Lektion 5: Listen, Graphiken und Dynamik | **Übung 2: Schleifen und Listen**

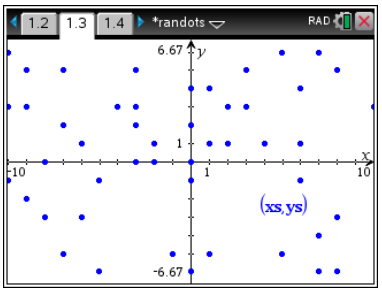
In dieser Übungseinheit wirst du lernen, wie man einzelne Listenelemente kontrolliert und wie man ein Programm aus einer Math Box in den Notes „dynamisieren“ kann.

Lernziele:

- Reelle Zufallszahlen (Dezimalzahlen) in einem gegebenen Intervall erzeugen
- Mit der „interaktiven“ Eigenschaft der Math Boxes in den Notes ein Programm „dynamisch“ machen

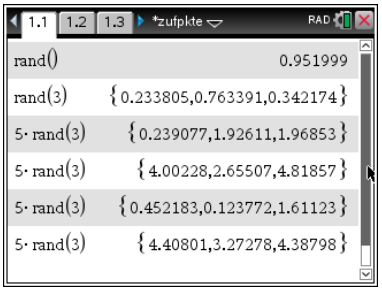
In der vorigen Übung…

… haben wir ein Programm geschrieben, das zwei Listen von Zufallszahlen erzeugt und diese in Form eines Streudiagramms auf einer Graphs-Seite dargestellt hat. Erinnere dich an die Ausgabe von **zufpkte(n)** wie rechts gezeigt.



Das Programm hat sich darauf beschränkt, nur ganzzahlige Werte für die x- und y-Koordinaten des Streudiagramms zu erzeugen. Daher lagen die Punkte auf den Gitterpunkten. Jetzt wollen wir das Programm auch auf nichtganzzahlige Werte erweitern, so dass die Punkte wesentlich dichter zu liegen kommen.

Anstelle von **randInt**(-10, 10) werden wir mit der **rand( )**-Funktion eine Dezimalzahl (zwischen 0 und 1) generieren und dies so skalieren, dass die Punkte wiederum innerhalb des Standardfensters liegen.



Sehen wir uns zuerst die Ausgaben von **rand( )** und **rand(k)** im *Calculator* an. Ohne Argument erzeugt **rand()** eine Dezimalzahl zwischen 0 und 1. Mit dem Argument 3 wird hingegen eine Liste von drei derartigen Dezimalzahlen ausgegeben. Diese Liste kann als Teil eines Ausdrucks dazu verwendet werden, dass die Werte die Grenzen [0, 1] überschreiten. So ergibt dann **5*rand(3)** eine Liste von reellen Zufallszahlen zwischen 0 und 5.

Das müssen wir wissen, um zufällige Dezimalzahlen zwischen -10 und +10 zu erzeugen. Das ist ein Intervall von 20 Einheiten. Wir beginnen mit -10 und addieren eine Zufallszahl zwischen 0 und 20 dazu, die wir mit **20*rand()** generieren.

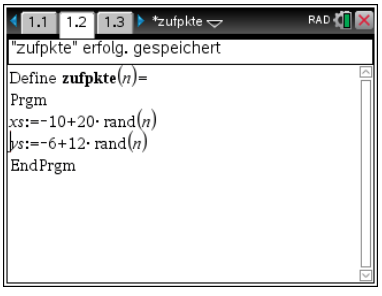
Hinweis: Mit **rand**( ) können wir eine dezimale Zufallszahl im Intervall [A, B] nach der folgenden Formel erzeugen:

n:= A+(B-A)***rand**().

Das hat also die Form n:= *Startwert* + *Intervallbreite* * **rand**()

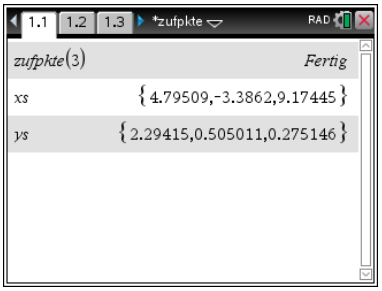
1. Wir ändern in unserem Programm **zufpkte(n)** die **randInt**-Funktionen in einen Ausdruck, der **rand()** enthält, und zwar folgendermaßen:

   xs := -10 + 20*rand(n)
   ys:= -6 + 12*rand(n)



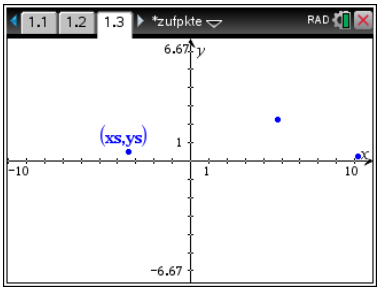
Diese Terme erzeugen *n* Werte zwischen -10 und 10 für **xs** und *n* Werte zwischen -6 und 6 für **ys**.

2. Lass das Programm im *Calculator* laufen und sieh dir die Werte von **xs** und **ys** an.

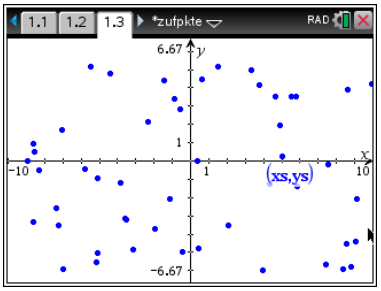


Hinweis: Wenn man mit einer neuen Funktion am TI-Nspire™ CX arbeitet, ist es hilfreich, die Funktion zuerst im *Calculator* zu studieren, bevor man sie in einem Programm einsetzt. Auch im Catalog wird Hilfe zu den notwendigen und optionalen Argumenten, bzw. zu den Ergebnissen der Funktion angeboten.

3. Dann gehe zum Streudiagramm von (**xs**,**ys**).



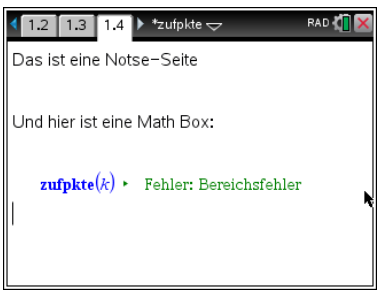
4. Starte das Programm nochmals mit einem größeren Argument *n*.
   - Rechts ist ein Screenshot nach der Durchführung von **zufpkte(50)** mit den Dezimalzahlen, die mit **rand()** erzeugt worden sind.



Hinweis: Der nächste Abschnitt führt einen weiteren Weg ein, wie man ein Programm über eine Math Box auf einer Notes-Seite ausführen kann. Diese Möglichkeit gibt es nur auf der TI-Nspire™ CX-Plattform. Damit kann man ein Programm laufen lassen, wann immer eine globale Variable oder ein Funktionsargument ihren Wert ändern. Die Änderung veranlasst dynamisch einen neuen Durchlauf des Programms, ohne, dass man es im *Calculator* neu aufruft.

Dynamische Programme

1. Um das Programm dynamisch zu gestalten, füge eine Notes-Seite ein.
2. Füge weiters eine Math Box in die Notes ein über ***menu> Einfügen> Math Box***.
3. Schreibe den Programmnamen mit einer Variablen (wir verwenden *k*) und drücke enter (oder rufe das Programm über die var-Taste auf).
   - Jetzt erscheint eine Fehlermeldung, da das Argument eine undefinierte Variable ist. Das werden wir gleich ändern.

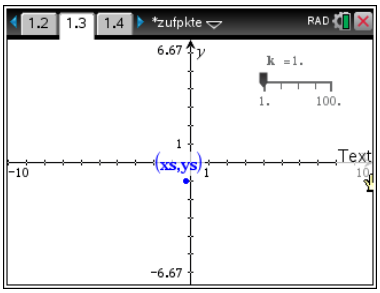


Der Parameter, den du hier nimmst, muss nicht der gleiche sein, wie der im Programmeditor verwendete. Der Wert wird dem Programmargument übergeben.

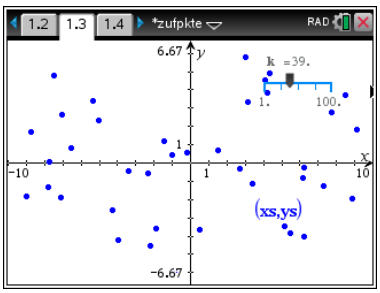
4. Installiere auf der Graphs-Seite (mit dem Streudiagramm) einen Schieberegler über ***menu> Aktionen> Schieberegler einfügen***
5. Nimm jene Variablenbezeichnung, die du in der Math Box genommen hast. Bei uns war das *k*.
6. Setze den Anfangswert auf 1, Minimum und Maximum auf 1 bzw. 100 und die Schrittweite auf 1, wie im rechten Bild angezeigt.
7. Drücke enter um den Schieberegler zu platzieren.
8. Bewege ihn auf eine passende Stelle und drücke enter oder klicke nochmals um ihn zu verschieben.

Du siehst, dass alle Punkte verschwunden sind, bis auf einen. Das ist das Resultat des Programms, das auf den neuen Wert für *k* (in unserem Demoprogramm) aus der Math Box auf der Notes-Seite reagiert.

Wenn du jetzt nochmals in die Notes schaust, wirst du merken, dass die Fehlermeldung verschwunden ist. Und an ihrer Stelle das Wort *Fertig* steht. Jetzt, wo das Argument *k* definiert ist, läuft das Programm ordnungsgemäß.



9. Bewege nun den Schieberegler für *k*. Jedes Mal, wenn der Wert für *k* geändert wird, wird das Programm erneut ausgeführt, neue Listen und damit neue Punkte werden generiert und schließlich auch im Streudiagramm dargestellt.



Hinweis: Wenn der Schieberegler das Argument um 1 erhöht, dann wird ein komplett neuer Satz von Punkten mit einem Punkt mehr geplottet. Es wäre eine ***Herausforderung***, das so zu programmieren, dass gerade nur ein zusätzlicher Punkt zum bestehenden Streudiagramm hinzugefügt wird, wenn der Schieberegler um 1 erhöht wird, und der zuletzt gezeichnete Punkt gelöscht wird, wenn der Schieberegler die Anzahl der Punkte um eins herabsetzt. Das ist eine interessante Aufgabe.