Vorlesung “Astronomische Datenanalyse”

Praktikum 1: IDL und einfache Datenauswertung

Version vom 30. Oktober 2003

Das Praktikum findet im Institut für Astronomie und Astrophysik, Abt. Astronomie im Gebäude Sand 1 statt.

Unter der URL `http://astro.uni-tuebingen.de/~wilms/teach/data/` finden Sie PS- und PDF-Dateien dieser Anleitung, die Sie sich ausdrucken können, sowie alle hier angegebenen Programme.

1 Einführung in IDL

Für die Durchführung des Praktikums in astronomischer Datenanalyse werden wir die Programmiersprache IDL (Interactive Data Language) von Research Systems, Inc., Boulder, CO, verwenden. Diese Programmiersprache wird in der Astronomie und Astrophysik häufig zur Datenanalyse eingesetzt. Im Gegensatz zu den meisten anderen populären Programmiersprachen (z.B. C, PASCAL, Fortran, usw.) ist IDL eine interpretierte Programmiersprache. Das bedeutet, daß Sie in IDL auch von der Kommandozeile aus direkt IDL-Befehle ausführen können. Das erleichtert die Fehlersuche und das Ausprobieren von Befehlen ungemein. Wie Sie in den späteren Praktika sehen werden verfügt IDL über mächtige und sehr schnelle Kommandos zur Behandlung großer Matrizen (Arrays), die sie auch für den Umgang mit "echten" Datenmengen geeignet macht.

Im folgenden soll Ihnen kurz IDL vorgestellt werden. Diese Einführung soll denjenigen von Ihnen, die schon einmal mit einer Programmiersprache gearbeitet haben, den Einstieg in IDL erleichtern. Sollten Sie noch nie mit einem Computer gearbeitet haben, werden Sie mit dem nächsten Unterkapitel sicherlich weniger anfangen können – wir werden Ihnen in diesem Fall eine "Einzeleinführung" geben.

Alle im folgenden aufgelisteten Programme können Sie unter

`http://astro.uni-tuebingen.de/~wilms/teach/data/`

auch herunterladen.

1.1 Aufruf von IDL: Der IDL Interpreter

Sollten Sie noch keinen Account am Astronomischen Institut besitzen, benutzen Sie bitte den Account `data`. Das Password werden wir Ihnen am ersten Praktikumstag bekanntgeben. Sie können IDL auf allen Unix-Maschinen des AIT benutzen, diese erreichen Sie unter X11 durch druck auf die rechte Maustaste. Zum Start von IDL geben Sie in der Shell bitte `idl` ein:

```
ait320:~> idl
```

Es erscheint die Programminformation für IDL und nach einigen Sekunden des Wartens, während denen IDL die am AIT installierten IDL Programmbibliotheken lädt, erscheint das IDL Prompt.

```
IDL Version 5.5a (linux x86). (c) 2001, Research Systems, Inc.
Installation number: XXXXX.
Licensed for use by: AIT TUEBINGEN

IDL>
```

Sie können jetzt mit IDL direkt kommunizieren. Wichtig für das gesamte Praktikum ist für Sie der Befehl, mit dem Sie die IDL-Onlinehilfe lesen können. Geben Sie dazu einfach ein Fragezeichen am Prompt ein:

```
IDL> ?
```

Nach einigen Sekunden erscheint ein neues Fenster, mit dem Sie den kompletten Text der IDL Handbücher lesen können. Besonders wichtig ist hierbei die Stichwortsuche, die Sie durch Klicken auf "Index" auf dem Hauptfenster der Hilfe erreichen können. Die Eingabe eines Suchworts (in Englisch...) führt hier häufig zum Erfolg. Im weiteren Verlauf dieses Praktikums werden Sie IDL-Routinen benötigen, die hier nicht ausführlich beschrieben sind. Die Dokumentation dieser Routinen können Sie über den Link "Alphabetical List of Routines" in Ruhe nachlesen. Weitere IDL-Unterprogramme, die Sie für die Lösung astrophysikalischer Probleme häufig benötigen, sind Teil der IDL Astronomy User's Library des Goddard Space Flight Center der NASA sowie die lokal am IAAT entwickelten Routinen der `aitlib`. Die Dokumentation dieser Routinen finden Sie auf den Webseiten des Instituts (`http://astro.uni-tuebingen.de/software/idl`).

1.2 Variable und Datentypen

Anders als in den meisten Programmiersprachen haben Variable in IDL keinen festen Typ sondern werden über eine Zuweisung deklariert. Zum Beispiel hat nach der Zuweisung

```
IDL> a=2
```

die Variable a den Wert 2. Hierbei ist a eine sogenannte integer-Variable, das heißt, sie enthält Zahlen ohne Nachkommastellen. Sie können den Typ einer Variablen interaktiv mit der help-Routine überprüfen. Nach der obigen Zuweisung beispielsweise liefert

```
IDL> help,a
```

die Ausgabe

```
A               INT       =        2
```

Nach der Eingabe von

```
IDL> a=2.0
```

hat a den Typ einer Fließkommazahl, also einer Zahl mit Nachkommastellen. Weitere Datentypen, die Sie benötigen werden, sind Fließkommazahlen hoher Genauigkeit ("double"), wie zum Beispiel

```
IDL> a=2.0D0
```

deklariert. Solche Zahlen haben ca. 16 Stellen Genauigkeit, während bei Fließkommazahlen einfacher Genauigkeit ca. 10 Stellen genau sind.

Beachten Sie bitte: wie in allen Programmiersprachen ist *nicht* die Zahl der eingegebenen Nachkommastellen dafür ausschlaggebend, ob eine Zahl einfache oder doppelte Genauigkeit hat! So ist nach

```
a=123.2352334623453474567234623456367458
```

trotz der großen Zahl der eingegebenen Nachkommastellen a dennoch nur vom Typ einfacher Genauigkeit. Überprüfen Sie dies bitte mit der help-Prozedur!

Wenn Sie sich den Wert einer Variablen ausgeben lassen möchten, dann benutzen Sie den Befehl print. Also zum Beispiel

```
IDL> print,a
```

Beachten Sie das Komma nach dem IDL-Befehl, eine (gräßliche) Eigenart von IDL!

IDL verfügt natürlich über alle Grundrechenarten und auch höhere Funktionen (bei trigonometrischen Funktionen ist das Argument im Bogenmaß einzugeben), beachten Sie den Namen der Logarithmusfunktion! Es gilt natürlich die Punkt- vor Strichrechnung:

```
IDL> a=3.141
IDL> b=1.4
IDL> c=3
IDL> print,a+b*c
      7.34100
IDL> print,sin(a),cos(a),tan(a)
  0.000592621    -1.000000 -0.000592621
IDL> print,sqrt(a)
      1.77229
IDL> print,alog(b)
     0.336472
IDL> print,a+(b-c*a)*a
     -22.0592
IDL> print,a^3.0
      30.9887
IDL> print,a^b
      4.96472
```

1.3 Einfache Programmierung

Bislang hatten wir IDL direkt vom Prompt benutzt. Auch wenn das für einfachste Aufgaben ausreichend ist, ist es häufig besser, ganze IDL Programme ablaufen zu lassen. Ein IDL Programm ist ein einfaches, mit einem beliebigen Editor geschriebenes ascii-File, das die Endung `.pro` trägt. Wenn Sie als Editor den `emacs` benutzen, sind Sie bei der Tübinger emacs-Installation automatisch im sogenannten `IDL-Mode`, bei dem die Syntax von IDL farblich gekennzeichnet wird. Das erleichtert die Programmierung von IDL ungemein. Ein Programm endet mit dem IDL-Kommando `END`, sie führen ein Programm vom IDL-Prompt mit dem Befehl `.RUN` gefolgt vom Programmnamen aus, also zum Beispiel

```
IDL> .run quadrat1
```

Beachten Sie den Punkt vor dem Kommando! Als einfaches Beispiel sollten Sie das folgende Programm zum Lösen einer quadratischen Gleichung in den Editor laden, das den Namen `quadrat1.pro` trägt:

```
;;
;; Einfache Loesung einer quadratischen Gleichung, Version 1
;;
;; J. Wilms, 1999/11/17
;;

;; Koeffizienten der quadratischen Gleichung
a=0.3
b=3.
c=5.

;; Loesungen
x1=(-b+sqrt(b^2.-4.*a*c))/(2.*a)
x2=(-b-sqrt(b^2.-4.*a*c))/(2.*a)

print,x1,x2

END
```

Alles, was in einem IDL-Programm nach einem Strichpunkt steht, wird von IDL nicht interpretiert. Der IDL-Modus des Emacs rückt mit zwei Strichpunkten stehende Kommentare automatisch ein, was zum Beispiel bei Schleifen und ähnlichem interessant ist (siehe unten). Beachten Sie ferner, daß jedes Programm Kommentare beinhalten sollte, was es eigentlich tut, von wem es geschrieben wurde, und wann es geschrieben wurde. Ferner ist eine gute Faustregel auch für längere Programme die sogenannte EVA-Regel, d.h. nach der *E*ingabe der Daten werden diese *v*erarbeitet und dann die Ergebnisse *a*usgegeben.

1.4 Einfache Programmstrukturen von IDL

Das im letzten Abschnitt vorgestellte Programm zur Lösung einer quadratischen Gleichung war nicht besonders benutzerfreundlich. So wird das Programm im Falle `a=0` mit einer Division durch Null sein Leben aushauchen. Ferner wird im Falle, daß die Diskriminante $b^2 - 4ac < 0$ ist, versucht werden, die Wurzel einer negativen Zahl zu ziehen, was ebenfalls keine gute Idee ist[1]. Es ist daher sinnvoll, diese Fälle vor der eigentlichen Rechnung abzufragen. Für solche Fälle gibt es in IDL die `IF`*Bedingung*`THEN` Konstruktionen, wie das folgende Beispiel verdeutlicht (Programm `quadrat2.pro`):

```
;;
;; Einfache Loesung einer quadratischen Gleichung, Version 2
;; (Bedingungen)
```

[1] Auch wenn IDL komplexe Zahlen erlaubt, liefern die Standardfunktionen wie z.B. die Quadratwurzel nur reellwertige Ergebnisse zurück.

```
;;
;; J. Wilms, 1999/11/17
;;

;; Koeffizienten der quadratischen Gleichung
a=1.0
b=3.
c=5.

IF (a EQ 0.) THEN BEGIN
    ;; schlecht, schlecht, schlecht
    print,'Gleichung ist keine quadratische Gleichung!'
    stop
END ELSE BEGIN
    ;; dieser Zweig wird ausgefuehrt, wenn a ungleich null ist
    print,'Alles in Butter!'
END

;; Diskriminante
disk=b^2.-4.*a*c

IF (disk LT 0.) THEN BEGIN
    print,'Quadratische Gleichung hat komplexe Loesung!'
    stop
END

;; Loesungen
x1=(-b+sqrt(disk))/(2.*a)
x2=(-b-sqrt(disk))/(2.*a)

print,x1,x2

END
```

Falls die Bedingung des `IF` Statements zutrifft, werden also die im nachfolgenden `BEGIN`...`END` Block folgenden IDL-Befehle ausgeführt, ansonsten der (optionale) `ELSE BEGIN`...`END` Block. Beachten Sie die Einrückung der Befehle, die das Verständnis des Programms vereinfacht! Im IDL-Mode des Emacs werden solche IDL-Strukturbefehle wie `IF`, `THEN` usw. farblich hervorgehoben, die Einrückung erfolgt im emacs automatisch und kann notfalls durch Drücken der Tabulator-Taste (mit "Tab" markiert, links oben auf der Tastatur) erzwungen werden. Die Instruktion `stop` hält das Programm an.

Die einzelnen Bedingungen für die `IF`-Statements sind Abkürzungen aus dem Englischen, z.B. `LE` für "Less then or equal", wichtig sind insbesondere:

$<$	$>$	$\geq$	$=$
LT	GT	GE	EQ

eine Aufzählung aller Vergleichsoperatoren finden Sie im Kapitel "Relational Operators" des IDL-Handbuchs, das Sie mit der Index-Suche der Online-Hilfe finden.

1.5 Unterroutinen und Funktionen

Die Programme `quadrat` und `quadrat2` sind zwar praktisch, weil sie eine quadratische Gleichung lösen können, allerdings ist es sehr unpraktikabel, vor der Lösung einer solchen Gleichung jedes Mal wieder den Programmtext editieren zu müssen. Daher gibt es in jeder Programmiersprache sogenannte Unterroutinen und Funktionen, in denen häufig wiederkehrende Routinen gelöst werden.

Am einfachsten läßt sich eine Unterroutine in IDL dadurch erklären, daß Sie ein einfaches Beispiel anschauen. Der folgende Programmtext `quadrat_solve.pro` enthält das Programm `quadrat2` als Unterroutine:

```
PRO quadrat_solve,a,b,c,x1=x1,x2=x2
  ;;
  ;; Einfache Loesung einer quadratischen Gleichung, Version 3
  ;; (unterroutinen)
  ;;
  ;; J. Wilms, 2002/05/12
  ;;
  ;; INPUT:
  ;;   a,b,c: Koeffizienten der quadratischen Gleichung ax^2+bx+c
  ;;
  ;; OUTPUT:
  ;;   x1,x2: Loesungen der quadratischen Gleichung
  ;;
  IF (a EQ 0.) THEN BEGIN
      ;; schlecht, schlecht, schlecht
      print,'Gleichung ist keine quadratische Gleichung!'
      stop
  END

  ;; Diskriminante
  disk=b^2.-4.*a*c

  IF (disk LT 0.) THEN BEGIN
      print,'Quadratische Gleichung hat komplexe Loesung!'
      stop
  END

  ;; Loesungen
  x1=(-b+sqrt(disk))/(2.*a)
  x2=(-b-sqrt(disk))/(2.*a)
END
```

Im Programm wird mit der Zeile

```
PRO quadrat_solve,a,b,c,x1=x1,x2=x2
```

die Prozedur `quadrat_solve` deklariert. Diese hat drei sogenannte “Positionsargumente”. Die Zuordnung dieser Positionsargumente zu den Variablen `a`, `b` und `c` erfolgt durch die Position nach dem Programmnamen. Zusätzlich hat die Prozedur noch zwei sogenannte “Keyword-Argumente”, mit den Namen `x1` und `x2`. In unserem Fall werden in diesen Keywörtern von der Routine die beiden Lösungen der quadratischen Gleichung zurückgeliefert. Klingt kompliziert? Ist es auf den ersten Blick auch... Was Keywörter sollen wird Ihnen am einfachsten durch die Benutzung von Unterroutinen mit vielen Parametern klar, wie wir sie im Laufe dieses Praktikums noch kennenlernen werden.

```
;;
;; Quadrat3: Lösung mehrerer quadratischen Gleichungen unter
;; Verwendung der Unterroutine quadrat_solve
;;
;; Joern Wilms, 2002/05/12
;;

a=3.0 & b=2.0 & c=-5.0 ;; (mehrere Anweisungen koennen durch & getrennt auch
```

```
                        ;; in einer Zeile stehen...)

;; Loesung der Gleichung -- Antwort in lsg1 und lsg2
quadrat_solve,a,b,c,x1=lsg1,x2=lsg2

print,'Loesung ist',lsg1,' und ',lsg2

;; Weiterer Fall
par1=3.
par2=24.
par3=-125.

quadrat_solve,x2=x2,x1=soln1,par1,par2,par3
print,'Loesung ist',soln1,' und ',x2


END
```

Das Programm löst zwei verschiedene quadratische Gleichungen und gibt jeweils die Nullstellen aus. Beachten Sie, daß der Name der Variablen, mit denen Sie `quadrat_solve` aufrufen, *nicht* mit dem Namen der jeweiligen Variablen in der Prozedur identisch sein muß! Am zweiten Aufruf von `quadrat_solve` sehen Sie auch einen der großen Vorteile der Keyword-Argumente: Im Gegensatz zu den Positionsargumenten ist ihre Reihenfolge nicht vorgeschrieben, da sie durch den Keyword-Namen schon eindeutig identifiziert sind.

1.6 Arrays und einfache Statistik

Um erste einfache Versuche der IDL-Programmierung zu lernen werden wir im folgenden Datensätze mit Hilfe eines Zufallszahlengenerators generieren und diese dann auswerten. IDL besitzt eine Reihe von Zufallszahlengeneratoren, die wir im ersten Schritt verwenden werden. Information über diese Generatoren erhalten Sie in der online-Hilfe durch Suche nach “random”, wir werden zunächst nur den Generator `randomu` verwenden. Um z.B. 100 gleichmäßig zwischen 0 und 1 verteilte Zufallszahlen zu erhalten, sagen sie

```
IDL> a=randomu(seed,100)
```

wobei Sie vorher die Variable `seed` auf einen von Ihnen zufällig gewählten ganzzahligen Wert gesetzt haben. Lesen Sie in der Dokumentation zu `randomu` nach, was das soll!

Nach dem Aufruf von `randomu` ist `a` ein sogenanntes Array, im obigen Fall ein Vektor mit 100 Elementen, die von 0 bis 99 durchnummeriert sind. Einzelne Elemente können Sie über eine index-Notation ansprechen. So gibt zum Beispiel

```
IDL> print,a[5]
```

das fünfte Element (Bruce Willis) von `a` zurück.

Was können wir nun mit solchen Arrays anstellen? Zum Beispiel können wir für die Datenauswertung statistische Kennzahlen ermitteln. Im nächsten Programmbeispiel zeigen wir Ihnen, wie Sie den Mittelwert der in `a` enthaltenen Zufallszahlen berechnen können (Programm `meanjw.pro`)[2]

```
;;
;; Berechnung des Mittelwerts von Zufallszahlen
;;

npt=100   ;; Zahl der Punkte
seed=1233 ;; Bitte variieren!
```

[2]Für IDL-Profis: natürlich geht das auch als Einzeiler, und es gibt natürlich auch für den Mittelwert eine IDL-Unterroutine…

```
data=randomu(npt) ;; npt Zahlen zwischen 0 und 1

sum=0. ;; Summe fuer den Mittelwert

;; Schleife ueber alle Elemente von data
FOR i=0,npt-1 DO BEGIN
    sum=sum+data[i]
END
mean=sum/npt

print,"Der Mittelwert ist ", mean

END
```

Erweitern Sie zunächst das oben gezeigte Programm zur Berechnung des Mittelwerts um die Berechnung der Standardabweichung und der Varianz. und des Medians.

Lesen Sie die Beschreibung der `randomu` Unterroutine. Sie können auch Poisson- oder normalverteilte Zufallszahlen erzeugen. Schreiben Sie Ihr Programm so um, daß Sie Normalverteilungen beliebigen Mittelwerts und Standardabweichung erzeugen können. Die IDL-Routine liefert Normalverteilungen mit Mittelwert 0 und Standardabweichung 1, diese Zahlen müssen Sie durch geeignete Addition des Mittelwerts und Multiplikation mit der Varianz transformieren um beliebige Normalverteilungen zu erreichen. Überprüfen Sie dies, indem Sie für die so erzeugten Zufallszahlen die Kennzahlen berechnen. Sie stellen fest, daß etwas anderes herauskommt? Stimmt. Unsere Beschreibung enthält nämlich einen Fehler... Korrigieren Sie diesen in Ihrem Programm!

1.7 Plotten und Häufigkeitsverteilungen

Besonders bei vielen Meßwerten ist es sinnvoll, diese zu visualisieren. Dazu ist IDL besonders geeignet. Einer der wichtigsten Befehle von IDL heißt `plot` und dient zur Darstellung von Daten auf dem Bildschirm.

Beispielsweise zeichnet Ihnen das folgende Programm die Sinusfunktion (Programm `sinus.pro`):

```
;;
;; Zeichnen der Sinus-Funktion
;; Joern Wilms, 1999/11/17
;;

npt=100 ;; 100 Punkte

;; findgen erzeugt ein Array, das alle ganzen Zahlen zwischen
;; 0 und npt-1 enthaelt. Die Variable !pi enthaelt (raten Sie!)
x=findgen(npt)/(npt-1) * 4.*!pi
y=sin(x)

;; Plotte die Funktion, Zeilen, die mit einem Dollarzeichen
;; beendet werden, werden in der naechsten Zeile fortgesetzt
plot,x,y,xtitle='x-Achsenbeschriftung', $
  ytitle='y-Achsenbeschriftung'

END
```

Wir wollen nun Häufigkeitsverteilungen der von uns erzeugten Zufallszahlen anschauen. Zur Berechnung einer solchen Häufigkeitsverteilung dient in IDL der Befehl `histogram`, der notorisch schwer zu verstehen ist. Lesen Sie bitte dennoch in der Online-Hilfe die zugehörige Beschreibung und versuchen Sie, mit Hilfe der Beschreibung und der Kommentare das folgende Programm zu verstehen (`histoequl.pro`):

```
;;
;; Histogramm gleichverteilter Zufallszahlen
;; Joern Wilms, 1999/11/17
;;

;; Zahl der Punkte
npt=100000

;; Kleinster x-Wert
xmin=0.

;; Groesster x-Wert
xmax=1.

;; Inkrement fuer ein Bin
dx=0.1

;; Erzeugung der Zufallszahlen (beachte: seed ist
;; NICHT gesetzt -- warum?)
data=randomu(seed,npt)

;; Berechnung des Histograms
histo=histogram(data,min=xmin,max=xmax,binsize=dx)

;; histo[i] enthaelt jetzt die Zahl der Datenpunkte in
;; data, die zwischen xmin+i*dx und xmin+(i+1)*dx liegen

;; Erzeugung der Bin-Werte
;; (die Funktion n_elements liefert die Zahl der Elemente in
;; ihrem Argument zurueck)
xx=xmin+dx*findgen(n_elements(histo))+0.5*dx

;; Plotten
plot,xx,histo,xtitle='x',ytitle='Zahl der Punkte'

END
```

Rufen Sie das Programm mit verschiedenen Werten von `dx` und `npt` auf, was passiert, wenn Sie `npt` immer größer machen?

Ändern Sie das Programm so ab, daß Sie anstelle der Zahl der Punkte die Wahrscheinlichkeitsdichte, d.h. den Anteil der zwischen x_i und x_{i+1} fallenden Punkte zeichnen.

Bauen Sie nun in Ihr Programm zur Berechnung von Mittelwerten etc. noch die Ermittelung des wahrscheinlichsten Werts der Verteilung ein. Hierfür müssen Sie sich überlegen, wie Sie das Maximum der Verteilung bestimmen – Lesen Sie dazu in der IDL-Onlinehilfe die Beschreibung der IDL-Funktion `max`.

Erweitern Sie das Programm jetzt so, daß Sie über das Histogramm der Häufigkeitswerte die theoretisch erwartete Wahrscheinlichkeitsdichte zeichnen. Der dazu notwendige IDL-Befehl heißt `oplot` (von "overplot").

1.8 Statistik von π

Die folgende IDL-Unterroutine liest aus der Datei `pi.txt` die ersten `ndigit` Dezimalstellen von π (`readpi.pro`):

```
PRO readpi,pi,ndigit=ndigit
   ;;
   ;; Unterroutine, liefert in pi die ersten ndigit Stellen von Pi
```

```
   ;; zurueck
   ;;

   IF (n_elements(ndigit) EQ 0) THEN ndigit=100000L
   pi=intarr(ndigit)

   openr,unit,'/home/wilms/publ/vorlesung/data/pi.txt',/get_lun
   readf,unit,pi
   free_lun,unit
END
```

Zum Beispiel enthält nach dem Aufruf

```
IDL> readpi,pi1,ndigit=100000
```

die Variable `pi1` die ersten 100000 Stellen von π.

Im folgenden werden wir die statistische Verteilung dieser Stellen untersuchen. Verwenden Sie dazu Ihr vorher geschriebenes Programm zur Darstellung von Wahrscheinlichkeitsdichten und plotten Sie sich die Verteilung der ersten 10, 100, 1000 und 500000 Dezimalstellen. Was fällt Ihnen auf? Berechnen Sie Mittelwert und Varianz dieser Häufigkeiten und vergleichen Sie diese mit dem Erwartungswert, daß alle Stellen gleich häufig auftreten. Weicht der Mittelwert signifikant vom erwarteten Wert ab?

1.9 Zufallszahlen selbst generieren

Wie funktioniert ein Zufallszahlengenerator? In der Vorlesung haben Sie die linear kongruenten Zufallszahlengeneratoren kennengelernt. Bei diesen wird die Zufallszahl x_{n+1} aus der vorherigen Zufallszahl x_n ermittelt nach dem Bildungsgesetz

$$x_{n+1} = (ax_n + c) \bmod m$$

wo a, c und m geeignet gewählte Zahlen sind. Solche Generatoren erzeugen ganze Zahlen, die dann in das Intervall von 0 bis 1 zu transformieren sind (Division durch m). Linear kongruente Generatoren sind nicht ohne Probleme, so daß die meisten heute verfügbaren Zufallszahlengeneratoren nicht vom Typ der kongruenten Generatoren sind, dennoch können sie gut zur Demonstration des Verhaltens solcher Generatoren verwendet werden.

Programmieren Sie jetzt einen Generator, wobei wir der Einfachheit halber $c = 0$ setzen. Um zu sehen, daß nicht alle Kombinationen von a und m gut sind wählen Sie $a = 29$ und $m = 97$. Generieren Sie jetzt 100 Zufallszahlen und plotten Sie diese gegeneinander.

Hierzu erzeugen wir uns in IDL zunächst ein leeres Plotfenster, bei dem die x- und y-Achse von 0 bis 1 geht:

```
IDL> plot,[0.,1.],[0.,1.],/nodata
```

Um den durch `x` und `y` erzeugten Punkt zu setzen verwenden Sie den `plots`-Befehl:

```
IDL> plots,x,y,psym=4
```

Hier wählt das Keyword `psym=4` ein bestimmtes Plotsymbol aus (Probieren Sie auch andere!). Plotten Sie jetzt die Paare (x_n, x_{n+1}).

Gute Generatoren erhalten Sie z.B. mit $m = 32749$ und $a = 162$ oder mit $a = 2^7 - 1$ und $m = 2^{31} - 1$. Modifizieren Sie Ihr Programm, daß es die letztere Kombination verwendet (Sehr große Integer-Zahlen müssen Sie in IDL durch ein Anhängen von `L` als "langwertige Integerzahl" kennzeichnen, also z.B. `m=2147483647L`).

Zum Test, ob die gewünschte Verteilung wirklich eine Gleichverteilung ist, müssen wir statistische Testverfahren heranziehen. Eines davon, das Sie in der Vorlesung schon besprochen haben, ist der χ^2-Test. Bilden Sie hierzu die Teststatistik

$$q = \sum_i^m \frac{(k_i - np_i)^2}{np_i}$$

wo k_i die Zahl der gemessenen Ereignisse vom Typ i ist, n die Gesamtzahl der Ereignisse und p_i die Wahrscheinlichkeit, daß das Ereignis vom Typ i eintritt. Insgesamt gibt es m verschiedene Ereignisse. In unserem Fall ist also k_i die Zahl der Zufallszahlen in Bin i, es gibt m verschiedene Bins und n ist die Gesamtzahl der Zufallszahlen.

Der Test gilt als bestanden mit Sicherheitswahrscheinlichkeit $1-\alpha$ wenn $q < \chi^2_{1-\alpha}(m-1)$ wo $\chi^2_{1-\alpha}(n)$ die χ^2-Verteilung ist. In IDL können Sie $\chi^2_{1-\alpha}(n)$ durch den Aufruf der Funktion `chisqr_cvf(alpha,n)` berechnen, so ist z.B. $\chi^2_{0.95}(5) = 11.07$.

Implementieren Sie bitte die Berechnung der χ^2-Summe in Ihr Programm und testen Sie den oben geschriebenen Zufallszahlengenerator und die in IDL enthaltenen Generatoren mit dem χ^2-Test.

1.10 Schnelleres Arbeiten mit Arrays

Eine wirkliche Stärke von IDL sind die sehr mächtigen Operationen beim Umgang mit Arrays. Aus pädagogischen Gründen hatten Sie oben beispielsweise den Mittelwert mit einer Schleife ausgerechnet:

```
sum=0.
FOR i=0,npt-1 DO BEGIN
    sum=sum+data[i]
END
mean=sum/npt
```

In anderen Programmiersprachen würden Sie einen Mittelwert wahrscheinlich ähnlich programmieren. IDL-Profis hingegen, die die eingebaute Mittelwertfunktion nicht verwenden wollen, schreiben

```
mean=total(data)/n_elements(data)
```

Die Funktion `total` berechnet die Summe aller Elemente in `data`, während die Funktion `n_elements(x)` die Zahl der Elemente in der Variablen `x` als Ergebnis hat. Insbesondere können Sie `n_elements` auch dazu benutzen, zu überprüfen, ob eine Variable überhaupt definiert ist. Mathematisch korrekt liefert `n_elements` dann nämlich `0` zurück.

Was ist, wenn Sie nur den Mittelwert eines Teils Ihrer Daten berechnen wollen? Sie benutzen die Erweiterung der oben angegebenen Array-Syntax:

```
b=a[5:10]
```

beispielsweise erzeugt ein neues Array `b`, das die Elemente 5 bis 10 von `a` enthält, während nach

```
c=a[12:*]
```

die Variable `c` alle Elemente von `a[12]` bis zum letzten Element von `a` enthält. Diese Zuweisung ist übrigens äquivalent zu

```
c=a[12:n_elements(a)-1]
```

Indizes für Arrays können auch wieder Arrays sein. Nach dem Programmstück enthält `d` die Elemente 3, 5 und 12 von `a`:

```
index = [3,5,12]   ;; index ist ein Integer-Array mit 3 Elementen
d=a[index]
```

Richtig schön ist es, solche Anweisungen an Bedingungen zu knüpfen. Dazu dient die `where`-Funktion, die ein Array auf all die Indizes eines Arrays zurückliefert, die eine Bedingung erfüllen. Im folgenden Beispiel erzeugen wir uns einige Zufallszahlen und suchen dann alle die Zufallszahlen heraus, die im Intervall zwischen 0.3 und 0.6 liegen:

```
data=randomu(seed,135)
ndx=where(data ge 0.3 and data le 0.6)
IF (ndx[0] ne -1) THEN intervaldata=data[ndx]
```

`where` liefert ein Array, das den Wert -1 enthält, zurück, wenn sie keine gültigen Arrayelemente gefunden hat.

Mithilfe von solchen Index-Zuweisungen können Sie nun auch einfach den Median einer statistischen Verteilung bestimmen. Für den Median müssen Sie Ihre Meßdaten der Größe nach sortieren (siehe die Beschreibung von `sort` in der IDL-Hilfe) und dann den Wert bestimmen, der in der Mitte der sortierten Liste liegt. Der Median von 5, 3, 6, 1 und 10 ist also 5. Enthält die Liste eine gerade Zahl von Meßwerten, dann ist der Median der Mittelwert der zwei mittleren Werte. Versuchen Sie nun, eine IDL-Funktion zu schreiben, die den Median berechnet. Um Ihre Ergebnisse zu überprüfen, vergleichen Sie sie mit dem Ergebnis der IDL-Routine `median`.