Lösung Aufgabe 2

- Es muss ein Weg gefunden werden, die Länge des längsten Laufs identischer Ziffern zu identifizieren:

```
maxLaenge<-1 ; aktuelleLaenge <- 1 ; old<-vek[1] #Startwerte
for (j in 2:20) {
  if (vek[j] == old) aktuelleLaenge <- aktuelleLaenge + 1
  else {
   if (aktuelleLaenge > maxLaenge) maxLaenge <- aktuelleLaenge
     old <- vek[j]
     aktuelleLaenge <- 1
    }
  }
```

- Hat man dieses Teilproblem gelöst, kann man die Struktur der Simulation von gestern direkt einsetzen.

```
results <- rep(NA , 10000)
for (i in 1:10000){
  vek <- sample(c(0,1), 20, replace=TRUE)
  maxLaenge<-1;  aktuelleLaenge <- 1;  old<-vek[1]
  for (j in 2:20) {
    if (vek[j] == old) aktuelleLaenge <- aktuelleLaenge + 1
    else {
    if (aktuelleLaenge > maxLaenge) maxLaenge <- aktuelleLaenge
      old <- vek[j]
      aktuelleLaenge <- 1
    } }
  results[i]<- maxLaenge
}
```

- In der Regel ist man nicht an solchen One-shot-Lösungen interessiert. Wenn möglich, versucht man die Lösungen so zu verallgemeinern, dass man das Programm in eine Funktion verwandelt, die dann flexibel bestimmte Parameter setzen kann, wie hier den Umfang der Stichprobe.

```
maxrun <- function(stichprobenlaenge){
  vek <- sample(c(0,1), stichprobenlaenge, replace=TRUE)
  maxLaenge<-1 ;   aktuelleLaenge <- 1 ;  old<-vek[1]
  for (j in 2:stichprobenlaenge) {
    if (vek[j] == old) aktuelleLaenge <- aktuelleLaenge + 1
    else {
    if (aktuelleLaenge > maxLaenge) maxLaenge <- aktuelleLaenge
    old <- vek[j] ;  aktuelleLaenge <- 1
    } }
  maxLaenge
  }
```

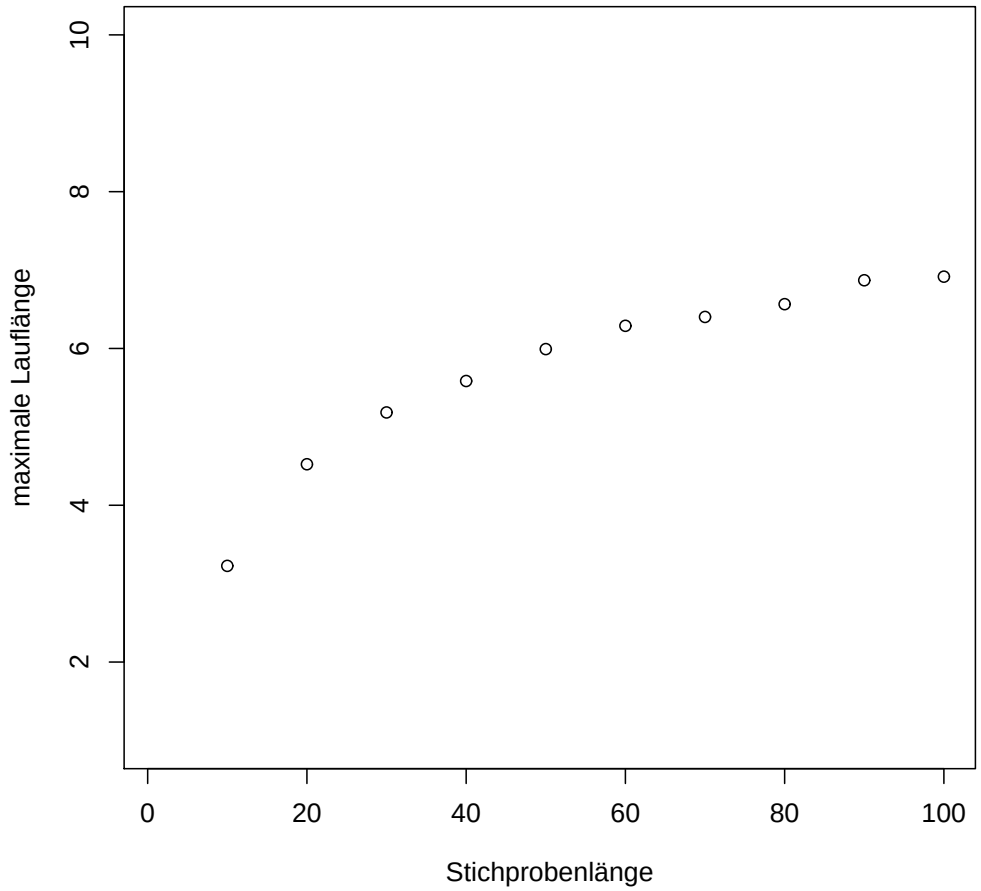
- Dann gehen so nette Dinge, wie die Veranschaulichung der Entwicklung des Durchschnitts der längsten Lauflänge bei wachsender Stichprobenanzahl.

- Schon für diese Aufgabe ist eine mathematischen Herleitung sehr schwer!

```
plot(1,xlim=c(1,100),ylim= c(1,10), t="n",
     xlab="Stichprobenlänge", ylab="maximale Lauflänge")

for (stichprobenlaenge in seq(10,100,10)) {
  results <- rep(NA , 1000)
  for (i in 1:1000){
    results[i] <- maxrun(stichprobenlaenge)
  }
  points(stichprobenlaenge, mean(results))
}
```

- Mit Ausgabe::


maximale Lauflänge
2
4
6
8
10
0
20
40
60
80
100
Stichprobenlänge

Zufall im Computer

- Experimente in unserem Sinne sind Computerexperimente.
- Problem: Computer sind per Definition deterministisch und sollen es auch sein! Wie bekomme ich zufällige Ergebnisse?
- Bevor Computer allgemein verfügbar waren, konnte man tatsächlich Tabellen mit Zufallszahlen kaufen. Die erste solche Tabelle erschien 1927 mit 40000 Zufallsziffern "zufällig" aus den Volkszählungsdaten gezogen (Tipett (1927)).
- Sogar in 2001 war in Bamberg/Bauer, Statistik, noch eine Seite mit Zufallsziffern.
- Es gab in der Zeit nach dem 2. Weltkrieg sogar spezialisierte Maschinen

zur Zufallszahlenerzeugung. 1955 wurde von der RAND Corporation eine Tabelle mit einer Million Zufallsziffern vermarktet.

- Der Nachteil natürlich: Extrem schlecht handzuhaben, wenig Zahlen!

- Eine (in der Regel aufwändige) Möglichkeit besteht darin, die bekannten echt zufälligen physikalischen Prozess zur Zufallsgewinnung zu nutzen und einen Computer mit einer Maschine zu koppeln, die diese Prozesse in Zufallszahlen verwandelt.

- Gibt es tatsächlch, teilweise sogar kommerziell!

- www.randomnumber.info oder random.org stellen Zufallszahlen auf Basis des Photonenspaltexperiments zur Verfügung.

- www.fourmilab.ch/hotbits stellt Zufallszahlen über Geigerzählermessungen zur Verfügung (sogar mit Sound!).

- Schon der legendäre Mark I Computer (1951) enthielt eine Schaltung, die jeweils 20 Zufallsbits in ein Register der Maschine schrieb, die über einen "widerstandsgesteuerten Rauschgenerator" erzeugt wurden. Übrigens auf direkte Empfehlung von Alan Turing!

- Erst moderne CPUs haben heute teilweise wieder Zufallsgeneratoren in Hardware eingebaut, die auf thermischen Schwankungen beruhen. Es werden mehrere überlagerte Wellen möglichst teilerfremder Frequenzen erzeugt und deren Amplituden dann "langsam" abgetastet. Die Schwingungsfrequenzen der Generatoren schwanken ständig minimal wegen unvermeidlicher Temperaturschwankungen, weshalb man so Zufallszahlen erhalten kann. Der VIA C3 "Nehemiah" war die erste CPU der neueren Zeit, die ein solches Device eingebaut hatte (2003).

- Eine kuriose Idee aus der Anfangszeit der Rechenmaschinen war, die letzten Nachkommastellen aufwändiger Multiplikationen als Zufallsziffern zu nutzen, da die mechanische Ungenauigkeit so groß war, dass man die letzten Stellen nicht vorhersagen konnte!

- Für die Verwendung in der Wissenschaft haben diese Zufallsquellen einige prinzipielle Probleme.

- Die exakten theoretischen Verteilungen der Zufallszahlen können oft nicht angegeben werden.

- Viel schlimmer: Versuche können nicht wiederholt werden! Das ist deutlich mehr Zufall, als man gebrauchen kann!

- Forschungsergebnisse sollten unbedingt reproduzierbar sein, auch wenn man stochastische Simulationen durchführt!

- Rein pragmatisch ist es fast unmöglich ein Programm zu debuggen, wenn man nicht an definierte Stellen im Ablauf springen kann.

- Konzeptionell verabschiedet man sich also von Konzept des “echten” Zufalls und gelangt zum Pseudozufall.

Pseudozufall

- Pseudozufall soll möglichst viele Eigenschaften von "richtigen" Zufallsgeneratoren bewahren, andererseits aber als deterministisches Programm implementierbar sein.

- Er muss also als eine Funktion programmierbar sein, die aus einer endlichen Anzahl von Parametern eine neue Zahl berechnen.

$$x_{n+1} = f(x_n, [x_{n-1}, \ldots], \textit{sonstige Parameter}).$$

- Das historisch älteste Beispiel für einen solchen Algorithmus stammt von John von Neumann (1946), einem der Väter des Computers.

- Sein Vorschlag war, eine große Integerzahl zu quadrieren, was die Anzahl der Stellen verdoppelt, und dann nur die jeweils mittleren Ziffern als nächste Zufallszahl zu behalten.

- Bsp: Ausgehend von 5772156649 gelangt man durch Quadrieren zu 33317792380594909201. Die nächste Zahl wäre dann 7923805949 usw.

- Stellt sich die Frage, inwiefern eine so deteriministisch erzeugte Zahlenfolge, bei der gilt

$$x_{n+1} = f(x_n)$$

  zufällig sein kann.

- Tatsächlich ist sie natürlich nicht zufällig. Durch den Startwert und die Abbildungsvorschrift ist die Folge vollständig determiniert!

- **Aber:** Für praktische Zwecke *erscheint* sie zufällig oder zumindest zufällig genug!

- Dieser konkrete Algorithmus von v. Neumann erwies sich als nicht besonders brauchbar.

- Es kann gezeigt werden, dass es zur Degeneration des Generators kommen kann, so dass kurze Zyklen auftreten.

- Frage: Können in einem Computer zyklenfreie Generatoren programmiert werden, die nur von einem Vorgängerwert abhängen?

- Für manche (große) Startwerte funktioniert das Verfahren aber oft gut.

- John von Neumanns ad hoc Ansatz entspricht nicht dem Wissen, dass heute vorhanden ist, um gute Generatoren zu programmieren.

Algorithmische Erzeugung gleichverteilter Zufallszahlen

- Da man, wie später gezeigt wird, aus gleichverteilten Zufallszahlen beliebig verteilte Zufallszahlen erzeugen kann, beschränken wir uns zunächst auf diese.

- Da Berechnungen mit ganzen Zahlen exakt in Computern durchzuführen sind, spezialisieren wir uns weiter auf die Erzeugung von gleichverteilten Zufallszahlen auf der Folge von ganzen Zahlen $0, \ldots, m$.

- Der Übergang zu stetig gleichverteilten Zufallszahlen gelingt dann durch einfache Division durch m.
  Sei X eine Zz gemäß der Gleichverteilung auf $0, \ldots, m$, so ist $U = X/m$ eine Zz (ungefähr) gemäß der Gleichverteilung auf [0,1].

- Haken: Das ist natürlich eine Idealisierung! X/m liegt auf einem Gitter mit Gitterweite $1/m$!

- Die Grenze m sollte deshalb hinreichend groß sein!

- Oft wird m als die größte in der jeweiligen Computersprache darstellbare, ganze Zahl gewählt.

Minimalexkurs: Der Restklassenkörper $\mathcal{Z}_m$.

- Sei m eine ganze Zahl. Dann ist $\mathcal{Z}_m$ die mathematische Struktur, wenn man alle Additionen und Multiplikationen der Zahlen $0, \ldots, m-1$ betrachtet und alle Ergebnisse modulo m berechnet.

- Bsp: 4*5 mod 6 = 2 oder 3+3 mod 6 = 0.

- Es existiert eine reichhaltige mathematische Theorie zu diesen sogenannten Restklassenkörpern.

- Hier ist nur interessant, dass man auf allen Rechenergebnissen eine modulo-Operation durchführt, um die Menge $0, \ldots, m-1$ nicht zu verlassen, sowie die Existenz inverser Elemente bzgl. Addition und Multiplikation in $\mathcal{Z}_m$.

Lineare Kongruenzgeneratoren

- Die heute gebräuchlichen (Pseudo-)Zufallsgeneratoren stammen alle von der Klasse der sogenannten linearen Kongruenzgeneratoren (LKG) ab.

- Ein linearer Kongruenzgenerator wird durch vier Zahlen vollständig bestimmt.

m, die größte Zahl bzw. der Restklassenkörper; $0 < m$,

a, ein Multiplikator; $0 \leq a < m$,

c, ein Inkrement; $0 \leq c < m$,

X_0, ein Startwert (oder seed); $0 \leq X_0 < m$.

- Der Generator erzeugt dann einen Strom von Zzen gemäß der Abbildung

$$X_{n+1} = (aX_n + c) \mod m, n > 0.$$

- Bsp: setzt man $m = 10$, $X_0 = a = c = 7$ dann ergeben sich die nächsten Werte zu

$$7, 6, 9, 0, 7, 6, 9, 0....$$

- Schon dieses Beispiel zeigt, dass nicht alle so erzeugten Folgen zufällig wirken!

- Stattdessen erkennt man hier, dass eine Periodenlänge des Generators von 4 vorliegt!

Aufgabe: Linearer Kongruenzgenerator in R

- Erstellen Sie (jetzt!) eine Funktion lnkong(), die aus m, a, X_n und c eine neue (Pseudo-)Zufallszahl gemäß obiger Beschreibung erzeugt.

- Überlegen Sie zunächst, wie ein einzelner Schritt in R formuliert werden müsste, verallgemeinern Sie dann zu einer Funktion!

Lösung:

```
lnkng <- function ( a, m, c, xn) { (a*xn + c )%% m}
```

Verallgemeinern Sie diese Funktion, so dass die Anzahl der zu berechnenden Zufallszahlen als Parameter übergeben werden kann!

Wahl der Parameter

- Wie gesehen, ist nicht jede Wahl der Parameter geschickt.
- Als entscheidend stellt sich die Kombination vom m und a heraus.
- Zunächst zur Wahl des Moduls m.
- Es ist leicht zu überlegen, dass selbst, wenn man nur 0 oder 1 wählen möchte, es nicht geschickt ist $m = 2$ zu setzen.
- Die zufälligste, mögliche Folge ist dann $0, 1, 0, 1....$ (Warum?)
- m sollte also groß gewählt werden!

- Ein Kriterium für die Auswahl ist die Geschwindigkeit, mit der die modulo Operation für das gewählte m auf dem speziellen Rechner durchzuführen ist, die sogenannte Wortlänge.

- Eine aus technischer Sicht geschickte Wahl ist $m = 2^w$, wenn w die Anzahl von bits in der Darstellung einer Integerzahl im gewählten Rechner ist.

- Additionen werden in einer solchen Maschine automatisch modulo m ausgeführt,

- bei Multiplikationen nutzt man einfach die hintere Hälfte des Ergebnisses.

- Wegen der bekannten Eigenschaften von $\mathcal{Z}_p$, wenn man m als Primzahl wählt, bietet es sich auch an, m als größte Primzahl kleiner als 2^w zu wählen!

- Generell sind Moduloberechnungen im Computer "billig" durchzuführen.

- Wenn schon Periodizitäten nicht vermeidbar sind, so ist es immerhin wünschenswert, wenn die Periodenlänge möglichst groß ist. Untersuchungen dazu führen zu Handreichungen, wie a geschickt zu gegebenen m zu wählen ist.

- Es kann folgendes Theorem bewiesen werden (Knuth, p. 17)
  Die linear-kongruente Sequenz, welche durch m, a, c und X_0 festgelegt wird, hat die maximale Periodenlänge m genau dann, wenn

  1. c teilerfremd zu m ist,
  2. $a - 1 = b$ eine Vielfaches jeder Primzahl p ist, die m teilt und
  3. b ein Vielfaches von 4 ist, wenn m ein Vielfaches von 4 ist.

- Untersuchungen der Periodizitäten lassen erkennen, dass $c = 0$ ohne Schaden gewählt werden kann.

Aufgabe 4:

Konstruieren Sie einige linearen Kongruenzgeneratoren (LKG) mit maximaler Periodenlänge. Nutzen Sie dazu die oben erstellte R Funktion und das vorstehende Theorem.