Prog
             n := n/k
             r :+ 1
        If MOD(n, k) ≠ 0 ∧ r ≠ 0
           m := APPEND(m, [[k, r]])
        If MOD(n, k) ≠ 0 exit
      k := NEXT_PRIME(k)
```

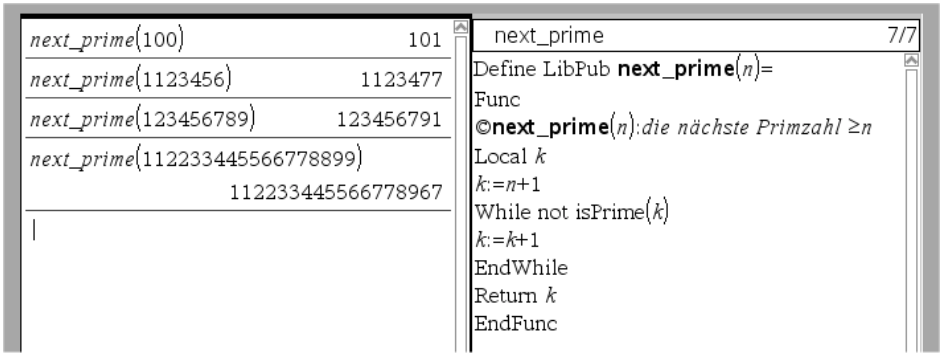
$$\text{faktoren}(720) = \begin{bmatrix} 2 & 4 \\ 3 & 2 \\ 5 & 1 \end{bmatrix}$$

$$\text{faktoren}(164110759) = \begin{bmatrix} 101 & 1 \\ 353 & 1 \\ 4603 & 1 \end{bmatrix}$$

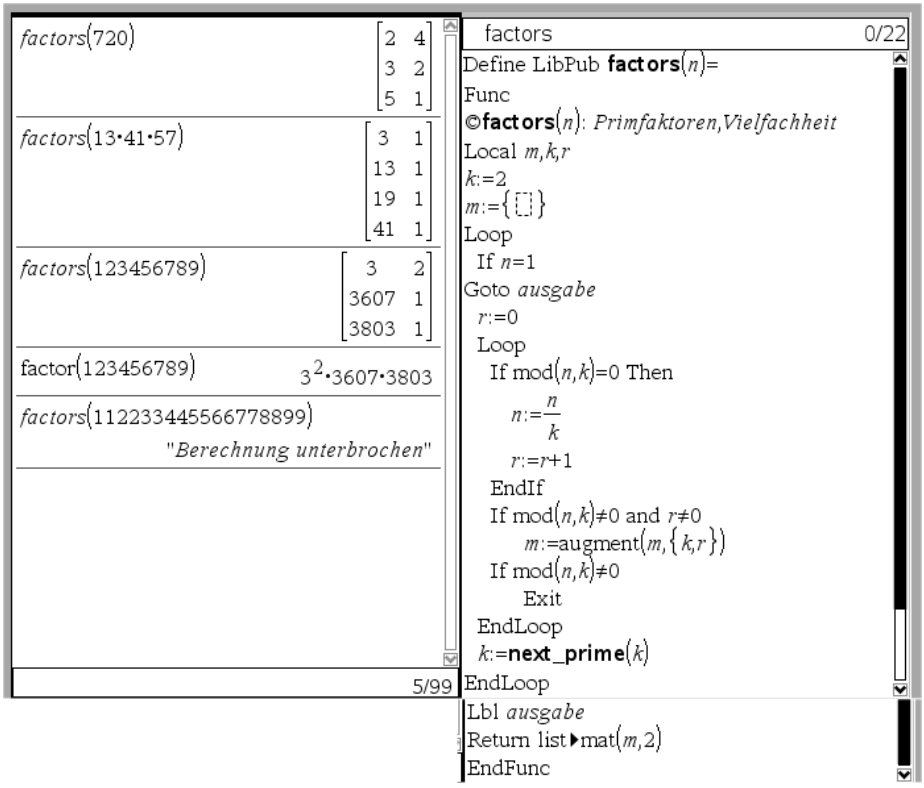
$$\text{faktoren}(78125) = [[5, 7]]$$

Die Erklärung ist nicht so schwierig: Wir dividieren die Zahl n der Reihe nach – mit $k = 2$ beginnend – durch die Primzahlen. Wenn sich die Division ganzzahlig ausgeht – wenn also MOD(n,k) = 0 – dann wird der Zähler r, der die Vielfachheit des Auftretens von k mitzählt, um eins erhöht und n wird durch k dividiert. Die allfällige mehrmalige Division durch den gleichen Primfaktor k erfolgt in der inneren *loop*-Schleife, der Schritt zur jeweils nächsten Primzahl in der äußeren. Die brauchbaren Paare [n, r] bei Auftreten eines Primfaktors in der Zerlegung werden aneinander gehängt (mit `APPEND`) und wenn n auf eins zusammengeschrumpft ist, wird die Liste in Form einer Matrix ausgegeben, wie aus den Testläufen ersichtlich ist.

Eine Schwierigkeit steht uns aber trotzdem noch ins Haus: `NEXT_PRIME(k)` liefert die auf k folgende nächste Primzahl. Auch diese Funktion ist im TI-*Nspire* nicht implementiert. Daher beginnen wir mit der Erzeugung der Funktion next_prime(n) für die Bibliothek:



Dabei können wir auf **isPrime(n)** zurückgreifen. Wir beginnen eine **While−EndWhile**-Schleife mit dem Wert $k = n+1$ und erhöhen k so lange, bis die Antwort auf **not isPrime(k)** „*falsch*“ ist. Dann wird die Schleife verlassen und k ausgegeben.

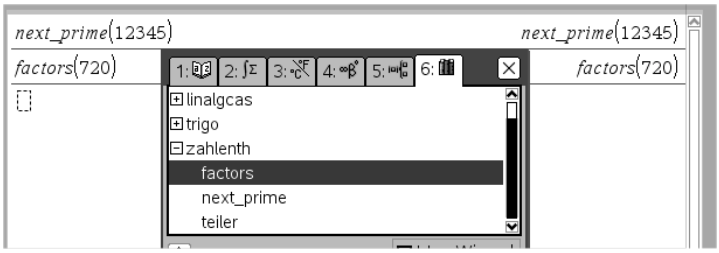


Die letzte Berechnung hat zu lange gedauert, daher habe ich sie durch einen längeren Druck auf die Pause-Taste abgebrochen.

Wenn Sie nun die beiden Programmcodes vergleichen, werden Sie feststellen können, dass sie im Kern übereinstimmen. In *Derive* werden die lokalen Variablen in die Parameterliste aufgenommen, während sie bei *Nspire* explizit als lokale Variable definiert werden müssen.

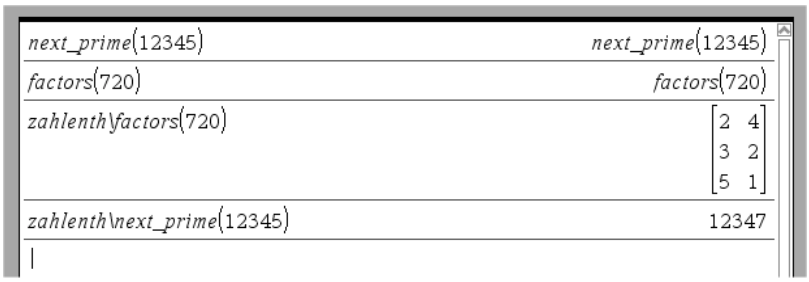
Sie finden auch beide Schleifen wieder. Die Ausgaberoutine habe ich der *Nspire*-Syntax angepasst. Die Paare werden zuerst in einer Liste gesammelt (zB {2,4,3,2,5,1} für $n = 720$) und dann in Form einer zweispaltigen Matrix ausgegeben.

Weil das nun wirklich wichtig ist, folgt nochmals der Hinweis zum Gebrauch der Bibliotheksprogramme. Ich habe nun ein neues Dokument geöffnet und möchte sowohl next_prime als auch factors verwenden. Das funktioniert vorerst nicht, auch wenn ich die Namen der beiden Funktionen im Gedächtnis habe. Wenn Sie aber zuerst den Katalog und dann in der Bibliothek den „Band“ zahlenth öffnen, können Sie beide Funktionen und auch teiler verwenden:



Sie können die Funktionen auch mit einem voran gesetzten zahlenth\ direkt aufrufen.

Ich wiederhole einen wichtigen Hinweis: Vergessen Sie bitte nicht, die Bibliotheken „aufzufrischen“. Das geschieht über einen Menüpunkt des -Menüs im Calculator oder über eine spezielle Schaltfläche in der Menüleiste.



Nun funktioniert alles so, wie wir es gerne haben wollten.

Aufgabe:

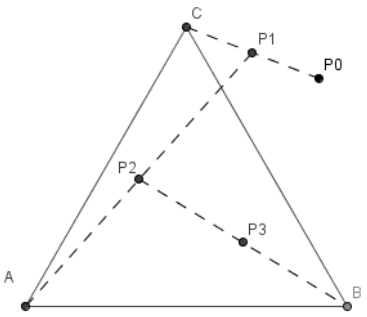
Mit Verwendung von factors(n) lässt sich die Funktion teiler(n) wesentlich effizienter programmieren. Versuchen Sie dies.

Erkunden Sie alle Menüs, die über die Schaltflächen des Editors geöffnet werden können.

3 Das Chaos - Spiel

Unser nächstes Programm wird eine ganze Menge neue Möglichkeiten ansprechen. Wir werden mit Zufallszahlen arbeiten, die erzeugten Daten im Spreadsheet unterbringen und schließlich auch graphisch darstellen. Alle angesprochenen Applikationen werden sich als untereinander verlinkt heraus stellen.

Das „Spiel“ verläuft folgendermaßen: In der Ebene wird ein gleichseitiges Dreieck gezeichnet. In der Ebene wird ein beliebiger Startpunkt P_0 festgelegt. Nun wird eine der drei Ecken des Dreiecks zufällig ausgewählt und diese mit P_0 verbunden. In meiner Skizze ist das der Punkt C. Der Mittelpunkt der Strecke – hier P_0C – ist der neue Punkt P_1. Es wird wiederum eine Dreiecksecke zufällig gewählt, wobei dies natürlich nochmals C sein könnte und der Mittelpunkt der entstandenen Strecke ergibt den Punkt P_2. Probieren Sie, etwa 20 Punkte auf diese Weise händisch zu erzeugen. Für die Zufallsauswahl reicht ein normaler Spielwürfel und die Augenzahlen 1 und 2 entsprechen dem Punkt A, 3 und 4 dem Punkt B und 5 bzw. 6 dem Punkt C. Die Punktmenge scheint sich chaotisch innerhalb der Dreiecksgrenzen auszubreiten!



Wenn wir sehen wollen, wie dieses „Chaos“ nach 100, 1000 oder noch mehr Schritten aussieht, werden wir wohl den Computer bemühen müssen.

Vorher werde ich die Eigenheiten des Zufallszahlengenerators besprechen, die für uns hier wichtig sind:

rand() erzeugt eine Zufallszahl zwischen 0 und 1,

rand(n) erzeugt *n* derartige Zufallszahlen,

randInt(u,o,n) erzeugt *n* ganze Zufallszahlen *z* mit $u \le z \le o$,

rand()	.943597
rand(3)	{.908319,.146688,.514702}
randInt(1,6,10)	{3,5,1,3,6,2,5,6,2,3}
randPoly(z,7)	$-4 \cdot z^7+9 \cdot z^6+7 \cdot z^5+z^4-9 \cdot z^3-7 \cdot z^2+8 \cdot z-9$
randBin(10,.7,10)	{7,6,8,7,7,8,6,7,9,5}
randNorm(2,.5)	1.54194
randNorm(2,.5,6)	{2.84882,2.54475,1.21173,1.66408,2.36781,1.01701}
randSamp({1,2,3},10)	{1,1,2,2,2,2,1,3,3,3}
randSamp({1,2,3,4,5,6,7,8,9,10},5,1)	{5,7,1,8,3}

randPoly(v,n) erzeugt ein Polynom vom Grad *n* in der Variablen *v* mit ganzzahligen Koeffizienten von -9 bis +9,

randBin(n,p,z) erzeugt *z* Ausfälle einer Binomialverteilung mit der Stichprobengröße *n* und dem Merkmalsanteil *p*. In der Abbildung auf Seite 20 wurden 10 Stichproben vom Umfang 10 aus einer Grundmenge gezogen, in der 70% Merkmalträger vorkommen,

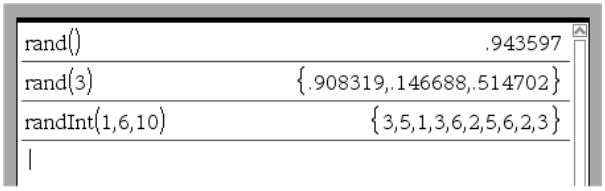
randNorm(m,s) erzeugt eine normalverteilte Zufallszahl mit dem Mittelwert m und der Standardabweichung *s*,

randNorm(m,s,n) erzeugt eine Liste von *n* derartigen Zufallszahlen,

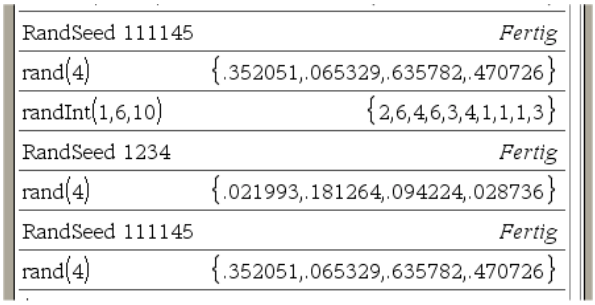
randSamp(Liste,n) erzeugt eine Zufallsstichprobe vom Umfang *n* aus der vorliegenden Liste, wobei „mit Zurücklegen" gearbeitet wird,

randSamp(Liste,n,1) erzeugt ebenfalls eine Stichprobe, wobei die gezogenen Elemente aber nicht wieder zurückgelegt werden.

Das sieht ja alles recht schön aus und funktioniert auch bestens. Wenn Sie aber Ihre nächste TI-*Nspire*-Sitzung eröffnen und wieder die Zufallszahlen aufrufen, oder wenn Sie in einer Schulklasse mit 25 Schülern das vorführen wollen, passiert folgendes:

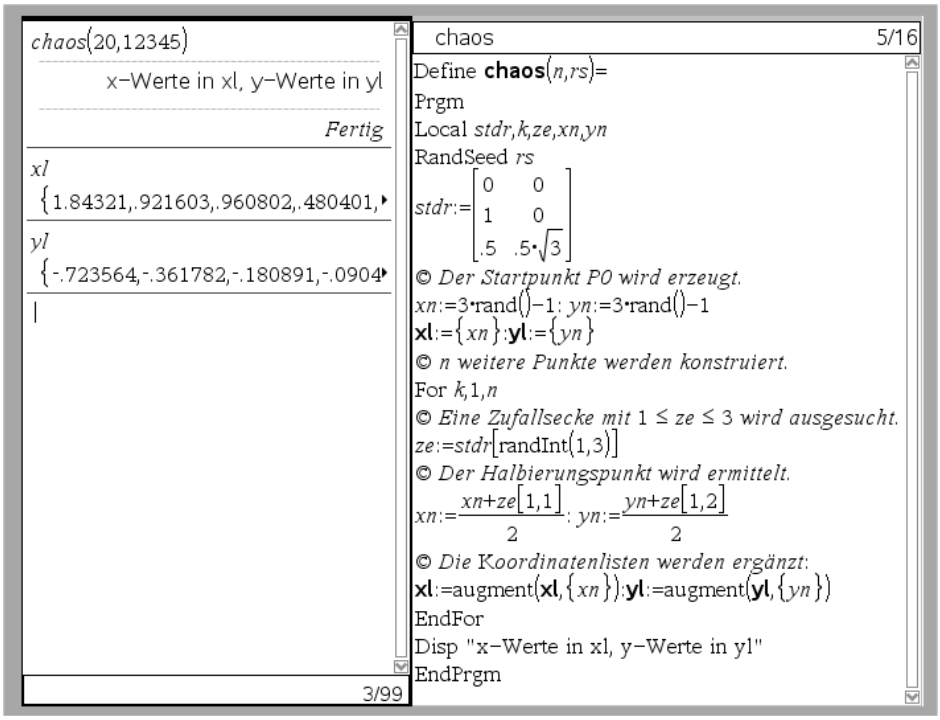


Die Zahlen erweisen sich als überhaupt nicht zufällig, sondern überall treten die gleichen Zahlen auf. Wir wissen natürlich, dass es sich um Pseudozufallszahlen handelt, die nach einem mehr oder weniger effektiven Algorithmus erzeugt werden, aber so kann es nicht gehen. Leider ist es beim TI-*Nspire* nicht – oder noch nicht – möglich, eine Größe, die mit dem Prozessor (zB mit der CPU-Zeit) verbunden ist, als Initialzündung für den Zufallsalgorithmus zu verwenden. Diese Initialzündung muss von außen kommen. Dazu verwenden wir hier die Anweisung **randSeed**:



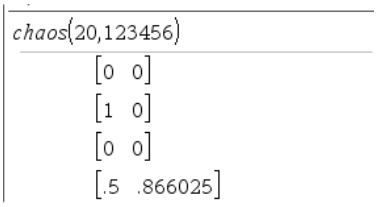
Das ist natürlich auch nicht ganz zufällig, da gleiche „Samen“ immer wieder gleiche Zufallszahlen als Früchte bringen. Aber damit lässt sich der Zufall „steuern“, was natürlich ein Widerspruch in sich ist. Da diese Startzahl nicht über einen Dialog abgefragt werden kann, wie es zB beim V200 oder TI-92 programmierbar war, muss sie als Parameter in den Programmaufruf eingebaut werden. Oder man beginnt die Sitzung mit **randseed Ganzzahl**.

Das Startdreieck *stdr* ist ein gleichseitiges Dreieck mit der Seitenlänge 1, einer Ecke im Koordinatenursprung und einer Seite auf der *x*-Achse.

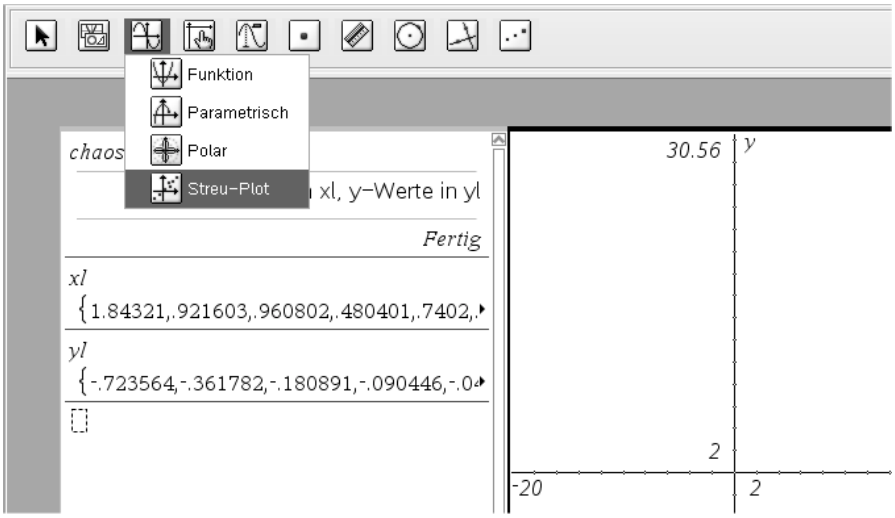


Ich arbeite hier mit einem echten Programm, weil die Listen, in denen die Koordinaten der „chaotischen“ Punkte gesammelt werden, extern in globalen Variablen *xl* und *yl* abgespeichert werden sollen. Alle anderen Variablen deklariere ich als lokal, um Datenmüll zu vermeiden. Im Rahmen des Programmierens – und vor allem bei der Fehlersuche – wird man die eine oder andere Variable global machen, um ihren Inhalt überprüfen zu können.

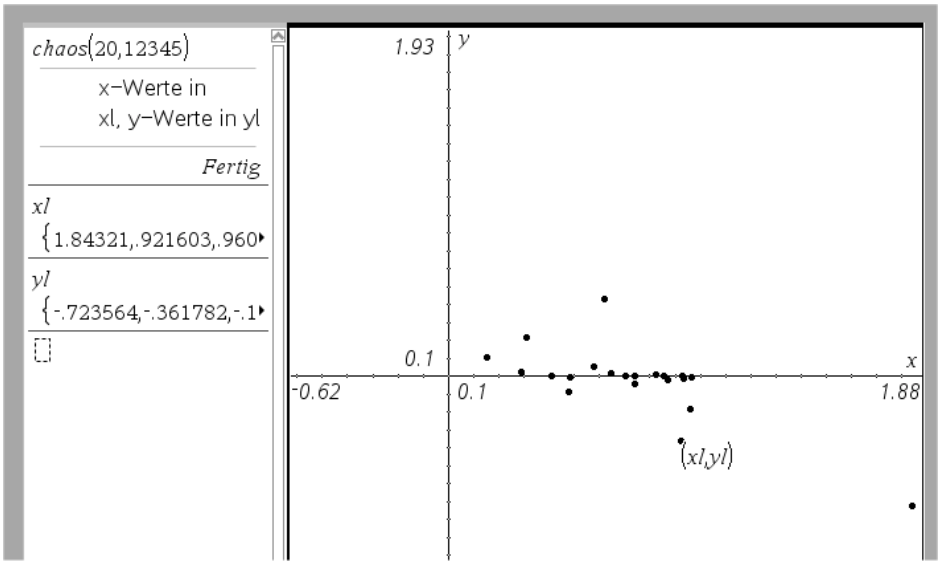
Sie können auch an geeigneten Stellen **Disp**-Anweisungen einbauen und dann im Calculatorfenster den Fortgang des Programms beobachten. Wenn ich zB nach der Zeile **ze:=stdr[randInt(1,3)]** die Anweisung **Disp ze** einbaue, dann erhalte ich im Calculatorfenster eine Ausgabe, die so beginnt, wie rechts gezeigt wird.



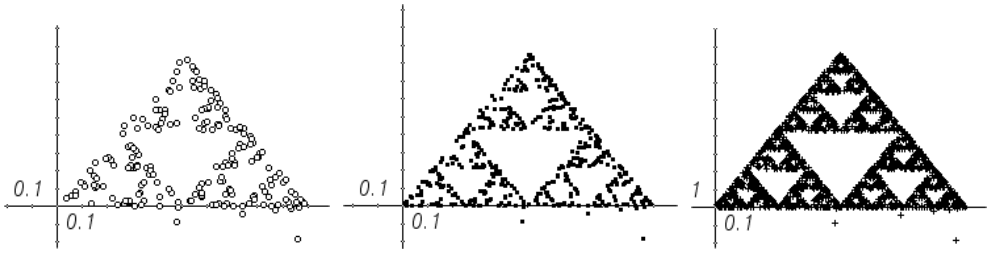
Nun wollen Sie ja sicherlich das Chaos auch sehen. Dazu teilen Sie das Fenster über das Seitenlayout-Menü vertikal und fügen Sie in die rechte Hälfte eine *Graphs & Geometry*-Seite ein, in der Sie gleich den Grafikmodus auf ein Streudiagramm (= Streu-Plot) ändern.



Dann brauchen Sie nur mehr unten in die Felder *x*← und *y*← die Variablen *xl* und *yl* einzutragen. Sie sehen einen Knäuel von Punkten, in den Sie sich „hineinzoomen“:

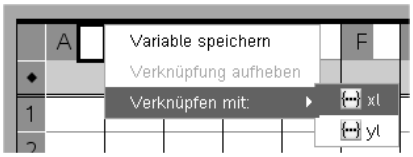


Rufen Sie nun der Reihe nach im Calculatorfenster das Programm mit steigenden Punktanzahlen auf : chaos(200,12345), chaos(400,12345), chaos(1000,12345) ein und beobachten Sie bitte, wie sich das „Chaos“ entwickelt. Über einen Rechtsklick auf einen Diagrammpunkt und die Option Attribute können Sie die Gestalt der Punkte verändern.



chaos(20,12345) chaos(500,12345) chaos(2000,12345)

Die Daten ließen sich auch in die *Lists & Spreadsheet* Applikation übertragen. Das soll nun zur Demonstration geschehen. Fügen Sie eine entsprechende Seite in Ihr Dokument ein und übernehmen Sie die Listen *xl* und *yl* in die Spalten A und B. Über die Var-Schaltfläche werden Sie eingeladen, die Spalten mit *xl* und dann in Spalte B mit *yl* zu verknüpfen. Nachdem dies erfolgt ist, sollte ihr Blatt so aussehen, wie unten rechts gezeigt:



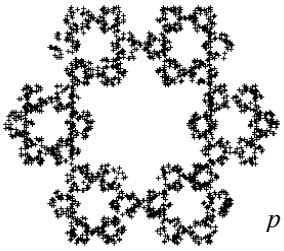
	A xl	B yl	C
◆			
1	1.84321	-.723564	
2	.921603	-.361782	
3	.960802	-.180891	
4	.480401	-.090446	
5	.7402	-.045223	
6	.8701	-.022611	

Aufgabe:

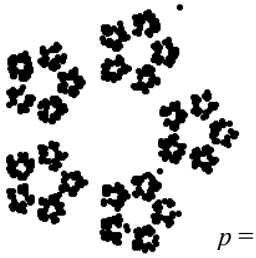
Ausgehend von einem regelmäßigen *p*-Eck mit $p \geq 3$ ergibt sich der jeweils nächste Punkt nach der Vorschrift:

$$\overrightarrow{P_{n+1}} = f \cdot \overrightarrow{P_n} + (1-f) \cdot \overrightarrow{Zufallsecke}$$

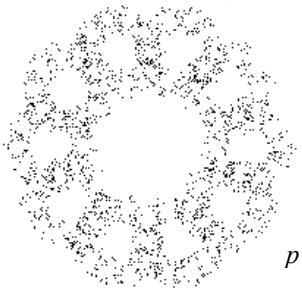
$$\text{mit } f = \frac{1}{2\left(1+\cos\frac{2\pi}{p}\right)}$$



$p = 6$



$p = 5$



$p = 10$

4 Eine Bibliothek für Dreiecke

In diesem Abschnitt wollen wir eine Bibliothek mit dem Namen trigo zusammenstellen, die zur Lösung von trigonometrischen Grundaufgaben, dh. vor allem zur Auflösung von allgemeinen Dreiecken herangezogen werden kann. Die dort bereit gestellten Programme sollen mehrere Kenntnisstufen der Schülerinnen und Schüler berücksichtigen.

In den *Notes* steht die Beschreibung der Programme, die zuerst natürlich geschaffen, aber dann angeboten werden. Es folgt eine Kopie der *Notes*:

Bibliothek TRIGO

Sinus- und Kosinussatz

cosw(s1,s2,s3) berechnet den Winkel, der der Seite s2 gegenüberliegt, wenn die drei Seiten s1, s2, und s3 gegeben sind.

coss(s1,w3,s2) berechnet die dritte Seite s3, wenn zwei Seiten s1 und s2 und der von ihnen eingeschlossene Winkel w3 gegeben sind.

sins(s1,w1,w2) berechnet die Seite s2, die dem Winkel w2 gegenüberliegt, wenn ein korrespondierendes Paar s1, w1 gegeben sind.

sinw(s1,s2,w1) berechnet den Winkel w2, der der Seite s2 gegenüberliegt, wenn das Paar s1 und gegenüberliegender Winkel w1 gegeben sind. Für den Fall, dass der Winkel der kleineren Seite gegenüberliegt, werden beide möglichen Winkel ausgegeben – falls sie existieren.

Auflösung von Dreiecken,

wenn drei Bestimmungsstücke vorliegen und der Fall nach SSS, SWS, WSW, SWW oder SSW angegeben wird. Auch der Fall WWW wird behandelt. Hier werden die Seitenlängen mit einem Proportionalitätsfaktor ausgegeben. In den Funktionen wird auf die Unterprogramme von oben zurückgegriffen.

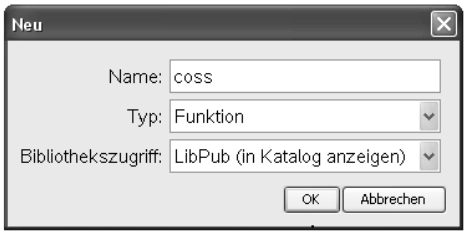
sss(s1,s2,s3)
sws(s1,w3,s2)
wsw(w1,s3,w2)
sww(s1,w1,w2)
ssw(s1,s2,w1)
www(w1,w2,w3)

liefern jeweils eine Matrix, die in der ersten Zeile die Seiten und in der zweiten Zeile die den Seiten gegenüberliegenden Winkel ausgibt.

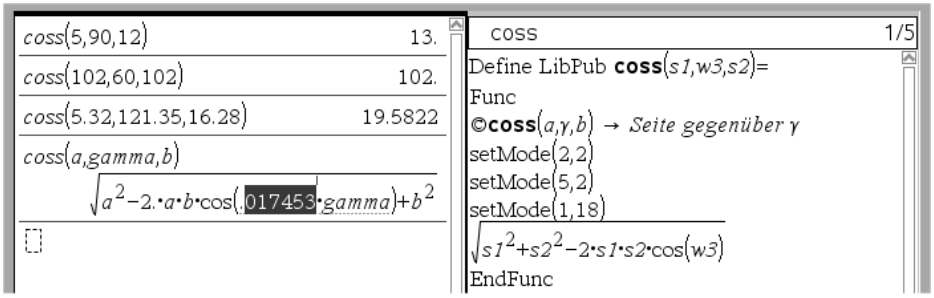
Das Programm **trigo(s1,s2,s3,w1,w2,w3)** liefert die Auflösung des Dreiecks, wenn drei der Parameter mit Zahlen belegt sind. In diesem Programm (eigentlich wieder Funktion) wird auf die "Unterprogramme" **sss** bis **www** zurückgegriffen.

Ich werde hier nicht alle Programme des Pakets trigo erklären, sondern einige davon Ihnen zum Programmieren überlassen. An Hand der vorgestellten Programme sollte Ihnen das nicht schwer fallen. Das komplette Paket wird Ihnen aber gerne zur Verfügung gestellt.

Öffnen Sie bitte ein neues Dokument und speichern Sie es sofort unter dem Namen `trigo` – oder unter einem anderen Namen, der Ihnen besser gefällt – im Verzeichnis `Eigene Dateien\TI-Nspire\mylib`, wie im Abschnitt 2 gezeigt worden ist. Für unsere Zwecke werden wir mit Funktionen auskommen, da keine Notwendigkeit besteht, globale Variable einzuführen. Dann öffnen Sie aus dem Calculator den Programmeditor zur Erstellung einer neuen Funktion und machen sie allgemein zugängig und auch im Katalog auffindbar.



Die erste Funktion *coss* ist rasch geschrieben und auch getestet. Sie soll jene Seite berechnen, die dem Winkel *w3*, der von den Seiten *s1* und *s2* eingeschlossen wird, gegenüber liegt – der Kosinussatz wird eingefordert.

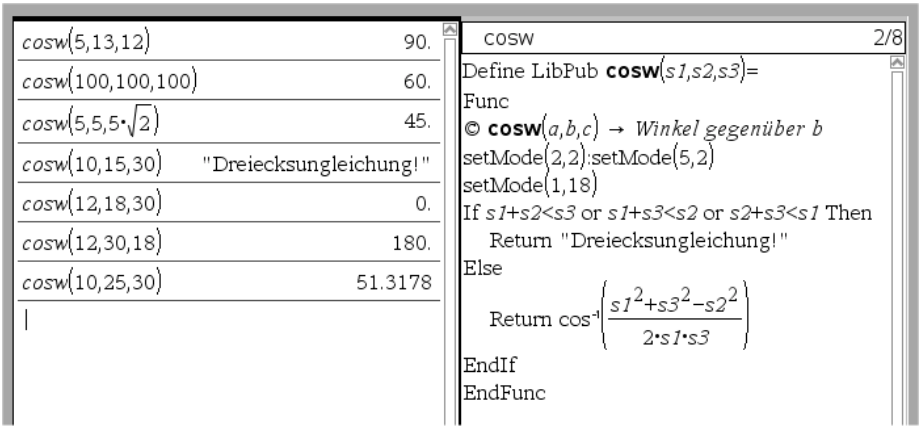


Über die drei **setMode**-Anweisungen, die wir über die letzte Schaltfläche -  - erhalten, werden jene Einstellungen festgelegt, die unabhängig von den getroffenen Dokumenteinstellungen für die Ausführung dieser einen Funktion gelten sollen: Rechnen im Gradmaß und eine approximierte Ausgabe mit 4 Dezimalstellen. Damit werden die Grundeinstellungen des Dokuments nicht verändert!

Frage an die SchülerInnen: Wie kommt die Zahl 0.017453 in der letzten Ausgabe zustande?

Die übrigen Schaltflächen dienen der raschen Eingabe der wichtigsten Programmierbefehle. Klicken Sie die Schaltflächen einfach an und sehen Sie sich die angebotenen Unterpunkte nochmals an.

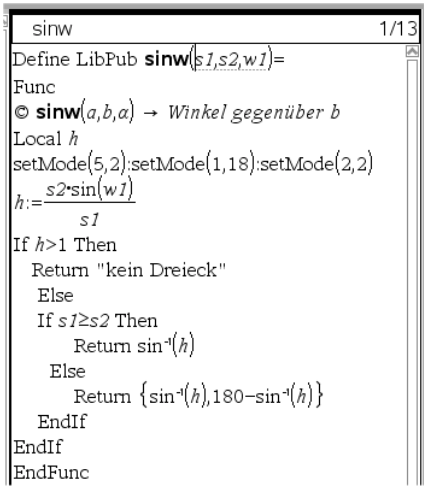
Bei der Erstellung von cosw(s1,s2,s3) werden wir eine Überprüfung der Sinnhaftigkeit der Eingabe einbauen, denn die Seiten müssen die Dreiecksungleichung erfüllen.



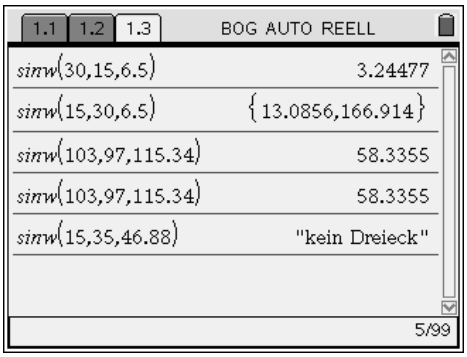
Beide **Return**-Anweisungen müssen nicht geschrieben werden, sie machen aber den Code leichter lesbar. (Frage: Wie sind die beiden Ergebnisse 0 und 180 zu interpretieren?)

Jetzt fehlt uns noch der Sinussatz in seinen beiden Formulierungen: Zwei Winkel und eine zugehörige Seite sind gegeben und die zweite Seite $s2$ ist gesucht: sins(s1,w1,w2) und die heiklere Sache mit zwei Seiten und einem gegenüberliegenden Winkel, denn hier kann es eine, zwei oder gar keine Lösung geben.

Ich überlasse Ihnen gerne das Programmieren von sins(s1,w1,w2) und beschränke mich auf sinw(s1,s2,w1):

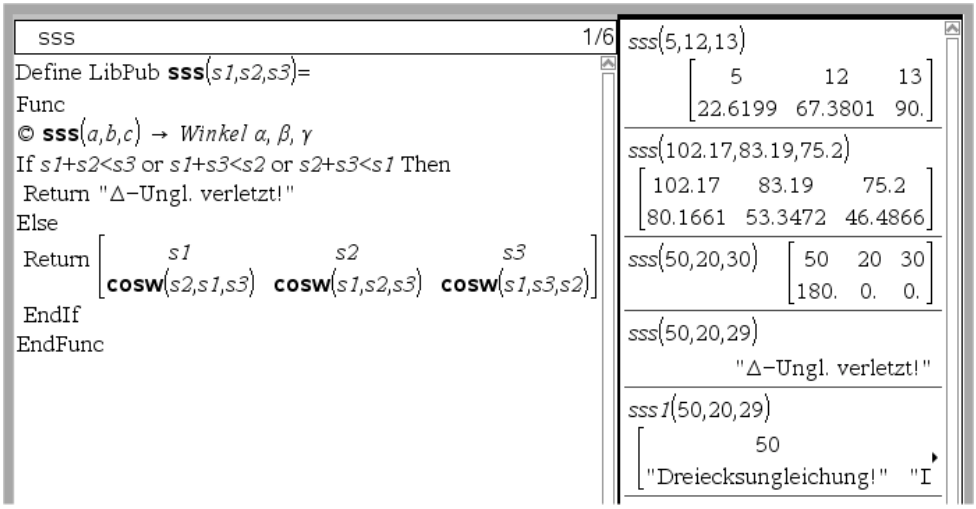


Beachten Sie bitte die geschachtelte **If−Else−EndIf**-Anweisung.



Nun haben wir die Grundformen von Sinus- und Kosinussatz als Funktionen definiert und können eine Stufe höher steigen.

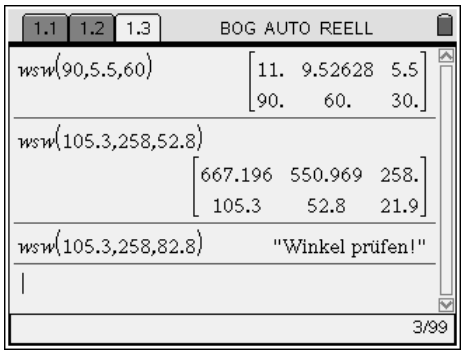
Der einfachste Fall liegt wohl vor, wenn alle drei Seiten des Dreiecks gegeben sind:



Die Abfrage im Zusammenhang mit der Dreiecksungleichung führen wir nochmals durch. Wie es aussehen würde, wenn wir das der „Unterfunktion“ cosw überlassen, sehen sie im Calculatorfenster unter sss1(50,20,29).

sws(s1,w2,s3) ist ganz einfach. Das kann ich getrost Ihnen überlassen. Sie müssen nur zuerst die Seite *s*2 mit coss berechnen - vergessen Sie nicht, *s*2 als lokale Variable zu definieren - und dann sind Sie ja schon wieder beim Fall SSS!

Der Fall wsw(w1,s3,w2) ist auch nicht schwierig. Nach Berechnung des dritten Winkels, können über zweimalige Anwendung von sins die restlichen beiden Seiten leicht gefunden werden. Die Ausgabematrix ist zu definieren. Eine Plausibilitätskontrolle ist vorzusehen. Betrachten Sie bitte die Ausgabe am Taschenrechner.



Auch der Fall sww(s1,w1,w2) lässt sich problemlos lösen, nachdem der fehlende Winkel *w*3 gefunden wurde, kann auf wsw(w3,s1,w2) zurückgegriffen werden.

Problematisch ist wiederum nur der Fall, wenn zwei Seiten und der einer Seite gegenüberliegende Winkel vorliegen, wobei die Parameter in einer eindeutigen Weise eingegeben werden müssen ssw(s1,s2,w1). Hier tritt wieder die Fallunterscheidung auf:

```
ssw                                                    13/13
Define LibPub ssw(s1,s2,w1)=
Func
© ssw(a,b,α = Winkel gegenüber a)
Local w3,k,ausg
setMode(2,2):setMode(5,2):setMode(1,18)
If s2·sin(w1)/s1>1: Return "keine Lösung!"
If s1≥s2 Then
   w3:=180−w1−sinw(s1,s2,w1)
   sws(s1,w3,s2)
EndIf
If s1<s2 Then
   w3:=180−w1−sinw(s1,s2,w1)
   ausg:=colAugment(sws(s1,w3[1],s2),["---"  "2. Lösung"  "---"])
   Return colAugment(ausg,sws(s1,w3[2],s2))
EndIf
EndFunc
```

Und hier sind einige Testdreiecke:

ssw(35,15,46.5)	$\begin{bmatrix} 35. & 43.5911 & 15. \\ 46.5 & 115.388 & 18.112 \end{bmatrix}$
ssw(35,15,133.5)	$\begin{bmatrix} 35. & 22.9405 & 15. \\ 133.5 & 28.388 & 18.112 \end{bmatrix}$
ssw(15,35,56.3)	"keine Lösung!"
ssw(15,35,10.3)	$\begin{bmatrix} 15. & 48.0682 & 35. \\ 10.3 & 145.042 & 24.6583 \\ "---" & "2. Lösung" & "---" \\ 15. & 20.8038 & 35. \\ 10.3 & 14.3583 & 155.342 \end{bmatrix}$

Vergessen Sie bitte nicht, immer wieder zu speichern.

Jetzt ist es schon längst an der Zeit, die Funktionalität der Bibliothek zu überprüfen.

Öffnen Sie bitte ein neues Dokument und rufen Sie den Calculator auf.

Wir wollen das Dreieck ABC mit *a* = 105,33, b = 82,17 und α = 110,13° nach seinen restlichen Bestimmungsstücken auflösen. Wir erkennen, dass der Fall SSW angesprochen ist.

Aktualisieren Sie zuerst die Bibliotheken und öffnen Sie anschließend den Katalog.

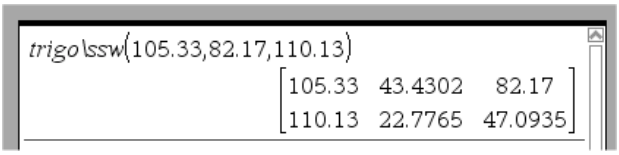
Unter Option 6 wird Ihnen Ihre eigene Bibliothek trigo angeboten. Wenn Sie auf das +-Zeichen klicken, sehen Sie alle darin enthaltenen Programm und Funktionen.

Doppelklicken Sie auf ssw.

Im Calculatorfenster sehen Sie den Funktionsaufruf vorbereitet und Sie können die erforderlichen Parameter eintragen.

Wir erwarten eine Lösung.

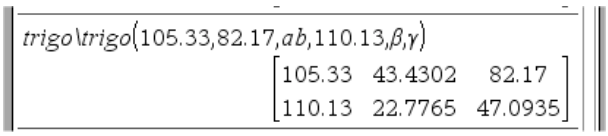
Es liegt am Benutzer, die Lösung richtig zu interpretieren, wobei das hier nicht schwierig sein sollte.



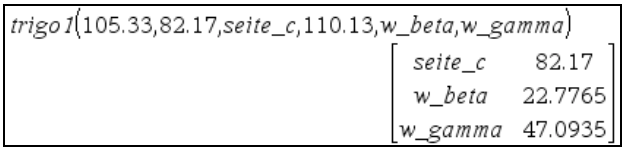
In der ersten Zeile stehen die drei Seiten und darunter die jeweils gegenüber liegenden Winkel. In der Standardbezeichnung ist demnach $c = 43{,}43$, $\beta = 47{,}09°$ und $\gamma = 22{,}78°$. Wir erkennen, dass die Funktion, wie auch alle anderen bisher definierten, von überall her recht bequem aufgerufen werden können.

Jetzt hätten wir aber noch einen Fall vergessen: was tun, wenn drei Winkel vorliegen. Ich weiß, dass dies meist übergangen wird, aber es ist doch ein interessanter Fall. Hier gibt es unendlich viele Lösungsdreiecke, die alle zueinander ähnlich sind. Das werden wir auch am Ende des Kapitels noch näher behandeln.

So weit, so gut. Aber das Bessere ist des Guten Feind. Es wäre doch noch einfacher, wenn wir die Auswahl der richtigen Funktion auch dem Computer überlassen könnten! Dann könnte die Durchführung so aussehen:



Oder vielleicht noch freundlicher? Was sagen Sie dazu:



Hier will ich nicht die ganze Prozedur nachvollziehen. Es geht natürlich zuerst einmal darum, aus den vorliegenden Daten herauszufiltern, welcher der Kongruenzsätze anzuwenden ist. Dann sind fallweise die Seiten oder Winkel zu vertauschen, dass sie auch richtig verarbeitet werden.

Für die Klassifikation habe ich mir ein Schema zu Recht gelegt, das jedem möglichen Angabefall eine eindeutige Zahl zuordnet.

Nach den Regeln der Kombinatorik kann man auf 20 verschiedene Arten 3 Elemente – ohne Wiederholung – aus 6 auswählen, weil $\binom{6}{3} = 20$.

	A	B a	C b	D c	E α	F β	G γ	H
1	www	0	0	0	1	1	1	7
2	sww	0	0	1	0	1	1	11
3	sww	0	0	1	1	0	1	13
4	wsw	0	0	1	1	1	0	14
5	sww	0	1	0	0	1	1	19
6	wsw	0	1	0	1	0	1	21
7	sww	0	1	0	1	1	0	22
8	ssw	0	1	1	0	0	1	25
9	ssw	0	1	1	0	1	0	26
10	sws	0	1	1	1	0	0	28
11	wsw	1	0	0	0	1	1	35
12	sww	1	0	0	1	0	1	37
13	sww	1	0	0	1	1	0	38
14	ssw	1	0	1	0	0	1	41
15	sws	1	0	1	0	1	0	42
16	ssw	1	0	1	1	0	0	44
17	sws	1	1	0	0	0	1	49
18	ssw	1	1	0	0	1	0	50
19	ssw	1	1	0	1	0	0	52
20	sss	1	1	1	0	0	0	56
21								

Diese Tabelle habe ich natürlich auch gleich mit *Nspire* angelegt. Fügen Sie eine Seite *Lists & Spreadsheet* ein, tragen Sie die Überschriften und Daten in die Spalten A bis G ein. In die Zelle *H1* schreiben Sie bitte die Formel `=32*b1+16*c1+8*d1+4*e1+2*f1+g1` und kopieren diese – wie in Excel in die restlichen Zellen der Spalte H.

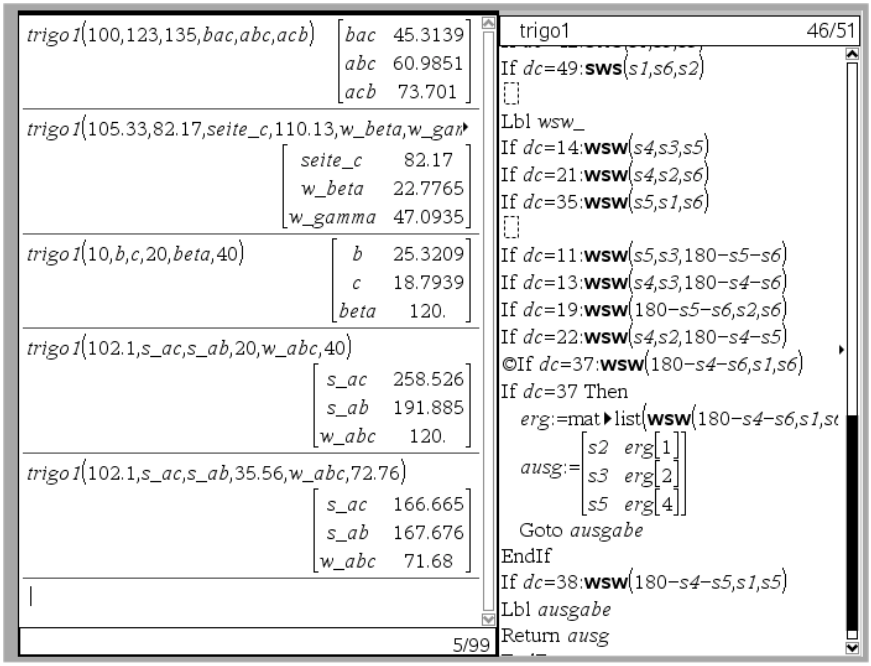
Leider finde ich kein deutliches Muster, das ich für eine effiziente Programmierung ausnützen könnte.

Daher mache ich es auf die „einfachste“ Art und Weise, wie dann der nächste Programmausschnitt zeigt.

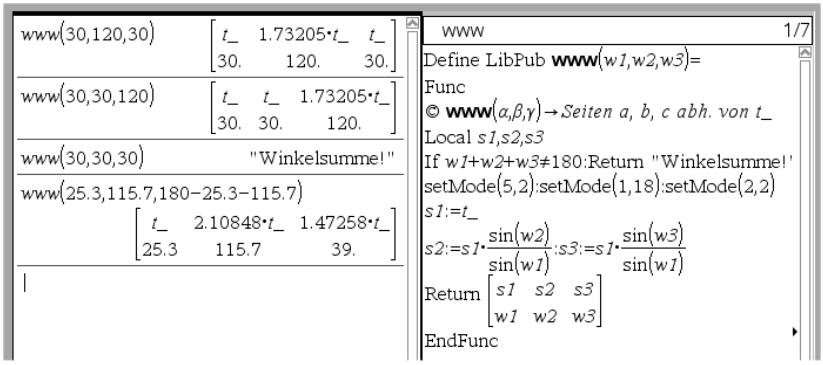
In der Zeile, die mit **v[1,i]** beginnt, werden den Elementen des Vektors *v* die Werte 1 oder 0 zugewiesen, je nachdem, ob der Eingabewert mit einer Ziffer oder dem Dezimalpunkt beginnt (ASCII-Code <=57) oder nicht. Damit erhalte ich die Zeilen der obigen Tabelle. Wenn die Summe der Elemente nicht mit 3 übereinstimmt, liegt ein Eingabefehler vor. Dieser Vektor wird skalar mit dem Vektor [32, 16, 8, 4, 2, 1] multipliziert und führt so zum Code jedes einzelnen Falles. Dann wird nach jedem Code einzeln verfahren, wobei einige zuerst gruppenweise zusammengefasst und dann in den durch Sprungmarken (= Labels) mit **lbl** gekennzeichneten Programmteilen einzeln behandelt werden. Ich glaube, dass Sie das leicht nachvollziehen können. Es ist sicherlich eine interessante Aufgabe für Schülerinnen und Schüler, diese Fälle zu analysieren.

```
trigo                                                    20/2
Define LibPub trigo(s1,s2,s3,s4,s5,s6)=
Func
© trigo(a,b,c,α,β,γ) – 3 Größen gegeben
Local v,i,dc:v:=[0  0  0  0  0  0]
For i,1,6
v[1,i]:=when(ord(left(string(#("s"&string(i))))))≤57,1,0)
EndFor
If sum(mat▸list(v))≠3:Return "Eingabefehler!"
dc:=dotP(v,[32  16  8  4  2  1])
If dc=56:sss(s1,s2,s3):If dc=7:www(s4,s5,s6)
If dc=28 or dc=42 or dc=49:Goto sws_:If dc=14 or dc=21 or dc=35:Goto wsw_
If dc=11 or dc=13 or dc=19 or dc=22 or dc=37 or dc=38:Goto wsw_
© Fall SSW
If dc=52:ssw(s1,s2,s4):If dc=44:ssw(s1,s3,s4):If dc=26:ssw(s2,s3,s5)
If dc=50:ssw(s2,s1,s5):If dc=41:ssw(s3,s1,s6):If dc=25:ssw(s3,s2,s6)
Lbl sws_
If dc=28:sws(s2,s4,s3):If dc=42:sws(s1,s5,s3):If dc=49:sws(s1,s6,s2)
Lbl wsw_
If dc=14:wsw(s4,s3,s5):If dc=21:wsw(s4,s2,s6):If dc=35:wsw(s5,s1,s6)
If dc=11:wsw(s5,s3,180–s5–s6):If dc=13:wsw(s4,s3,180–s4–s6)
If dc=19:wsw(180–s5–s6,s2,s6):If dc=22:wsw(s4,s2,180–s4–s5)
If dc=37:wsw(180–s4–s6,s1,s6):If dc=38:wsw(180–s4–s5,s1,s5)
EndFunc
```

Im nächsten Bild zeige ich eine Möglichkeit, die Ausgabe noch angenehmer für den Anwender zu gestalten. Für jeden Fall wird die spezifische Ausgabematrix gestaltet und nach dem Label *ausgabe* im Calculatorfenster ausgegeben.



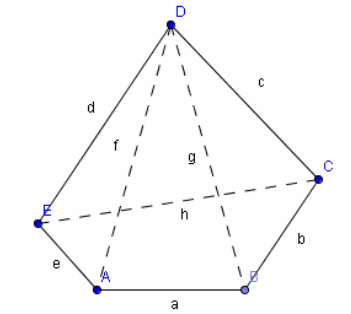
Jetzt bin ich Ihnen noch den Fall WWW schuldig! Ich führe den Proportionalitätsfaktor *t_* ein und gebe die Seiten mit diesem Faktor behaftet aus.



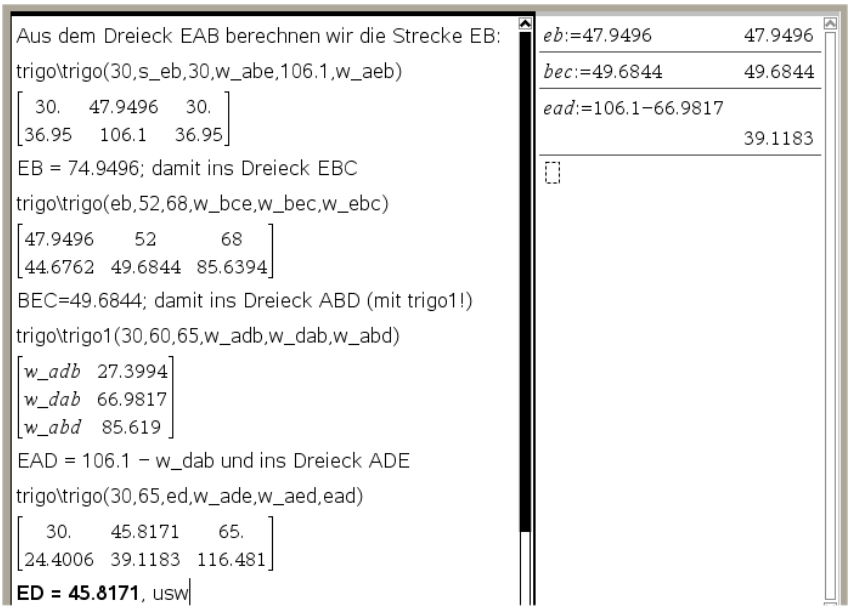
Zum Abschluss lösen wir mit dem Werkzeug eine komplexere „traditionelle Vermessungsaufgabe“:

Von einem Fünfeck ABCDE kennt man:
AB = 30, BC = 52, AE = 30, AD = 65,
BD = 60, EC = 68
und den Winkel EAB = 106,1°.

Berechne ED und CD.



Wir bearbeiten die Aufgabe gleich in den „frischen“ *Notes* und evaluieren darin die Funktionen. Auch die Zuweisungen **eb:=** und **bec:=** könnten in den Notes erfolgen.



5 Wir haben ein CAS: ein Programm für Extremwertaufgaben

Die Extremwertaufgaben sind wohl ein Dauerbrenner unter den Anwendungen der Differentialrechnung. In meiner Lehrerlaufbahn habe ich immer Fragen gefürchtet wie diese: „Wo brauche ich im wirklichen Leben die quadratische Pyramide mit dem größten Volumen, die man einer Kugel einschreiben kann?“ Hand aufs Herz, wissen Sie eine ehrliche Antwort auf diese Frage?

Wenn man mit einem CAS arbeitet, reduziert sich bald der mathematische Gehalt dieser Aufgaben auf das Auffinden und Formulieren von Haupt- und Nebenbedingung(en) und auf das Interpretieren der Ergebnisse. Alle übrige Arbeit (Reduktion der Anzahl der Variablen in der Hauptbedingung, Nullsetzen der ersten Ableitung und Lösen der entstehenden Gleichung usw.) wird dann rezeptartig vom CAS übernommen.

Ich finde, dass es eine sehr reizvolle Aufgabe ist, dieses „Rezept“ in die Form eines lauffähigen Programms zu gießen. Damit könnten auch die Extremwertaufgaben eine ganz neue Qualität erhalten. Erst damit wird man ja gezwungen, jeden Schritt genau zu analysieren und auch allfällige Sonderfälle in seine Überlegungen mit einzubeziehen.

Betrachten wir eine Standardaufgabe, die aus der Hauptbedingung (mit zwei Variablen) und einer Nebenbedingung besteht:

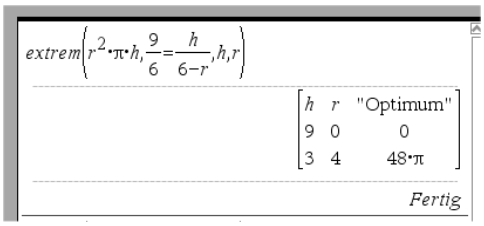
> Einem geraden Kreiskegel mit $r = 6$ und $h = 9$ ist der volumsgrößte Kreiszylinder einzuschreiben. Die Achsen der beiden Körper fallen zusammen.

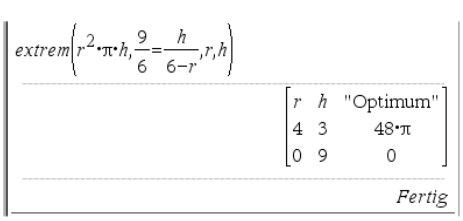
$V(r,h) = r^2\,\pi\,h = \text{Maximum}$ Hauptbedingung

$\frac{9}{6} = \frac{h}{6-r}$ Nebenbedingung

Das Rezept besteht darin, aus der Nebenbedingung eine der beiden Variablen herauszurechnen und dafür in die Hauptbedingung einzusetzen, die entstandene Funktion nach der verbleibenden Variablen zu differenzieren und die Nullstellen der ersten Ableitung zu finden. Damit sind die lokalen Extremwerte gefunden. Die restliche Aufgabe ist, aus der Nebenbedingung den Wert der anderen Variablen zu berechnen und schließlich den Maximal- oder Minimalwert der Hauptbedingung anzugeben. Abschließend wird über die zweite Ableitung die Art des lokalen Extremwerts bestimmt.

Ich versuche nun, meinem Programm extrem die Lösung des Problems zu entlocken:





Als Parameter sind zuerst die Haupt- und dann die Nebenbedingung einzugeben, gefolgt von den beiden Variablen, wobei die Reihenfolge bestimmt, welche Variable aus der Nebenbedingung herausgerechnet wird, dh. nach der zweiten Variablen wird schlussendlich differenziert.

Wie im „wirklichen Leben" kann diese Reihenfolge für die bequeme Lösbarkeit entscheidend sein. Die Bestimmung der Art der Extremwerte ist in die Minimalfassung des Programms noch nicht aufgenommen.

Ich habe das Programm so kompakt geschrieben, dass es sich auf einer Bildschirmseite vollständig darstellen lässt.

```
extrem                                                    14/17
Define extrem(hb,nb,w1,w2)=
Prgm
© extrem(Hauptbed., Nebenbed., 1. Var., 2. Var.)
Local i,nz,v1,v2,sols1,solsv1,nsols1,auxv1,hb1,sols2,ausg
© Überschrift der Ausgabe, Lösungen vorerst in Spalten
ausg:=[ w1
        w2
        "Optimum" ]
v1:=w1: v2:=w2
© die NB wird nach w1 (=v1) aufgelöst und die Lösungen in sols1 gesammelt
sols1:=zeros(left(nb)−right(nb),v1):nsols1:=dim(sols1)
For i,1,nsols1
auxv1:=sols1[i]:hb1:=hb|v1=auxv1
© die HB heißt nun hb1, nach v2 differenziert u. nach v2 aufgelöst
sols2:=cZeros(d/dv2(hb1),v2)
© die Werte für die erste Variable werden bestimmt
solsv1:=auxv1|v2=sols2
hb1:=hb1|v2=sols2:nz:=dim(hb1): hb1:=augment(augment(solsv1,sols2),hb1)
hb1:=list▸mat(hb1,nz):ausg:=augment(ausg,hb1)
EndFor
Disp ausg^T
EndPrgm
```

Die Überschrift können Sie entweder über eine Eingabemaske eingeben, oder Sie schreiben **ausg:=[w1;w2;"Optimum"]**. Zum Schluss transponiere ich die entstehende Ausgabetabelle und erhalte das oben gezeigte Ergebnis.

Aufgabe

Schreiben Sie das Programm extrem um in eine Funktion**.**

Da wir mit einem CAS arbeiten, sollte es auch möglich sein, mit allgemeinen Werten für Höhe und Radius des gegebenen Kegels zu arbeiten. So nehmen wir nun anstelle von 9 und 6 die Variablen *hk* und *rk*. (Außerdem arbeite ich mit der Funktion!)

$$extremf\left(r^2\cdot\pi\cdot h,\frac{hk}{rk}=\frac{h}{rk-r},h,r\right)$$

h	r	"Optimum"
$\frac{hk}{3}$	$\frac{2\cdot rk}{3}$	$\frac{4\cdot hk\cdot rk^2\cdot\pi}{27}$
hk	0	0

Hurra, das scheint ja wirklich zu gehen!

Aufgabe: Ergänzen Sie das Programm oder die Funktion um die Ausgabe der Art des Extremwerts, wie das folgende Bild zeigt:

$$extremext\left(r^2\cdot\pi\cdot h,\frac{9}{6}=\frac{h}{6-r},h,r\right)$$

h	r	"Optimum"	"Art des Extremwts"
9	0	0	"lok. Min."
3	4	$48\cdot\pi$	"lok. Max."

Es folgen nun einige Beispiel, die mit diesem Werkzeug bearbeitet werden.

Lassen sich auch Aufgaben mit Wurzeln bearbeiten? Eine Standardaufgabe führt mich zu diesem Modell:

3	4	$48\cdot\pi$	"lok. Max."

$$extremf\left(150\cdot\sqrt{15^2+x^2}+250\cdot\sqrt{25+y^2},\frac{y}{5}=\frac{15}{x},x,y\right)$$

"Fehler: Bereichsfehler"

© die HB heißt nun hb1, nach v2 di

$sols2:=cZeros\left(\frac{d}{dv2}(hb1),v2\right)$

© die Werte für die erste Variable

$solsv1:=auxv1|v2=sols2$

$hb1:=hb1|v2=sols2$

$$approx\left(extremf\left(150\cdot\sqrt{15^2+x^2}+250\cdot\sqrt{25+y^2},\frac{y}{5}=\frac{15}{x},\right.\right.$$

x	y	"Optir
-6.16553+10.679·i	-3.0411−5.26734·i	2744.34+
-6.16553−10.679·i	-3.0411+5.26734·i	2744.34−
12.3311	6.0822	488:

$auxv1:=sols1[i]$

$hb1:=hb|v1=auxv1$

© die HB heißt nun hb1, nach v2 di

$sols2:=cZeros\left(\frac{d}{dv2}(hb1),v2\right)|v2>0$

© die Werte für die erste Variable

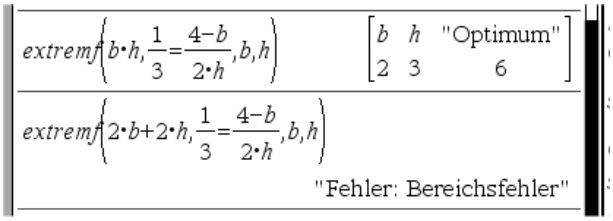
$solsv1:=auxv1|v2=sols2$

$hb1:=hb1|v2=sols2$

Damit entgehen uns aber mögliche negative Teillösungen und die Lösung 0. Ich habe noch keine Möglichkeit gefunden, dieses Problem zu lösen. Lassen Sie ich bitte wissen, wenn Sie eine Idee dafür haben. (Mit *Derive* geht es problemlos!)

Zwei weitere Aufgaben:

> Einem gleichschenkligen Trapez ($a = 4$, $h = 1{,}5$ und $c = 3$) ist das
> a) flächengrößte
> b) umfangsgrößte Rechteck einzuschreiben.



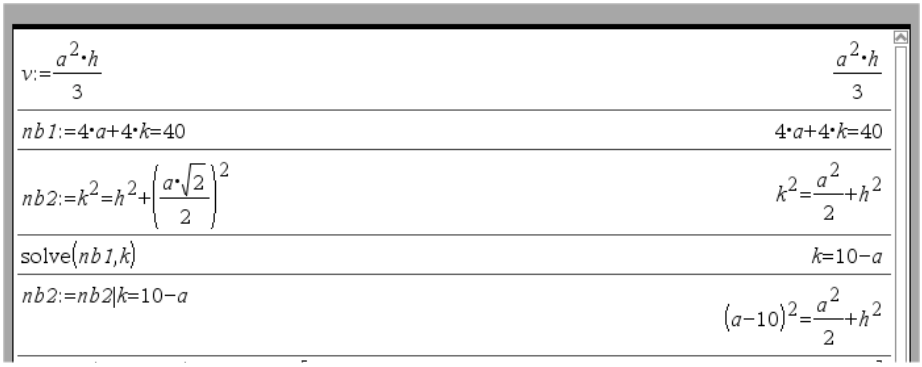
Wenn man das erste Ergebnis richtig interpretiert – und nicht nur gläubig abschreibt – wird man erkennen, dass es ein derartiges Rechteck mit der Höhe $h = 3$ in einem Trapez mit der Höhe 1,5 gar nicht geben kann! Hier liegt offensichtlich ein Randextremum vor. Beachten Sie bitte in diesem Zusammenhang die gestellte Aufgabe!

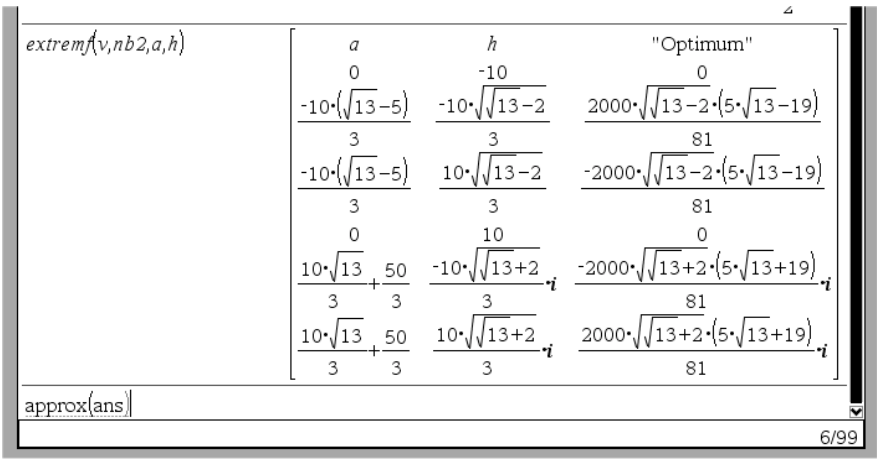
Bei der zweiten Aufgabe wird ein „Bereichsfehler“ angegeben. Hier hat die erste Ableitung gar keine Lösung, es gibt keine lokalen Extremwerte! Auch dazu gibt es eine Aufgabe.

Aufgabe: Ergänzen Sie das Programm für den Fall, dass es kein lokales Extremum gibt um eine entsprechende Ausgabe.

Zum Schluss soll noch eine rechentechnisch recht aufwändige Aufgabe dem CAS überlassen werden. (Wie wichtig die Lösung für uns ist, ist eine andere Frage!)

> Aus 40cm Draht soll das Kantenmodell einer geraden Pyramide mit quadratischer Grundfläche gefertigt werden. Bei welchen Abmessungen hat die Pyramide das größte Fassungsvermögen?





Nach Erledigung der Vorarbeit (= Reduzierung auf zwei Variable) erhalten wir eine sehr ausgebreitete Antwort, die am besten approximiert wird.

a	*h*	"Optimum"
0.	-10.	0.
4.64816	-4.22368	-30.4181
4.64816	4.22368	30.4181
0.	10.	0.
28.6852	-7.89202·*i*	-2164.62·*i*
28.6852	7.89202·*i*	2164.62·*i*

a	*h*	"Optimum"	"Art des Extremwts"
0.	-10.	0.	"lok. Max."
4.64816	-4.22368	-30.4181	"lok. Max."
4.64816	4.22368	30.4181	"lok. Min."
0.	10.	0.	"lok. Min."
28.6852	-7.89202·*i*	-2164.62·*i*	"komplexe Lösung"
28.6852	7.89202·*i*	2164.62·*i*	"komplexe Lösung"

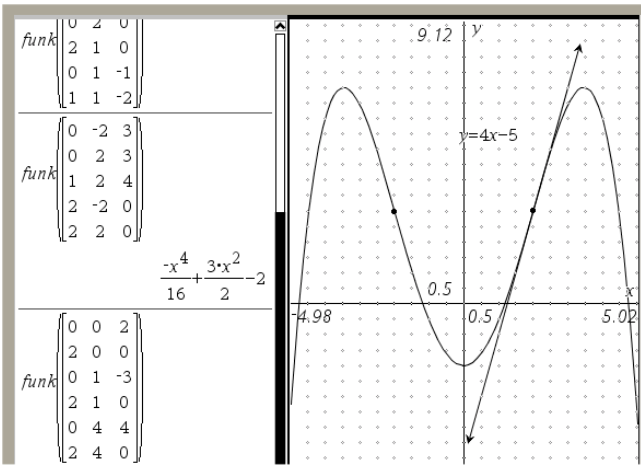
Die Länge der Seitenkante *k* muss dann noch berechnet werden.

Weitere Aufgaben

Erstellen Sie ein Programm zur Bearbeitung von „umgekehrten Kurvendiskussionen“ (= „Steckbriefaufgaben“), wie zB die nächste:

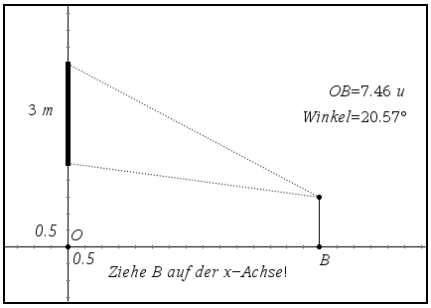
Der Graph einer Polynomfunktion hat in den Punkten P(-2|3) und Q(2|3) Wendepunkte, wobei die Tangente in Q die Steigung 4 hat.

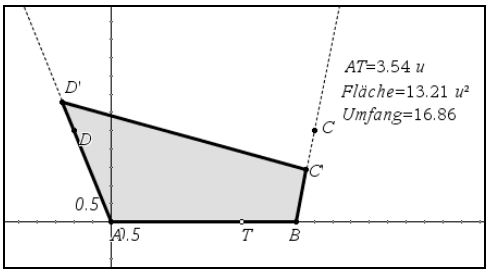
Lesen Sie die Matrix so:
„die nullte Ableitung an der Stelle -2 ist 3, die nullte Ableitung an der Stelle 2 ist 3, die erste Ableitung an der Stelle 2 ist 4 usw.



Lösen Sie die folgenden Extremwertaufgaben. Eine grafische Aufbereitung kann auf einer *Graphs & Geometry*-Seite mit Hilfe dynamischer Geometrie erfolgen. Die entsprechenden Bildschirme sind abgebildet.

Ein 3 m hohes Bild hängt an der Wand; sein unterer Rand ist 2,50 m über dem Fußboden. Wie weit muss sich ein Betrachter, dessen Auge sich 150 cm über dem Boden befindet, von der Wand entfernen, um das Bild unter einem möglichst großen Sehwinkel bewundern zu können?





Gegeben ist das Viereck ABCD mit A(0|0), B(5|0), C(5,5|2,5) und D/-1|2,5). T ist ein beliebiger Punkt auf AB. Auf AD ergibt sich der Punkt D' mit AD' = AT und auf BC der Punkt C' mit BC' = BT. Für welche Lage von T hat das entstehende Viereck ABC'D'

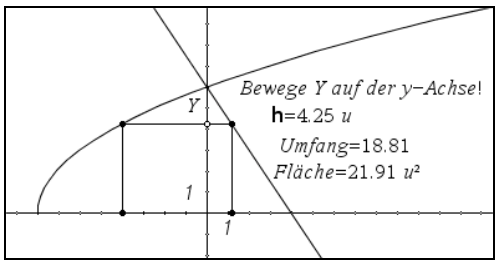
a) größten Umfang und

b) größten Inhalt?

Die nach rechts offene Parabel mit dem Scheitel in (-8|0) geht durch den Punkt (0|6). Die Parabel und die Gerade g: $3x + 2y = 12$ begrenzen mit $y = 0$ in der oberen Halbebene einen Bereich, dem ein achsenparalleles Rechteck eingeschrieben werden kann.

a) Bestimmen Sie das Rechteck mit dem größten Flächeninhalt.

b) Bestimmen Sie das Rechteck mit dem größten Umfang.

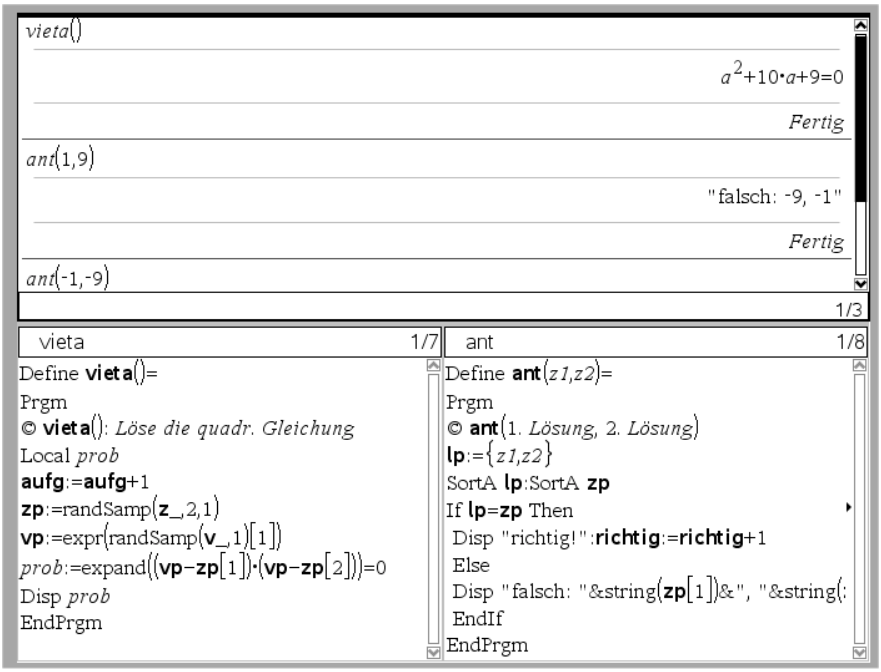


Schreiben Sie ein Programm für eine Funktionsdiskussion. Versuchen Sie es zuerst nur für Polynomfunktionen. In der Ausgabe sollen Nullstellen, Extremwerte und Wendepunkte ausgewiesen werden (eventuell mit Bestimmung der Art der Wendepunkte und der Ausgabe der Steigung in den Wendepunkten). Eine allfällig vorhandene Symmetrie kann auch angegeben werden.

Erzeugen Sie eine Bibliothek analysis und nehmen Sie extrem() usw. in diese auf.

6 Trainingsprogramme für Grundfertigkeiten

Man kann ein CAS auch sehr gut dazu nützen, um mathematische Grundfertigkeiten zu trainieren. Meine Erfahrung hat gezeigt, dass Schüler damit in Eigenverantwortung viel lieber arbeiten als mit Listen von Übungsbeispielen aus Lehrbüchern und Aufgabensammlungen. Ich möchte Ihnen hier ein erprobtes Übungsprogramm zum Trainieren des Vieta´schen Wurzelsatzes vorstellen:



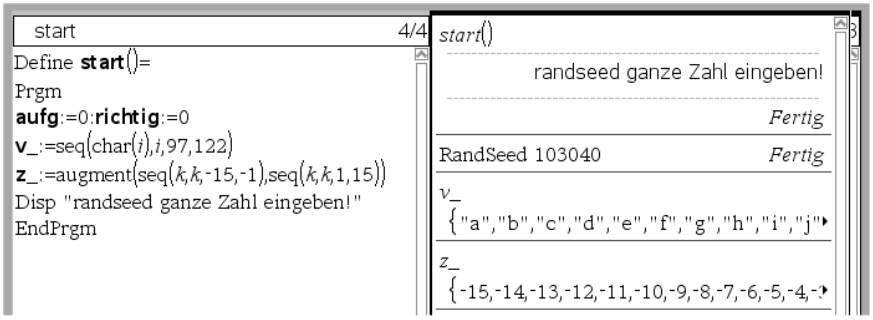
Als Übungsprogramm kann das nur dann richtig geeignet sein, wenn nicht immer die gleichen Aufgaben angeboten werden – da kommt wieder der Zufall ins Spiel – und wenn eine gewisse Kontrolle angeboten wird.

In der globalen Variablen *aufg* werden die gestellten Aufgaben mitgezählt – siehe nächste Seite die Beschreibung der Initialisierung mit start(). *zp* ist eine Zufallsauswahl (ohne Wiederholung) von zwei Zahlen aus der Liste *z_* und *vp* ist eine zufällig ausgewählte Variablenbezeichnung aus der Liste *v_*. *z_* und *v_* werden ebenfalls durch start() generiert.

Im obigen Bildschirm müssen Sie sich noch den Befehl **randseed Zahl** wie in Abschnitt 3 näher beschrieben wurde dazu denken. Damit werden immer andere Aufgaben gestellt, vorausgesetzt Sie nehmen nicht immer dieselbe **Zahl**.

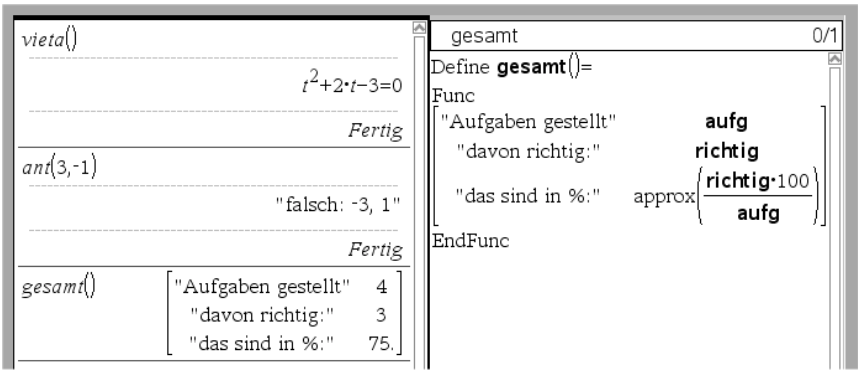
Mit **ant(1.Lösung, 2.Lösung)** teilt der Benutzer seine Vermutung dem Rechner mit und bekommt sofort die Rückmeldung. Im Hintergrund werden die richtigen Antworten in der globalen Variablen *richtig* mitgezählt.

Das nächste Bild zeigt start() und den Beginn einer Sitzung.

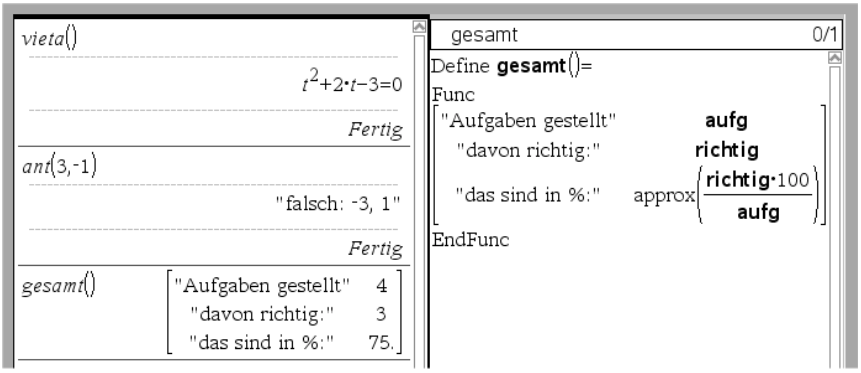


Beachten Sie, dass weder vieta() noch start() einen Parameter benötigen!

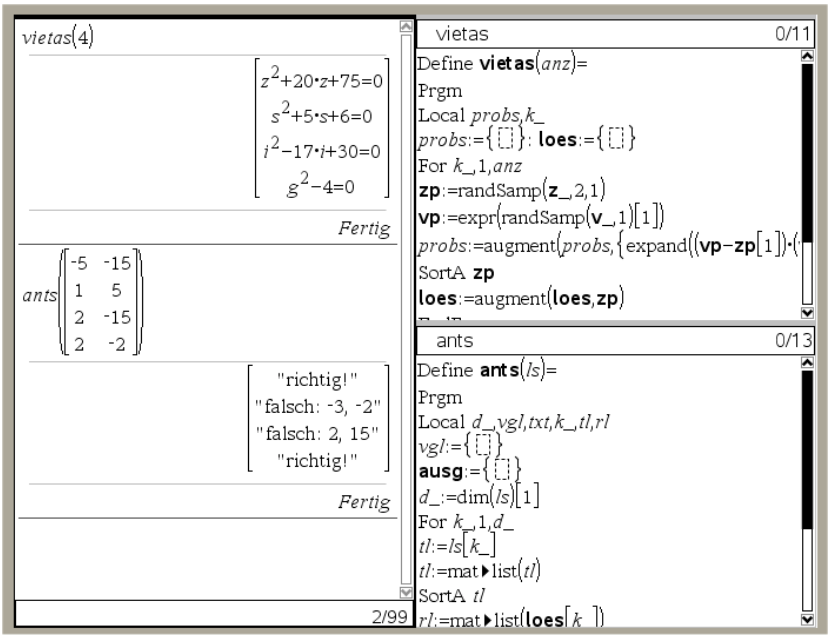
Sie sehen auch noch die beiden generierten Listen *v_* und *z_* (globale Variable). So könnte dann eine Sitzung ablaufen:



Wenn die Sitzung beendet wird, kann sich der Übende mit gesamt() eine Übersicht seiner Bemühungen ausgeben lassen:



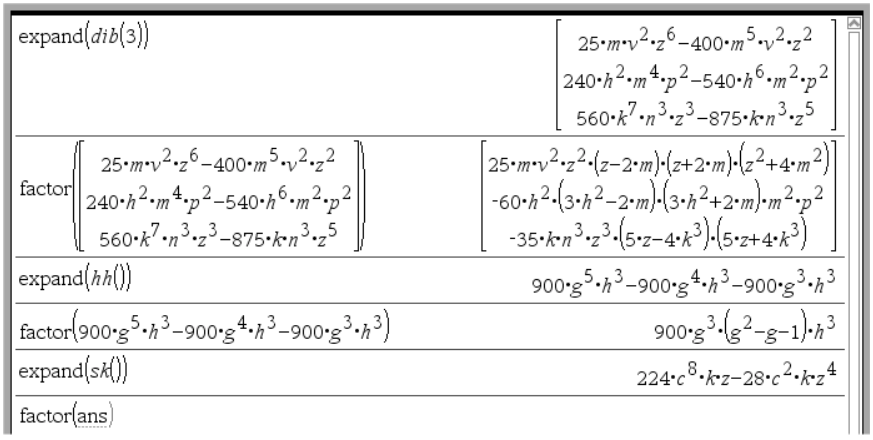
Ich stelle Ihnen noch eine zweite Variante vor, in der dem Übenden gleich ein Block von Übungsaufgaben angeboten wird. Mit vietas(k) wird eine Liste von k quadratischen Gleichungen mit ganzzahligen Lösungen ausgegeben. Die Antworten müssen im Programm ants in Form einer $k \times 2$ Matrix eingegeben werden.



Aufgaben

Ergänzen Sie start(), vietas() und ants() um Kommentare, die im Katalog aufscheinen und richten Sie eine Bibliothek training ein, in die sie die Programme, die in den nächsten Aufgaben gestellt werden, aufnehmen können.

Erstellen Sie ein Übungsprogramm zum Faktorisieren von Termen (zB zum Herausheben von gemeinsamen Faktoren, zum Zerlegen von vollständigen Quadraten, der Differenz zweier Quadrate, von Summe und Differenz von Kuben usw.)



Erstellen Sie ein Übungsprogramm zum Ausmultiplizieren von Quadraten und Kuben von Binomen usw.

7 Wir überwinden die Regression 4. Grades

Weithin bekannt - neuerdings auch in der Schulmathematik - sind lineare und quadratische Regression, die auf dem Prinzip der „kleinsten Quadrate“ beruhen.

Zur Erinnerung:

Gegeben sind empirische Daten als Zahlenpaare $\{(x_1,y_1), (x_2,y_2), ..., (x_n,y_n)\}$. Nach Darstellung dieser Daten als Streudiagramm in einem geeigneten Koordinatensystem liegt oft der Schluss nahe, dass zwischen den x- und y-Werten ein linearer Zusammenhang bestehen könnte. Die Regressionsrechnung sucht nun eine lineare Funktion $f(x) = k\,x + d$ so, dass die Summe der Quadrate der Differenzen aus den tatsächlichen und theoretischen Funktionswerten minimal wird:

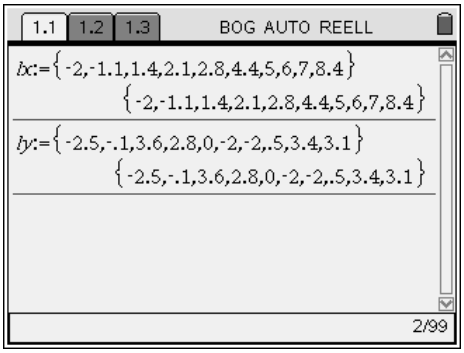
$$\Omega(k,d) \;=\; \sum_{i=1}^{n}(f(x_i) \;-\; y_i)^2 \;=\; \sum_{i=1}^{n}(k\,x_i + d \;-\; y_i)^2 \;=\; \text{Minimum}$$

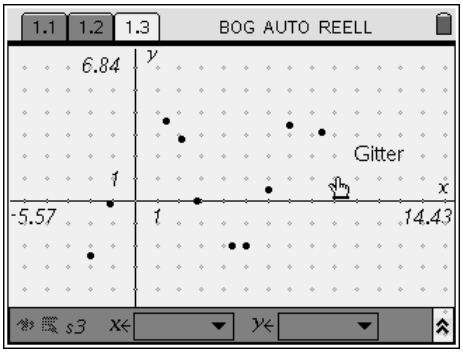
Das erweist sich als eine Extremwertaufgabe mit den beiden Variablen k und d. Man findet k und d, indem man $\Omega(k,d)$ partiell nach k und d differenziert und das entstehende lineare Gleichungssystem nach k und d auflöst.

Dieses Verfahren wird nun verallgemeinert. Es wird an einem Beispiel demonstriert:

x_i	-2,0	-1,1	1,4	2,1	2,8	4,4	5,0	6,0	7,0	8,4
y_i	-2,5	-0,15	3,6	2,8	0	-2,0	-2,0	0,5	3,4	3,1

Wir speichern die Daten in den beiden Listen *xl* und *yl* und machen uns eine Bild in Form eines Streudiagramms:





Hier ist lineare Regression sicher nicht mehr angebracht. Eine Polynomfunktion höherer Ordnung könnte passen, die Lage der Punkte erinnert aber eher an Winkelfunktionen. „Zufälligerweise“ liegt die Periodenlänge zwischen 6 und 7 ($\approx 2\pi$).

Wir suchen eine approximierende Funktion $\Phi(x)$ als Linearkombination der Funktionen $\{1, \sin(x), \cos(x)\}$.

$$\Phi(x) \;=\; a \cdot 1 \;+\; b \cdot \sin(x) \;+\; c \cdot \cos(x)$$

Damit ergibt sich eine Extremwertaufgabe mit den drei Variablen a, b und c:

$$\Phi(x_i) = a_1 f_1(x_i) + a_2 f_2(x_i) + a_3 f_3(x_i)$$

$$\Omega(a,b,c) = \sum_{i=1}^{N} (\overbrace{a \cdot 1 + b \cdot \sin x_i + c \cdot \cos x_i}^{} - y_i)^2 = \text{Minimum}$$

$$\frac{\partial \Omega}{\partial a} = \sum_{i=1}^{N} (2 (a + b \sin x_i + c \cos x_i - y_i) \cdot 1) = 0$$

$$\frac{\partial \Omega}{\partial b} = \sum_{i=1}^{N} (2 (a + b \sin x_i + c \cos x_i - y_i) \cdot \sin x_i) = 0$$

$$\frac{\partial \Omega}{\partial c} = \sum_{i=1}^{N} (2 (a + b \sin x_i + c \cos x_i - y_i) \cdot \cos x_i) = 0$$

Alle drei Gleichungen lassen sich durch 2 kürzen, ausmultiplizieren und umordnen:

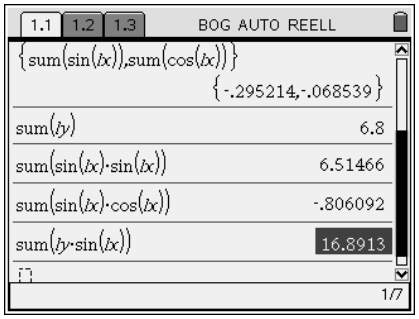
$$a \sum_N 1 \cdot 1 + b \sum_N \sin x_i \cdot 1 + c \sum_N \cos x_i \cdot 1 = \sum_N y_i \cdot 1$$

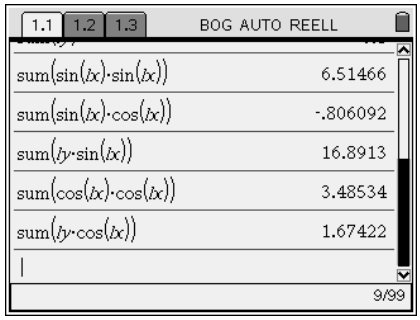
$$a \sum_N 1 \cdot \sin x_i + b \sum_N \sin x_i \cdot \sin x_i + c \sum_N \cos x_i \cdot \sin x_i = \sum_N y_i \cdot \sin x_i$$

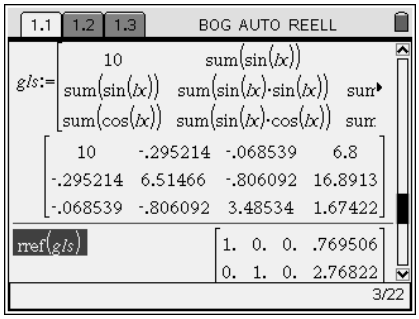
$$a \sum_N 1 \cdot \cos x_i + b \sum_N \sin x_i \cdot \cos x_i + c \sum_N \cos x_i \cdot \cos x_i = \sum_N y_i \cdot \cos x_i$$

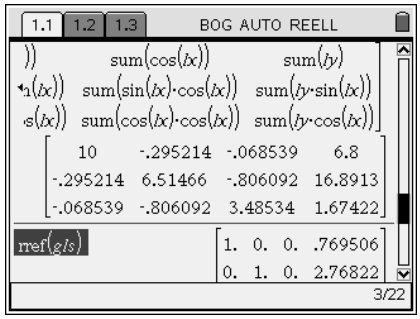
Wir stellen für die vorliegenden Daten ($N = 10$) das Gleichungssystem auf, lösen es nach den Variablen a, b und c auf und bestimmen die approximierende Funktion. Die so entstandene Regressionslinie tragen wir zum Streudiagramm ein.

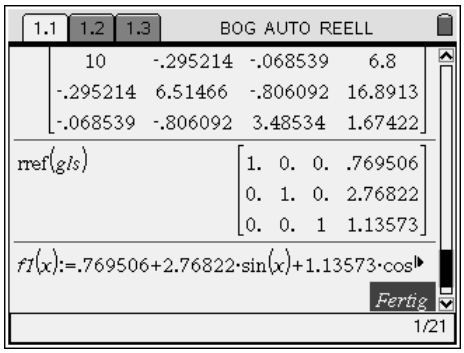
Hinweis: **sum(Liste)** ergibt die Summe aller Listenelemente. Dabei könnten wir die Zahlenwerte einzeln ausrechnen (lassen) und dann das Gleichungssystem lösen, oder wir übernehmen die Summen direkt zB in die Koeffizientenmatrix des Gleichungssystems.

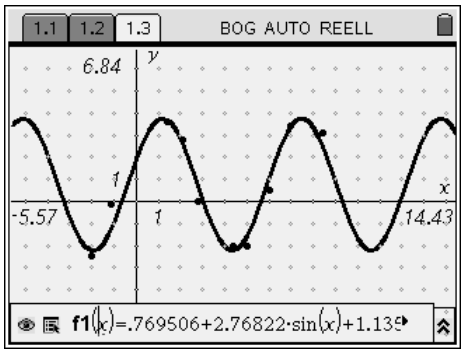






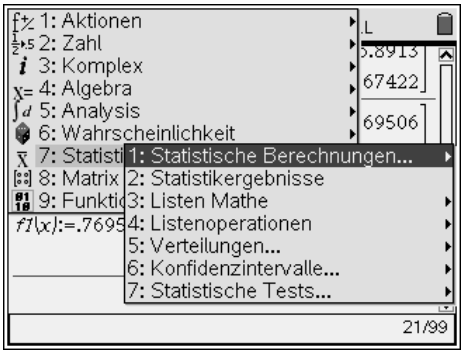


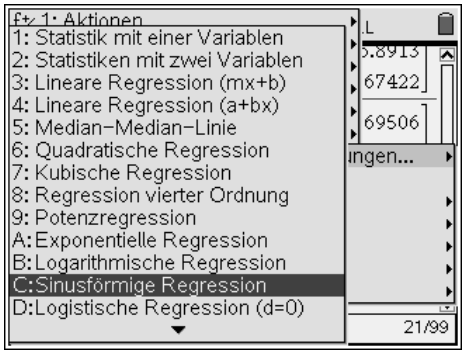


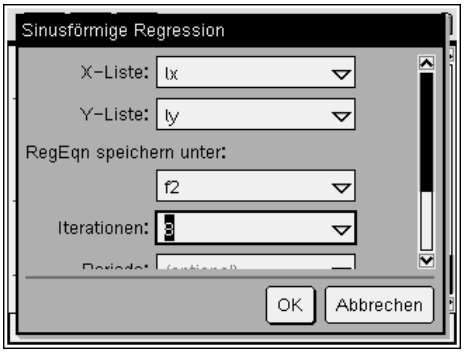


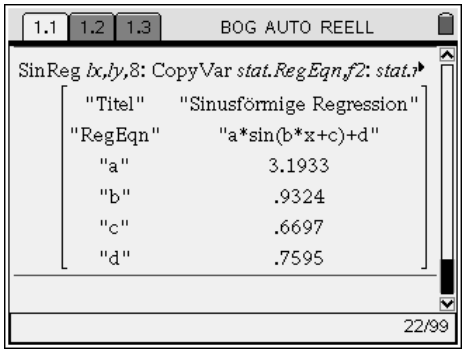
Mit den Werten für *a*, *b* und *c* definieren wir die Näherungsfunktion und ihr Graph im Streudiagramm zeigt, dass wir gar nicht so schlecht liegen dürften.

Zum Vergleich wenden wir aber die eingebaute „Sinusförmige Regression“ an:





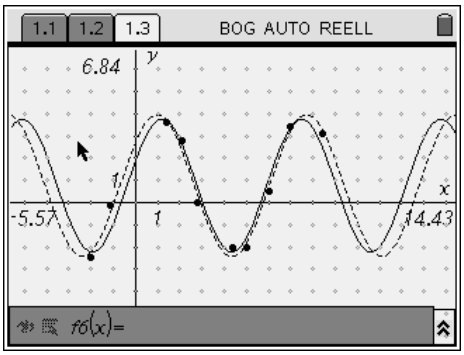




Die Regressionsfunktion wurde unter dem Namen $f2(x)$ gespeichert und kann nun zu $f1(x)$ ins Streudiagramm eingetragen werden.

Anschließend werden wir die Prozedur in einem kurzen Programm zusammenfassen.

$f2(x)$ wurde strichliert gezeichnet.

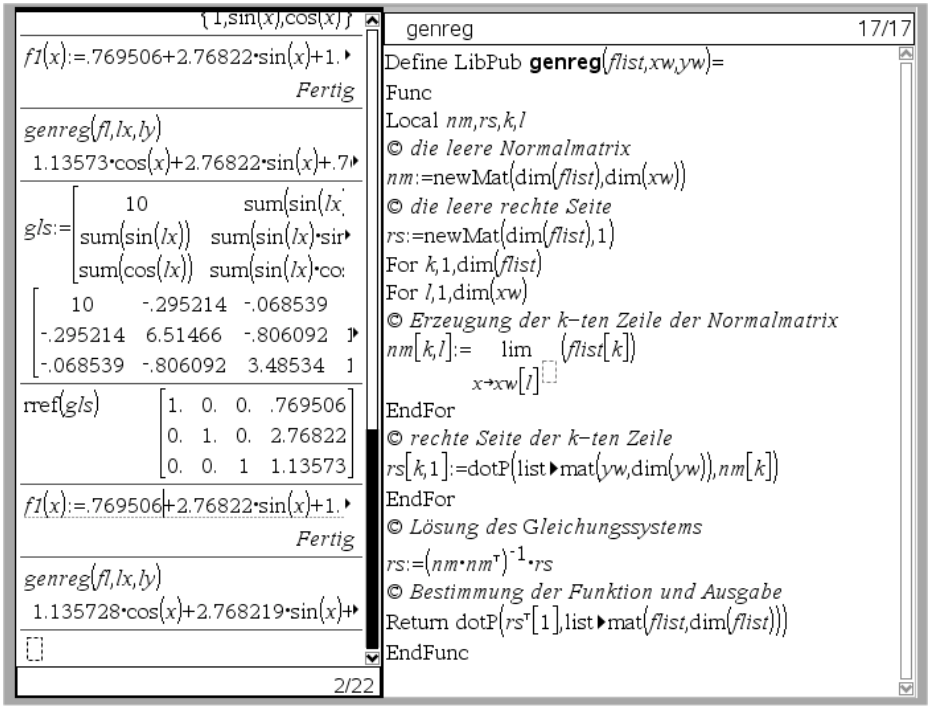


Das oben gezeigte Verfahren kann unter Einsatz von Matrizen auf elegante Weise durchgeführt werden. Dazu erzeugt man zuerst eine Matrix M - die so genannte **Normalmatrix**:

$$M = \begin{pmatrix} f_1(x_1) & f_1(x_2) & f_1(x_3) & \dots & f_1(x_{10}) \\ f_2(x_1) & f_2(x_2) & f_2(x_3) & \dots & f_2(x_{10}) \\ f_3(x_1) & f_3(x_2) & f_3(x_3) & \dots & f_3(x_{10}) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ \sin(x_1) & \sin(x_2) & \sin(x_3) & \dots & \sin(x_{10}) \\ \cos(x_1) & \cos(x_2) & \cos(x_3) & \dots & \cos(x_{10}) \end{pmatrix}$$

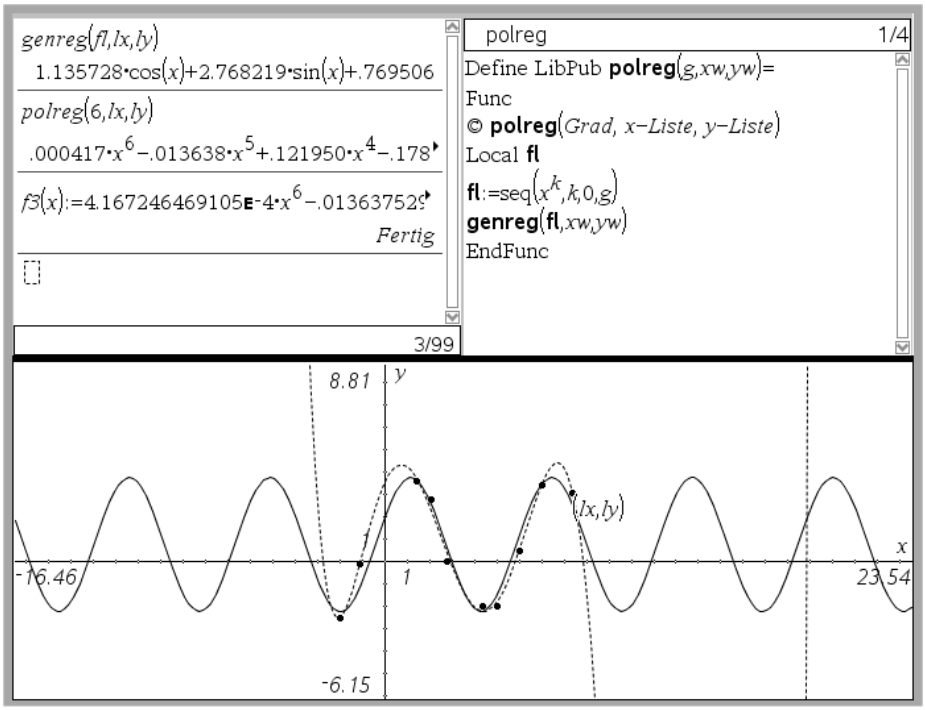
Es ist nun leicht zu sehen, dass das Produkt $M \cdot M^T$ (= die Transponierte von M) genau jene Matrix ist, die wir oben Element für Element bestimmt haben. Im Zentrum des Programms wird daher der Aufbau der Normalmatrix liegen. Wir zeigen das Programm der besseren Lesbarkeit halber wieder auf dem PC-Schirm.

Als Eingabeparameter habe ich die Liste der erzeugenden Funktionen, sowie die Listen der x- und y-Werte vorgesehen.



Sie erkennen die Übereinstimmung.

Für die polynomiale Regression vom Grad *g* muss die Funktionsliste der Potenzen von *x* programmintern generiert werden, diese wird dann an genreg übergeben.



Ich habe hier durch die 10 Punkte eine Polynomfunktion vom Grad 6 gelegt (punktierter Graph) . Im Bereich der Datenpunkte wäre die Funktion noch akzeptabel, die mögliche Periodizität kommt natürlich überhaupt nicht zum Ausdruck.

Etwas – was für viele vielleicht überraschend ist – interessantes passiert, wenn man mit dem CAS *Derive* diese selbst gemachte „trigonometrische Anpassung“ mit Hilfe der FIT-Funktion ausführt:

```
FIT([x, a + b·SIN(x) + c·COS(x)], pkte)
1.135727644·COS(x) + 2.768218762·SIN(x) + 0.7695058028
```

Hier ist offensichtlich auch dieser Algorithmus implementiert!

Wenn Sie die Datei unter dem Namen statistik im Folder `mylib` abspeichern, steht Ihnen diese Regression aus allen übrigen Applikationen zur Verfügung und allfällige andere Statistikwerkzeuge können später auch dazu genommen werden (wie zB die, die wir im nächsten Abschnitt erzeugen werden).

Aufgaben

Im obigen Beispiel konnte eine Periodenlänge von ca 2π angenommen werden. Ändern Sie diese „trigonometrische Regression“ so ab, dass sie auch eine vermutete Periodenlänge *p* berücksichtigen können. Dabei müssen Sie die Periodenlänge als weiteren Parameter in die Parameterliste aufnehmen.

Der Aufruf könnte lauten: trigreg(xListe,yListe,p).

Die Daten für eine „Lorenzkurve“ (= Quintilwerte) sind gegeben durch die folgenden Datenpunkte: (0|0), (0,2|0.04), (0.4|0.12), (0.6|0.35), (0.8|0.6) und (1|1). Die Kurve muss durch den Koordinatenursprung verlaufen. Suchen Sie mit Hilfe der vorliegenden Programme eine geeignete Funktion 4. Grades und vergleichen Sie mit einer kubischen Regressionslinie durch den Ursprung.

Vor allem im angelsächsischen Bereich ist die Median-Regression – abgekürzt *MedMed-Regression* – recht verbreitet. Auch die TI-Taschenrechner und daher auch TI-*Nspire* bieten diese Regression an.

Das Ergebnis ist eine Gerade, die durch die Punktwolke der Datenpaare zu legen ist. Man berechnet dieser Gerade auf die folgende Art und Weise:

Die vorliegenden Datenpaare werden nach steigenden *x*-Werten sortiert. Dann wird die Menge der Datenpaare in möglichst gleich große Gruppen geteilt. Falls die Anzahl der Datenpaare nicht durch drei teilbar ist, hat die mittlere Gruppe entweder ein Datenpaar mehr oder weniger als die beiden Randgruppen.

Für alle drei Gruppen wird der Medianpunkt bestimmt, dessen Koordinaten die Medianwerte der jeweiligen *x*- und *y*-Werte in den Gruppen sind. Wir bezeichnen diese Punkte der Reihe nach mit M1, M2 und M3. Die MedMed-Gerade verläuft parallel zur Verbindung von M1 und M3. Der Abschnitt auf der *y*-Achse dieser Geraden heiße $d1$. Durch M2 ist eine Parallele dazu zu legen und deren Abschnitt auf der *y*-Achse zu bestimmen, $d2$. Von der endgültigen Geraden kennt man nun bereits die Steigung. Ihr Abschnitt *d* auf der *y*-Achse ist ein gewichtetes Mittel aus $d1$ und $d2$, nämlich $(2 \cdot d1 + d2)/3$.

Erzeugen Sie Programm zur Bestimmung der MedMed-Regression. Erfinden Sie einen Satz von Daten und zeichnen Sie die MedMed-Gerade dazu. Überprüfen Sie Ihre gewonnene MedMed-Gerade mit der entsprechenden Funktion des TI-*Nspire*.

(Für die Berechnung des Medians gibt es eine *Nspire*-Funktion. Auch das Sortieren von Datenpaaren nach dem *x*-Wert ist möglich. Informieren Sie sich bitte im Referenzhandbuch.)

8 Mit dem TI-*Nspire* zum Lottohaupttreffer

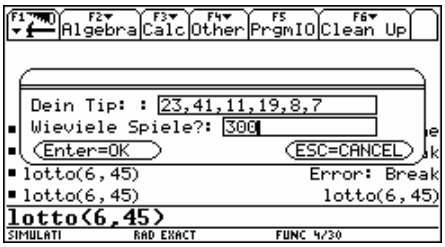
Eine der attraktivsten Einsatzmöglichkeiten des Computers ist die Durchführung von Simulationen. Das reicht vom kleinen Würfelspiel bis zu hochkomplexen Simulationen von Wirtschaftsabläufen, Simulationen von Auswirkungen von Umweltveränderungen, Flugsimulatoren usw.

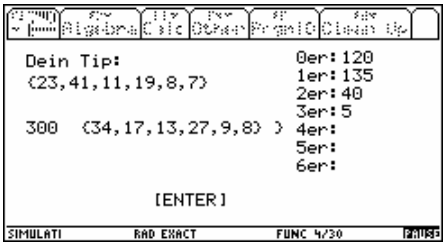
Wir wollen im letzten Abschnitt das überall bekannte Lottospiel simulieren und gleichzeitig auch die auftretenden Häufigkeiten graphisch darstellen.

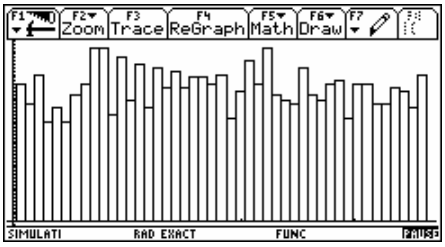
Ich habe dazu ein selbst erstelltes Programm für den TI-92/Voyage 200 gefunden. Da sieht die Sache so aus:

Auf den genannten Rechnern ist die Eingabe über einen Dialog möglich.

Hier werden 300 Ziehungen simuliert, dann wird eine Übersicht der Anzahl der Richtigen und ein Histogramm der absoluten Häufigkeiten der gezogenen Zahlen ausgegeben.







Hier wird die österreichische Version des Lottospiels, nämlich „*6 aus 45*“ simuliert: auf 6 Zahlen zwischen 1 und 45 wird gesetzt. Bei der Ziehung wird dann auch noch eine Kugel mit einer „Zusatzzahl“ gezogen, die wir hier nicht berücksichtigen wollen.

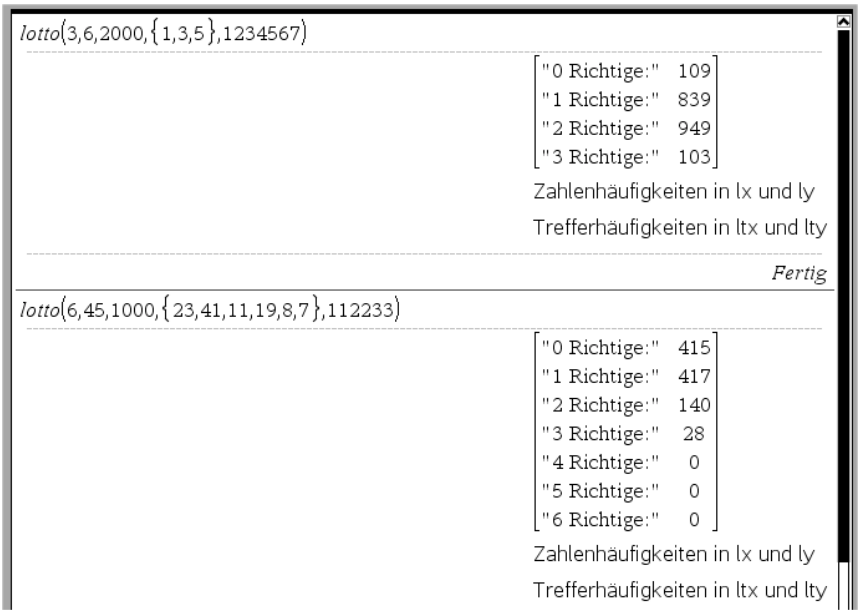
In diesem Programmlauf werden 300 Ziehungen von *6 aus 45* durchgeführt, wobei mein persönlicher Tipp aus den Zahlen 23, 41, 11, 19, 8 und 7 besteht.

Mit TI-*Nspire* müssen wir auf den Eingabedialog verzichten. Alle Parameter sind in der Parameterliste des Programms anzuführen. Ich denke an einen Programmaufruf mit der Syntax:

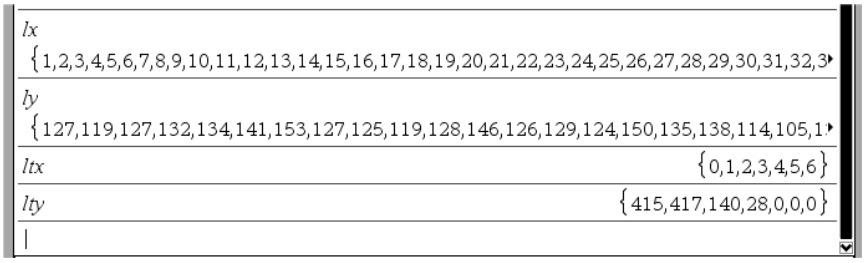
lotto(Klassenzahl, Anzahl aller Nummern, Anzahl der Ziehungen, Tippliste, Startzahl für den Zufallsgenerator).

Wir können dann nicht nur *6 aus 45* sondern auch *2 aus 10* oder *10 aus 100* spielen. Die jeweiligen Ergebnisse bilden die Grundlage für weitere statistische Untersuchungen bzw. für Bestätigungen von wahrscheinlichkeitstheoretischen Überlegungen.

Auf diese TI-*Nspire*-Version des Lottoprogramms wollen wir hin arbeiten:



Dies sind die außerhalb des Programms gespeicherten Werte der globalen Variablen, deren Beschreibung gleich folgt.



Die Ergebnisse der Simulationen von 2000 Ziehungen von *3* aus *6* und von 1000 Ziehungen von *6 aus 45* kann man aus der Matrix herauslesen: Bei *6 aus 45* habe ich 3 Richtige insgesamt 28 mal erreicht, 2 Richtige 140 mal, 417 mal habe ich eine genau eine Zahl erraten und bei 415 Ziehungen habe ich keine einzige Zahl richtig auf meinem virtuellen Lottoschein angekreuzt. Die absoluten Häufigkeiten des Auftretens der Zahlen von 1 bis 45 sind in der Liste *ly* gespeichert, während in *lx* die Zahlen von 1 bis 45 selbst abgelegt sind. Dies brauchen wir dann für die grafische Darstellung.

ltx und *lty* sind die entsprechenden Listen für die Zufallsvariable „*Anzahl der richtigen Zahlen bei einer Ziehung*“.

Bevor wir uns das Programm näher ansehen, wollen wir die Gewissheit haben, dass die Simulation auch richtig ist. Dazu ist es notwendig, die theoretischen Wahrscheinlichkeiten mit den experimentellen relativen Häufigkeiten zu vergleichen.

Bei den Ziehungen handelt es sich um klassische Experimente, die mit der hypergeometrischen Verteilung beschrieben werden. Aus einer Grundmenge von N (= 45) Elementen, in der sich A (= 6) Merkmalträger (meine Zahlen) befinden, werden Stichproben vom Umfang n gezogen. Wir suchen die Wahrscheinlichkeit, dass in der Stichprobe genau a (= 0, 1, 2 ,3, 4, 5, 6) Merkmalträger enthalten sind.

$$p(X = a) = \frac{\binom{A}{a} \cdot \binom{N-A}{n-a}}{\binom{N}{n}}.$$

Erwartungswert und Standardabweichung der Verteilung sind gegeben durch:

$$\overline{X} = \frac{A}{N} \cdot n \quad \text{und} \quad \sigma^2 = n \cdot \frac{A}{N} \cdot \left(1 - \frac{A}{N}\right) \cdot \frac{N-n}{N-1}.$$

$$\text{seq}\left(\frac{\text{nCr}(6,x)\cdot\text{nCr}(39,6-x)}{\text{nCr}(45,6)}, x, 0, 6\right)$$

$$\left\{\frac{155363}{387860}, \frac{82251}{193930}, \frac{82251}{543004}, \frac{9139}{407253}, \frac{741}{543004}, \frac{39}{1357510}, \frac{1}{8145060}\right\}$$

$$\text{approx}\left(\left\{\frac{155363}{387860}, \frac{82251}{193930}, \frac{82251}{543004}, \frac{9139}{407253}, \frac{741}{543004}, \frac{39}{1357510}, \frac{1}{8145060}\right\}\cdot 1000\right)$$

$$\{400.565, 424.127, 151.474, 22.4406, 1.36463, .028729, .000123\}$$

$$\text{approx}\left(\text{seq}\left(\frac{\text{nCr}(6,x)\cdot\text{nCr}(39,6-x)}{\text{nCr}(45,6)}, x, 0, 6\right)\cdot 300\right)$$

$$\{120.169, 127.238, 45.4422, 6.73218, .409389, .008619, .000037\}$$

Beachten Sie den Einsatz der **seq**-Funktion zu Berechnung der Folge von Wahrscheinlichkeiten! Mit **nCr(k,n)** berechnen Sie den Binomialkoeffizient $\binom{k}{n}$.

Die theoretischen Erwartungswerte für die beiden durchgeführten Zufallsexperimente sind 1,5 bzw. 0,8. Die beiden Experimente liefern die Mittelwerte 1,52 und 0,78. Den Vergleich der theoretischen und experimentellen Varianzen überlasse ich dem Leser.

An den Anfang des Programms stellen wir die Vorbereitung der Ausgabematrix:

```
lotto                                                    20/20
Define LibPub lotto(klasse,los,spiele,tipp,rdseed)=
Prgm
© lotto(Kl, Los, Spiele, {Tipp},rdseed )
Local treffer,lose,ziehung,tr,i,k,j
RandSeed rdseed
treffer:=newMat(klasse+1,2)
For i,0,klasse
treffer[i+1,1]:=string(i)&" Richtige:"
EndFor
```

Wir öffnen den Editor für das Programm lotto mit den angezeigten Programmparametern. Es ist denkbar, dass wir noch andere Simulationsprogramme erstellen, die in der Bibliothek simulation zusammengefasst werden, daher ermöglichen wir den öffentlichen Bibliothekszugriff.

Alle nicht weiter benötigten Variablen werden als lokale Variable definiert. Mit *treffer* erzeugen wir eine zweispaltige Matrix mit Überschrift, in der wir die Häufigkeiten der aufgetretenen Treffer – von keinem bis alle Nummern (6) werden gezogen – zusammengefasst ausgeben werden. Vorerst sind alle Elemente dieser Matrix 0. (Davon können Sei sich mit der Anweisung **disp treffer** unmittelbar vor Beginn der **For**-**EndFor**-Schleife überzeugen.)

In dieser Schleife generieren wir die Beschreibung der Ergebnisse, beginnend mit „**0 Richtige:**“.

```
lose:=seq(l_,l_,1,los):lx:=lose: ly:=newList(los)
For i,1,spiele
ziehung:=randSamp(lose,klasse,1):SortA ziehung
tr:=0
For k,1,klasse
   ly[ziehung[k]]:=ly[ziehung[k]]+1
   For j,1,klasse
      If ziehung[k]=tipp[j]:tr:=tr+1
   EndFor
EndFor
treffer[tr+1,2]:=treffer[tr+1,2]+1
EndFor
Disp treffer
ltx:=seq(k_,k_,0,klasse)
lty:=mat▸list(treffer^T[2])
Disp "Zahlenhäufigkeiten in lx und ly"
Disp "Trefferhäufigkeiten in ltx und lty"
EndPrgm
```

lose ist eine Liste der Zahlen von 1 bis *los* (45). Für die Beschreibung der absoluten Häufigkeiten der gezogenen Zahlen werden die Werte der Zufallsvariablen (1 – 45) benötigt, für die ich die Liste *lx* anlege, die ja inhaltsgleich mit *lose* ist. Die Liste *ly* mit gleich vielen Elementen wird vorbereitet, in der die Häufigkeiten an den entsprechenden Stellen abgelegt werden. In diesen beiden Listen finden wir dann die Koordinaten der Punkte für die Darstellung der Häufigkeiten zumindest in einem Streudiagramm.

Dann wird eine **For**-**EndFor**-Schleife geöffnet, die insgesamt *spiele*-mal durchlaufen wird. In jedem Spiel wird eine *ziehung* von *klasse* Elementen aus der Liste *lose* vorgenommen. Da ist der **randSamp**-Befehl mit dem dritten Parameter 1 für das Ziehen ohne Zurücklegen ein ausgezeichnetes Hilfsmittel (siehe Seite 21). Das Sortieren der Zahlen ist nicht notwendig,

aber wenn Sie die Ziehungen mit einer **Disp**-Anweisung ausgeben wollen, dann sieht eine geordnete Liste einfach besser aus. Außerdem will ich darauf hinweisen, dass innerhalb eines Programms Sortierungen vorgenommen werden können, innerhalb einer Funktion aber nicht. In der Variablen *tr* wird für jede Ziehung die Anzahl der Richtigen mitgezählt. Vor jeder Ziehung wird *tr* auf 0 zurück gestellt.

Bleiben wir bei *6 aus 45* und nehmen wir an, dass die Ziehung {4, 12, 19, 38, 40, 42} ergeben hat. Für alle 6 Elemente wird die Häufigkeit des Auftretens in der Liste *ly* um eins erhöht. (Wenn zB $k = 3$, dann wird *ly*[19] um eins vermehrt.

In der nächsten Schleife wird nun das *k*-te gezogene Element mit allen Elementen meines Tipps verglichen und wenn eine Übereinstimmung auftritt, dann wird die Trefferanzahl *tr* um 1 erhöht. Am Ende eines jeden Spiels wird das entsprechende Trefferfeld – in der zweiten Spalte der Ausgabematrix – um eins erhöht.

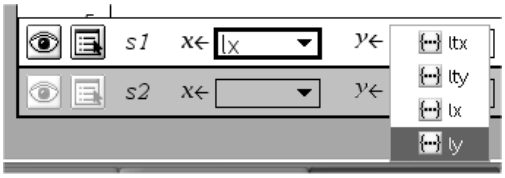
In den Listen *ltx* und *lty* sammeln wir die Koordinaten zur Darstellung der Häufigkeiten der Zufallsgröße *Anzahl der Richtigen bei einer Ziehung*.

Wenn alle Ziehungen durchgeführt worden sind, bleiben nur mehr die Ausgabe des Ergebnisses und ein Hinweis, in welcher Form die zusätzlich gewonnenen Daten abgerufen werden können.

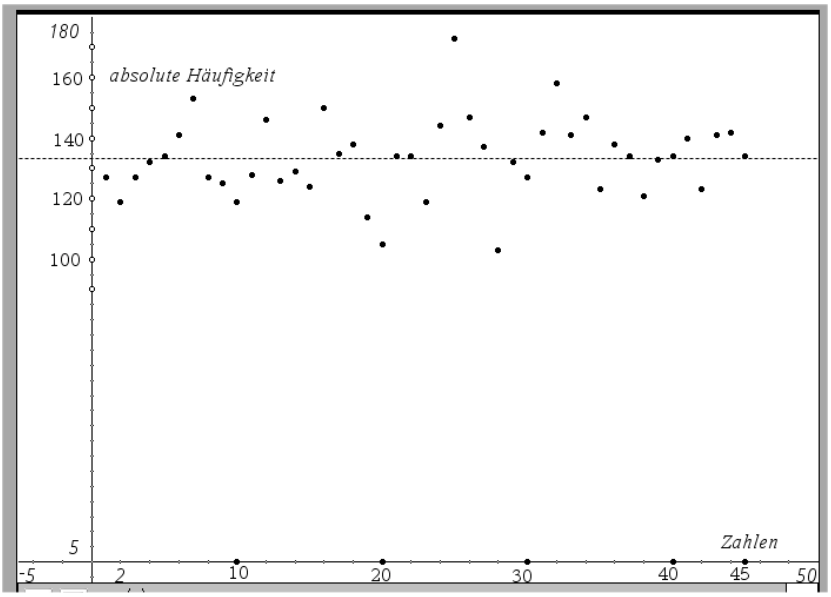
Nun soll unsere Simulation auch noch graphisch dargestellt werden. Da die Häufigkeiten bereits vorliegen, werden wir nicht auf die *Data & Statistics*-Applikation zurückgreifen, sondern auf einer *Graphs & Geometry* das Diagramm selbst erstellen.

Nach dem Öffnen der Applikation legen Sie zuerst wie auf Seite 23 gezeigt, die Darstellungsform *Streuplot* fest. Dann stellen Sie den Zeichenbereich ein. Über die Schaltfläche mit dem Händchen erreichen Sie den Menüpunkt *Dialogfeld Achseneinstellungen*. Nach den Daten der Listen *lx* und *ly* legen Sie die Fensterparameter etwa so fest, wie daneben gezeigt.

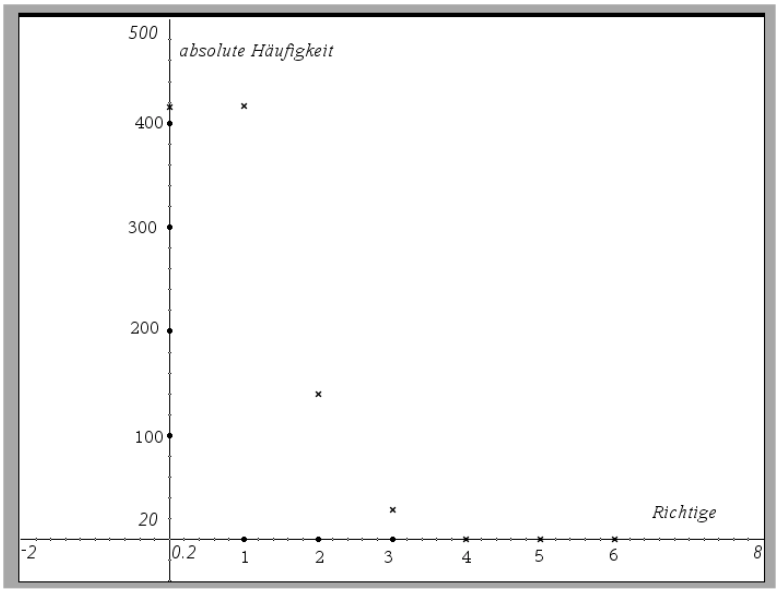
Nun müssen Sie nur noch für die *x*- und *y*-Werte die Listen *xl* und *yl* festlegen.



Sie können die Skalierung gestalten, die Achsen beschreiben und nach einem Wechsel in die Darstellungsform Funktion auch noch den theoretischen Erwartungswert für das Auftreten aller Zahlen zwischen 1 und 45 als waagrechte Linie $y = \frac{6000}{45} \approx 133,3$ eintragen. Die Grafik sollte dann etwa so aussehen:



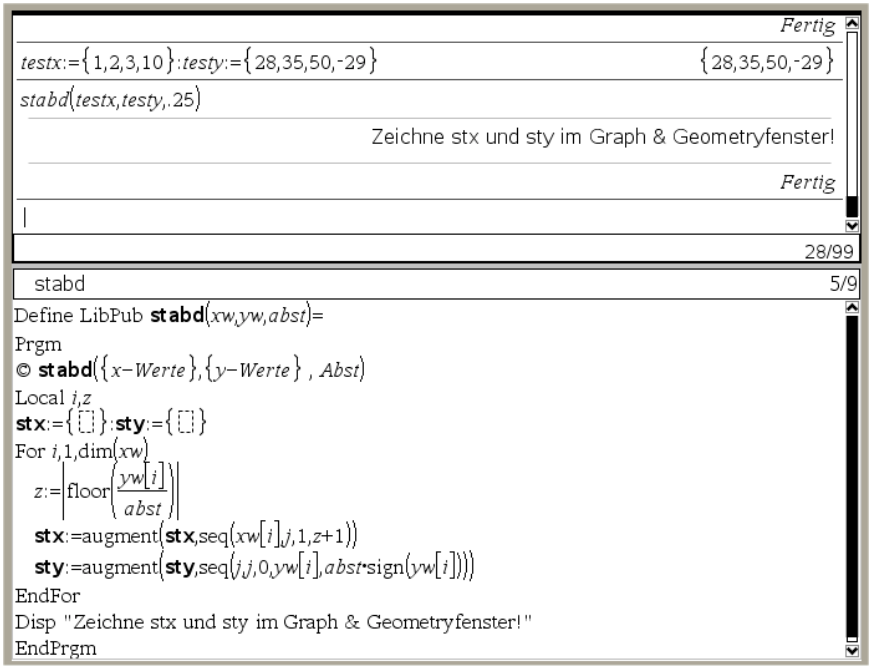
Die Darstellung der Verteilung der „Richtigen“ beim Tippen können Sie sicherlich selbst erreichen. Entweder arbeiten Sie im gleichen Zeichenfenster oder Sie öffnen ein neues.



Wie bereits oben bemerkt, lassen sich mit der *Data & Statistics*-Applikation Histogramme und Kastendiagramme nur von einer Variablen erzeugen. Wir hätten daher im Programm ganz einfach Listen mitschreiben müssen, in denen gezogene Zahl um gezogene Zahl, bzw. die Richtigen bei jedem Spiel eingetragen werden. Diese Listen können dann statistisch aufbereitet werden. Leider sind wir da auch am PC den Beschränkungen des Taschenrechners unterworfen, denn die Tabelle der *Lists & Spreadsheet*-Applikation gestattet nur bis zu maximal 2500 Zeilen. Bei 1000 Ziehungen von je 6 Kugeln müsste für die gezogenen Zahlen eine Liste von 6000 Elementen eingetragen werden, und das sind eindeutig zu viele.

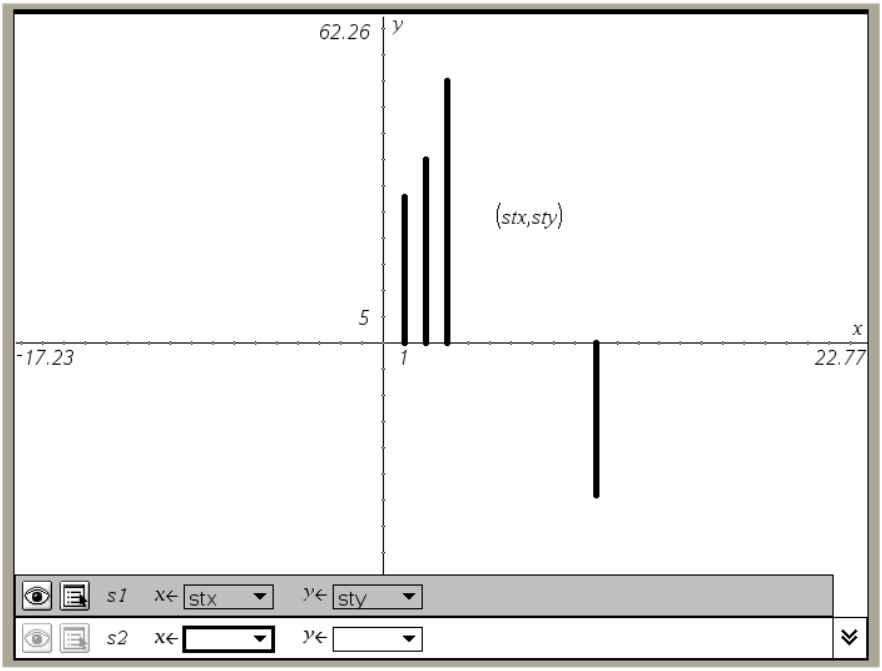
Wir wollen aber die Gelegenheit nutzen und unabhängig von den angebotenen Werkzeugen ein Programm zur Erstellung eines Stab-, bzw. eines Histogramms schreiben, das aus einer Werteliste mit zugehöriger Häufigkeitsverteilung diese Diagramme ins *Graphs & Geometry*-Fenster zeichnen lässt.

Wir beginnen mit dem Stabdiagramm, wobei ich drei Eingabeparameter vorsehe. Die Listen der *x*- und *y*-Werte (wenn es sich um keine Häufigkeiten handelt, können diese auch negativ sein!) und den senkrechten Abstand in dem ich einzelne Punkte „male“, die in Summe dann die Stäbe darstellen sollen.



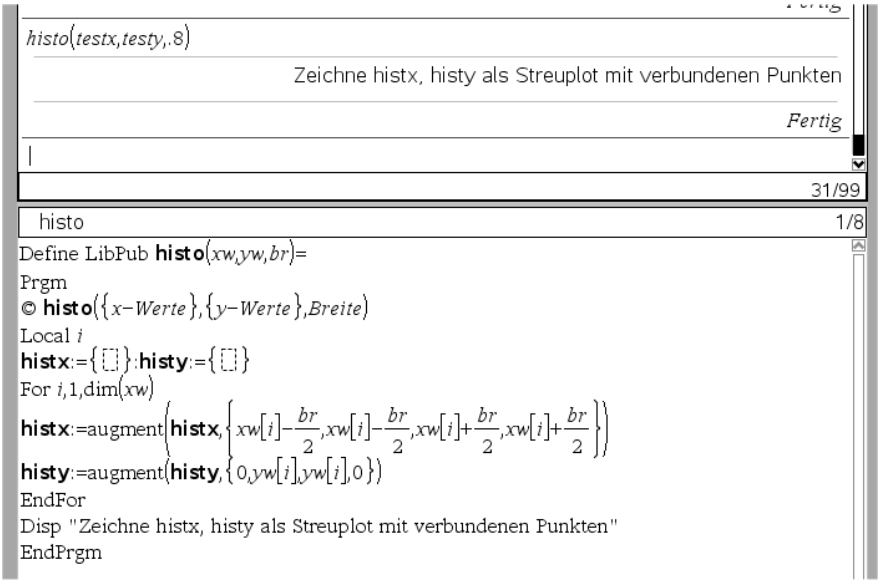
Die interessanten Zeilen des Programms sind die beiden Zuweisungen für stx und sty, in denen die Listen der *x*- und *y*-Werte in Einzelpunkte zerlegt werden. Der erste *y*-Wert ist 28, daher machen wir draus 28/0,25 = 112 + 1 = 113 Punkte (weil auf der *x*-Achse soll ja auch ein Punkt liegen), deren *y*-Koordinaten Werte von 0 bis 28 mit dem Abstand 0,25 annehmen.

Wie sieht das Stabdiagramm, das zu den Listen *testx* und *testy* generiert wird, aus?



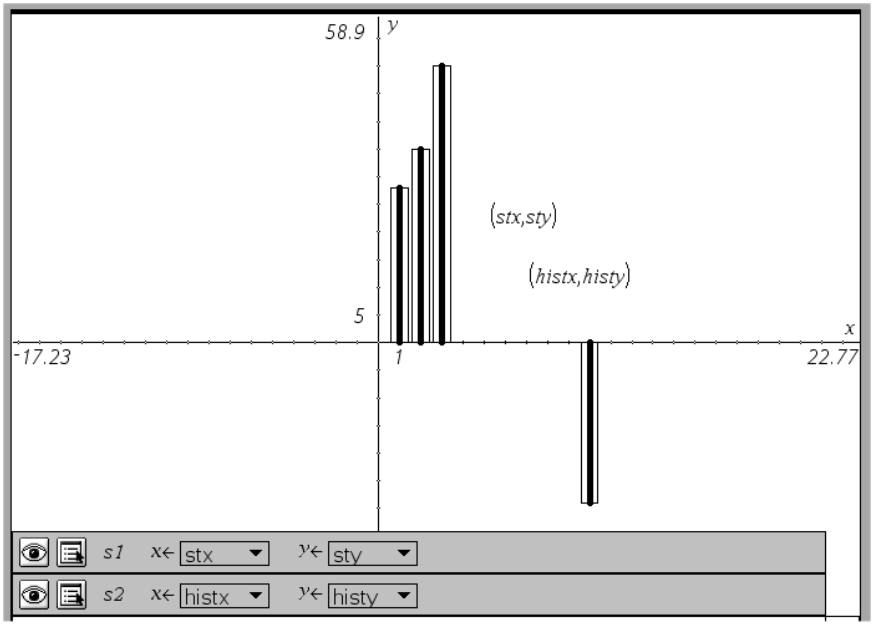
Den Punkten des Streudiagramms – in Wirklichkeit ist es ein solches – verleiht man das Attribut „voller Kreis“. Natürlich muss man auch hier aufpassen, dass man nicht zu viele Punkte erzeugt, denn sonst überschreitet man wieder die Kapazität des Programms.

Und nun zum Histogramm:

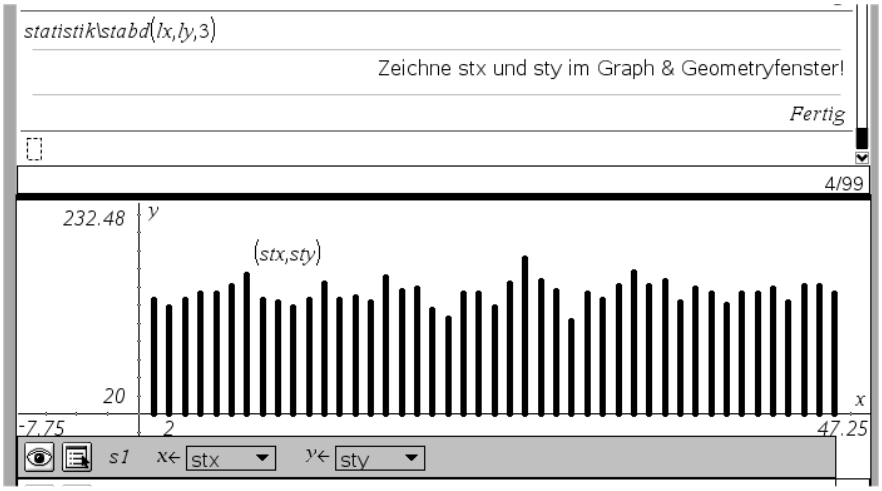


Hier zeichnen wir ganz einfach Punkte, die, wenn sie verbunden werden, einen Linienzug ergeben, der die Rechtecke mit den verbindenden Strecken auf der *x*-Achse darstellt. Diese Rechtecke haben die Höhe der jeweiligen *y*-Werte und sind mit der Breite *br* symmetrisch um die *x*-Werte ausgestattet.

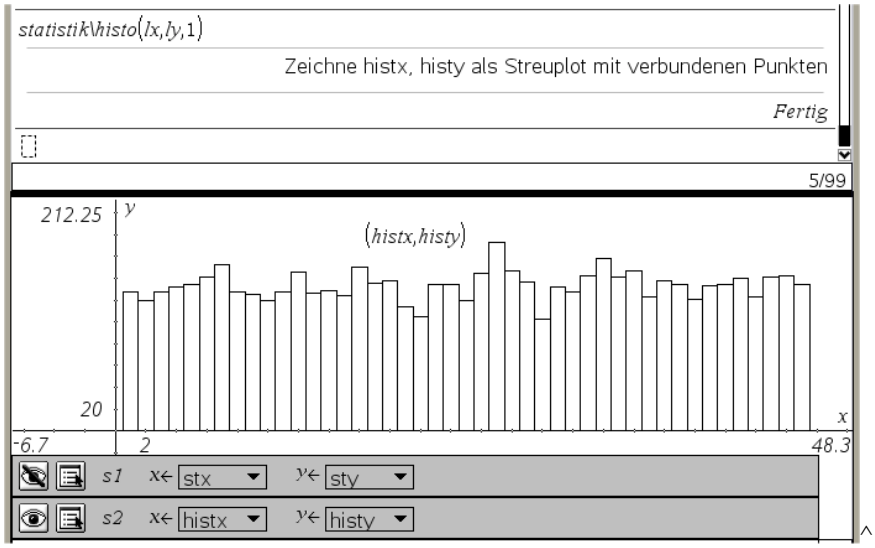
Wir zeichnen das Histogramm zum Stabdigramm für die Testwerte von vorhin:



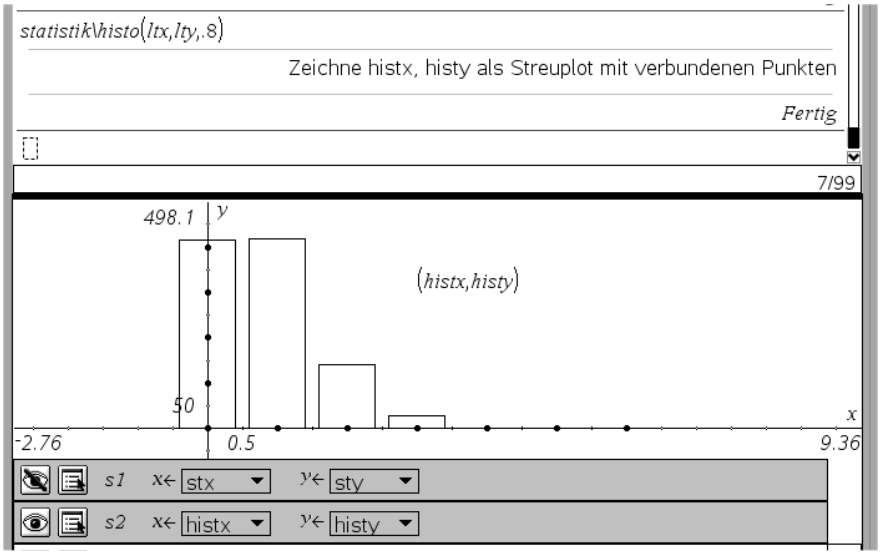
Nachdem wir diese beiden Dienstprogramme in der Bibliothek statistik gespeichert haben, öffnen wir wieder die Simulation, aktualisieren die Bibliotheken, rufen stabd auf und zeichnen das Stabdiagramm:



Und das ist nun die Darstellung der absoluten Häufigkeiten in einem Histogramm:



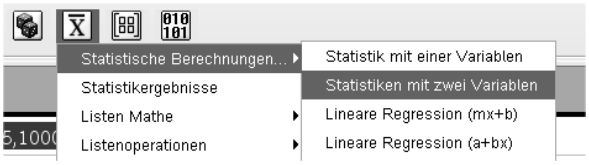
Hier zeige ich Ihnen noch die Darstellung der Häufigkeit der „Richtigen“ bei einem abgegebenen Tipp. Eine geeignete Beschriftung der Achsen kann noch hinzugefügt werden.



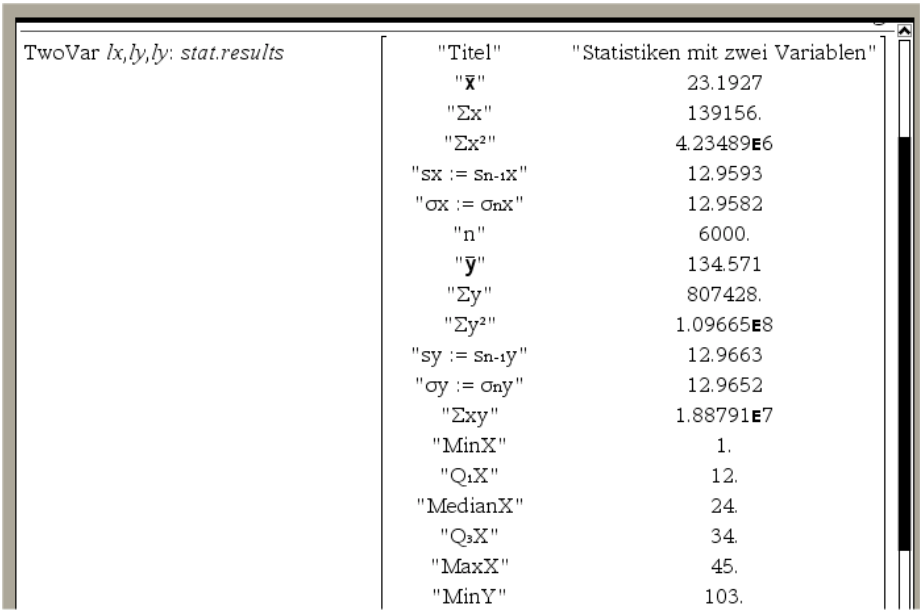
Von den gängigen Diagrammtypen fehlen noch Kasten- und Kreisdiagramm (Boxplot und Pie Chart). Auch das ließe sich programmieren.

Die gewonnenen Listen können für statistische Untersuchungen verwendet werden. Machen Sie dazu mit mir einen kleinen Ausflug in die beschreibende Statistik.

Über die $\bar{X}$-Schaltfläche gelangen Sie zu den Statistischen Berechnungen und dann weiter zur Statistik mit zwei Variablen:



In einer Ausgabematrix finden Sie alle interessanten Ergebnisse, die auch einzeln abgefragt und somit weiter verwendet werden können.



TwoVar *lx,ly,ly: stat.results*		
	"Titel"	"Statistiken mit zwei Variablen"
	"$\bar{x}$"	23.1927
	"Σx"	139156.
	"Σx^2"	4.23489E6
	"$sx := s_{n-1}x$"	12.9593
	"$\sigma x := \sigma_n x$"	12.9582
	"n"	6000.
	"$\bar{y}$"	134.571
	"Σy"	807428.
	"Σy^2"	1.09665E8
	"$sy := s_{n-1}y$"	12.9663
	"$\sigma y := \sigma_n y$"	12.9652
	"Σxy"	1.88791E7
	"MinX"	1.
	"Q_1X"	12.
	"MedianX"	24.
	"Q_3X"	34.
	"MaxX"	45.
	"MinY"	103.

stat.Q_1X	12.
stat.MedianX	24.
stat.Q_3X	34.

Das können Sie natürlich mit den Trefferlisten wiederholen und erhalten die statistischen Daten für die Anzahl der „Richtigen“ bei den vorgenommenen 1000 Ziehungen.

Aufgaben

Ergänzen die das Programm lotto um wichtige Plausibilitätskontrollen:

a) der Umfang der Ziehung darf nicht größer sein als der Vorrat aller Zahlen,

b) Die Anzahl der Elemente im abgegebenen Tipp muss dem Umfang der Ziehung entsprechen,

c) Die Zahlen im eigenen Tipp müssen im Vorrat aller „Kugeln" enthalten sein.

Schreiben Sie das Programm so um, dass Sie die „Ein-Variablen-Statistik" in der *Data & Statistics*-Applikation einsetzen können und auch das *Nspire*-Histogramm und das *Nspire*-Kastendiagramm direkt einsetzen können (Hinweis auf Seite 55).

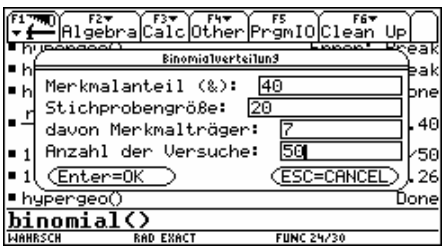
Verwenden Sie das Grundgerüst von lotto zur Simulation von zahlreichen weiteren Zufallsexperimenten, wie zB:

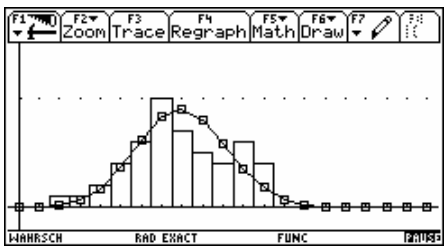
- nur gerade/ungerade Gewinnzahlen werden gezogen
- mindestens 1, 2, 3, … Gewinnzahlen bei einer Ziehung werden erreicht, wenn ein Jahr lang jede Woche 6 Tipps abgegeben werden
- der Geburtstag eines Spielers sei der 5. Dezember. 5 und 12 sollen gezogen werden
- die Zahl *m* sei die kleinste/größte unter den gezogenen Zahlen
- es werden zwei/drei aufeinander folgenden Zahlen gezogen (Zwillinge/Drillinge)

Programmieren Sie eine Serie von Münzwürfen und analysieren Sie die Anzahlen von WAPPEN und ZAHL.

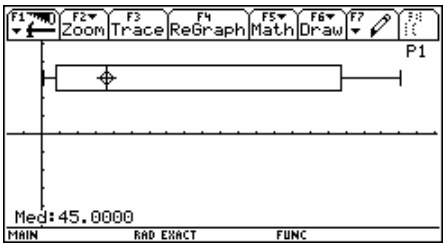
Programmieren Sie eine Serie von Würfen mit einem Würfel und analysieren Sie die Ergebnisse. Verallgemeinern Sie auf einen „Würfel mit *n* Seiten".

Simulieren Sie eine Binomialverteilung und erzeugen Sie eine passende graphische Darstellung: Auf dem Voyage 200 sehen die Dialogeingabe und die graphische Aufbereitung so aus:





Vielleicht gelingt Ihnen gar die Programmierung eines Kastendiagramms (siehe Bild vom Voyage 200 auf der nächsten Seite). Sie müssen den Abstand über der *x*-Achse und die Breite der Box unter die Programmparameter aufnehmen.



(Tipp: Erzeugen Sie die Figur als ein mit Strecken verbundenes Streudiagramm.)

Literaturhinweise

[1] *Computer Graphics with Derive*, Maria Koth, DNL#32 – 34, 1999

[2] *Elementary Linear Algebra with Derive*, Robert Hill u.a., Chartwell-Bratt, 1995

[3] *Mastering the TI-92*, Larry Gilligan u.a., Gilmar Publishing, 1996

[4] *Mathe Trainer I für TI-89/92/92+*, Josef Böhm, bk-teachware, 2000

[5] *Maximum- and Minimum Values*, Erich Zott, DNL#12, 1993

[6] *Optimierungsaufgaben grafisch, numerisch, ...*, Josef Böhm, bk-teachware, 1998

[7] *Optimization Problem with TI-Nspire CAS*, Bernhard Kutzler, DNL#68, 2007

[8] *Parabeln und Co. Erforschen mit TI-Nspire CAS*, Rainer Wonisch, bk-teachware, 2008

[9] *Programmieren in Derive*, Josef Böhm, bk-teachware, 2002

[10] *Reverse Discussion of a Curve*, Otto Reichel, DNL#15, 1994

[11] *Step Functions and Riemann Integrals*, Wolfgang Pröpper, DNL#66, 2007

[12] *The MedMed-Regression with Derive*, Josef Böhm, DNL#41, 2001

[13] *The Trigonometric Superbox*, Josef Böhm, DNL#23, 1996

[14] *TI-Nspire™CAS = Successor of Derive*, Bernhard Kutzler, DNL#67, 2007

[15] *TI-Nspire™CAS Referenzhandbuch*, Texas Instruments, 2008[*]

[16] *TI-NspireCAS Softwareguide*, Texas Instruments, 2008[*]

[17] *Vedic Mathematics using Derive*, Nurit Zehavi u.a., DNL#43, 2001

[18] *Vieta by Chance*, Jan Vermeylen, DNL#20, 1995

[19] *Einführung in TI-NspireCAS*, B Kutzler, `http://b.kutzler.com/src/download.htm`

[*] Zum Download von `http://education.ti.com`

Der DNL (Derive Newsletter) ist das Journal der *International Derive and TI-CAS Usergroup*. Informationen finden Sie auf http://www.austromath.at/dug

Index