Abschlussprüfung Sommer 2020 - Entwicklung eines Softwaresystems

Familienname, Vorname: Cevin Voigt

Prüfungsnummer: 101 20514

Ausbildungsbetrieb: CAE Elektronik GmbH

Inhaltsverzeichnis

1 Änderungen zu Montag **3**

2 Verbale Beschreibung der realisierten Verfahren **3**
2.1 Zufallszahlen Generatoren 3
2.1.1 LCG 4
2.1.2 Polar Generator 4
2.1.3 Eigener Generator 5
2.2 Güte Funktionen 5
2.2.1 Serielle Autokorrelation 5
2.2.2 Sequenz Up-Down Test 5
2.2.3 Eigene Qualitätsfunktion 6

3 Auswertung und Diskussion der Gütefunktionen **6**
3.1 Linearer Kongruenz-Generator 6
3.2 Polar Methode/Generator 8
3.3 Eigener Generator 9
3.4 Auswertung 9

4 Zusammenfassung und Ausblick **10**
4.1 Zusammenfassung 10
4.2 Ausblick 10

1 Änderungen zu Montag

Die Programmatischen Änderungen zu Montag beinhaltet die Funktion der Güteklassen. Jede Güteklasse beinhaltet 2 Konstruktoren. Der erste Konstruktor bekommt die Parameter die der jeweilige Test benötigt zum Beispiel bei der Autokorrelation wäre die die Anzahl der zu generierenden Zufallszahlen und die der Abstand zur Nächsten Zufallszahl der betrachtet werden soll. Beim Zweiten Konstruktor handelt es sich um fast den selben wie der erste Konstruktor. Der Konstruktor bekommt zusätzlich zu den Parametern auch eine Ausgabe Klasse übergeben. Die Ausgabeklassen implementieren das eigens implementierte IOut Interface, was ein funktionales Interface ist. Diese IOut Interface hat nur eine Methode die ein Objekt übergeben bekommt und dies dann verarbeiten kann. Standard implementierte Ausgabe Klassen sind Console, FileWriter und NoOutput. Bei den Klassen handelt sich es um eine Konsolen Ausgabe, einer Datei Ausgabe und keiner Ausgabe. Die Implementierten Gütefunktionen beschreiben bei Übergabe die schritte die sie Gehen und das Ergebnis genauer. Zu den Verschiedenen Ausgabetypen und Klassen gibt es nun auch eine Table Klasse die dazu genutzt wird eine Tabelle aufzustellen. Diese Besitz lediglich die Funktionen Header hinzuzufügen und Werte passend zu den Headern. Die ToString Methode der Table Klasse gibt dann die Tabelle in Standard csv Schreibweise aus. Dies kann dann in einem FileWriter nützlich sein um eine csv Datei für die kommenden Tests zu erstellen. Die GeneratorFactory beinhaltet nun alle Beispiel LCG's und eine Methode zum generieren von LCG's auch jeder weiter vor-implementierte Generator muss über diese Factory Klasse Initialisiert werden. Die Konstruktoren der vor-implementierten Klassen sind jeweils Package private und können dadurch nur von der Factory erstellt werden.

Ein weitere Änderung zum Montag ist die eigene Güteklasse die im laufe dieses Dokumentes erklärt wird.

2 Verbale Beschreibung der realisierten Verfahren

2.1 Zufallszahlen Generatoren

Zu den realisierten Zufallsgeneratoren gehören der Linearer Kongruenz-Generator (LCG), Polar Generator und ein eigens Entwickelter Generator. Dafür habe ich ein Generator Interface bereitgestellt das alle Generatoren Implementieren müssen. Dieses besitzt Methoden zum Generieren von der nächsten Zufallszahl und zur Ausgabe des Intervalls der Generatoren. Eine Standardmethode zur Generierung von n Zufallszahlen befindet sich auch in dem Interface. Diese werde ich nun weiter erläutern.

2.1.1 LCG

Der Linearer Kongruenz-Generator, oder kurz LCG, benutzt ein Iteratives Verfahren um Zufallszahlen zu generieren. Dabei wird ein Multiplikator a, eine Verschiebung c und ein Modulator m benötigt. Der Multiplikator dient der Bestimmung der Steigung des Linearen Verfahrens und die Verschiebung auf der X Achse. Ohne den Modulator wäre dieses verfahren einfach gesagt eine Lineare Funktion. Es ist jedoch ein Iteratives verfahren was den letzten X Wert benutzt um den neuen X Wert auszurechnen. Der Modulator sorgt dafür das man eine gewisse Grenze hat. Die genannten X Werte sind die generierten Zufallszahlen. Jedoch gibt es eine gewisse Periodizität bei dieser Methode. Ab einen gewissen X Wert wird der Anfangswert wieder erreicht und man bekommt die gleiche Folge noch einmal. Aber die Zufallszahlen befinden sich im Bereich zwischen 0 und m. Die Funktion sieht wie folgt aus:

$x_{i+1} = (a * x_i + c) \, mod \, m$

Ein LCG besitzt auch eine sogenannte Periodizität. Diese wird berechnet indem man das erste Element nimmt und dann solange Zufallszahlen berechnen lässt bis die erste Zahl nochmal vorkommt.

Realisiert wurde das in dem Programm, indem man die Faktoren in dem Konstruktor bereits übergeben muss und die Methode der Klasse zur Generierung der Zufallszahlen das Mathematische Prinzip des LCG übernimmt und dann die neue Zufallszahl ausgibt.

2.1.2 Polar Generator

Der Polar Generator benutzt eine Koordinate u,v im Einheitskreis und nutzt diese um neue Zufallszahlen zu generieren. Die Koordinaten werden dabei bereits schon zufällig bestimmt und müssen zwischen -1 und 1 liegen. Mit dieser zufälligen Koordinate wird nun die Position q im Einheitskreis berechnet mit der Formel $q = u^2 + v^2$. Dann wird gecheckt ob q zwischen 0 und 1 liegt. Wenn dies nicht der Fall ist so wird eine neue Zufallskoordinate erstellt und ein neues q berechnet. Ist das q ordentlich gewählt berechnet man den Faktor p. Dieser sorgt dafür das aus den vorher gewählten unabhängig, gleich verteilten Zufallszahlen nun zwei neue voneinander unabhängige, standardnormalverteilte Zufallszahlen werden. Der Faktor p wird wie folgt berechnet: $p = \sqrt{\frac{-2*\ln(q)}{q}}$und dann jeweils auf das gegebene u und v drauf multipliziert. Dadurch entstehen nun 2 neue standardnormalverteilte Zufallszahlen.

Realisiert wurde das in dem Programm, indem man der Klasse im Konstruktor bereits einen Zufallsgenerator übergeben musste. Dieser wurde dann zur Berechnung der Zufallskoordinaten benutzt. Sollte der Generator der übergeben wird keinen Intervall von Minimum -1 und Maximum 1 aufweisen wird ein Fehler zurückgegeben. Dann wird in der vorgegebenen Methode zur Generierung der Zufallszahl der Algorithmus durchgeführt. Dafür nutze ich eine do-while schleife

um die Koordinaten zu bestimmen und danach die Standard Mathe-Bibliothek von Java um das p auszurechnen. Den zweite entstehende Wert wird abgespeichert und bei erneuter Generation der Zufallszahlen wird zuerst geguckt ob es bereits einen Wert gibt oder nicht. Der erste wert der Methode wird bereits zurückgegeben.

2.1.3 Eigener Generator

Der von mir selbst implementierte und realisierte Generator ist ein auf der Sinus Funktion und der aktuellen Zeit basierender Algorithmus. Man benutzt die aktuelle Zeit in Nanosekunden jedoch nur die letzten 3 Ziffern und Teilt diesen durch 360 und davon den Rest als Grad zahl an. Die Grad zahl muss dann in eine Bogenmaßzahl umgewandelt werden. Dazu wird die Formel $rad = \frac{deg*\pi}{180}$ verwendet. Die daraus resultierende Zahl wird dann in den absoluten Sinus eingesetzt. Dadurch erhält man im Intervall von 0 bis 1 zufällige Zahlen.

Realisiert wurde das in dem Programm, ohne das Parameter übergeben werden mussten. Dadurch das, das Prinzip der Methode auf der Aktuellen zeit funktioniert und der Sinus in der Mathe-Bibliothek vorhanden ist, ist eine Parameterübergabe nicht von Nöten.

2.2 Güte Funktionen

Die Güte Funktionen dienen der Qualitätsbeschreibung eines Generators. Die zu realisierenden Gütefunktionen waren die serielle Autokorrelation, der Sequenz Up-Down Test und eine eigene Güte Funktion.

2.2.1 Serielle Autokorrelation

Die serielle Autokorrelation beschreibt die Beziehung zweier Wertepaare im Vergleich zum Erwartungswert. Die Beziehung zweier Zahlen ist die sogenannte Korrelation und beschreibt ob es einen Zusammenhang zwischen diesen Zahlen gibt. Dabei wird besonders darauf geachtet ob es Muster zwischen den Wertepaaren gibt. Allgemein wird die Summe aller Multiplizierten Differenzen zwischen Erwartungswert und Wertepaar berechnet und dann als Zähler in der Division mit der Summe aller Zufallszahlen in der Differenz mit dem Erwartungswert betrachtet. Um so weiter der dabei entstehende Wert von der 0 abweicht desto weniger handelt es sich um eine Zufällige Verteilung. Die Formel mathematisch dargestellt sieht folgendermaßen aus: $p_k = \frac{\sum_{i=1}^{n-k}(x_i - \mu) * (x_{i+k} - \mu)}{\sum_{i=1}^{n-k}(x_i - \mu)^2}$. Wobei k der Abstand zwischen zwei Wertepaare ist.

2.2.2 Sequenz Up-Down Test

Der Sequenz Up-Down Test vergleicht die Reihenfolge der Zufallszahlen. Dabei wird zuerst eine Bitfolge aus Nullen und Einsen generiert die den unterschied 2er Nachbarn beschreibt. Um die Bitfolge zu erstellen gilt folgen-

des $\begin{cases} 1 & für\, x_i < x_{i+1} \\ 0 & für\, x_i > x_{i+1} \end{cases}$. Dadurch entsteht eine Liste mit aufeinander folgenden Nullen und Einsen. Diese Sequenz wird dann nochmals in eine kleinere Sequenz aufgeteilt indem man die Häufigkeit nebeneinander stehender Nullen und Eisen betrachtet. Als Beispiel einer Zufallszahlenkette der Länge 10 erhalten wir als Bitfolge 1,1,0,0,0,1,0,1,1. Die Daraus resultierende Häufigkeitssequenz sieht dann wie folgt aus: 2,3,1,1,2. Aus dieser Liste lassen sich dann die nötigen k's herausnehmen die benötigt werden für den späteren Qualitätsfaktor. In unserem Beispiel würde k = 1,2,3 sein und mit der Formel $N(k) = \frac{(k^2+3*k+1)*n-(k^3+3*k^2-k-4)}{\frac{(k+3)!}{2}}$ können dann die erwarteten Häufigkeiten berechnet werden. Die Qualität ist dann die Erwartete Häufigkeit minus die tatsächliche Häufigkeit. Das Ergebnis der Qualität kann man dementsprechend Deuten das umso geringer der Durchschnitts Entfernung zwischen gezählter und erwarteter Häufigkeit ist umso Höher ist die Qualität.

2.2.3 Eigene Qualitätsfunktion

Die von mir selbst implementierte und realisierte Qualitätsfunktion beschreibt den Abstand des Arithmetischen Mittels mit dem Erwartungswert. Das Arithmetische Mittel berechnet sich aus den gegeben n Zufallszahlen mit folgender Formel $\overline{x} = \frac{\sum_{i=1}^{n} x_i}{n}$. Der zurückgegebene Qualitätsfaktor gibt dann an wie hoch die unterschieden des Mittelwertes zu dem eigentlichen Erwartungswert sind. Um so größer das n gewählt wird umso kleiner sollte der Abstand dieser zum Erwartungswert sein.

3 Auswertung und Diskussion der Gütefunktionen

In diesem Abschnitt werde ich jede gegebenen Zufallszahlengenerator mit jeder vorhanden Gütefunktion Testen. Dabei werde ich sowohl eine kleine als auch eine hohe Anzahl an Generierten Zufallsvariablen Nutzen und auch viele Variationen Durchführen.

3.1 Linearer Kongruenz-Generator

Bei den LCG gibt es unendlich viele Möglichkeiten einen LCG aufzustellen. Ein guter LCG muss aber ein paar Kriterien Folgen. Die allgemeinen Kriterien für eine gute Folge lauteten:

1. $m > 0$
2. Multiplikator a, $0 \leq a < m$
3. Verschiebung c, $0 \leq c < m$
4. Startwert X_0, $0 \leq X_0 < m$

a	c	x0	Autokorrelation	Sequenz Up-Down	Eigene Qualitätsfunktion
1	4	5	-5.73E-04	12.50833333	0
2	3	7	1	Infinity	-2
7	4	0	0.249	57.25	1.5
9	7	9	0.989	57.25	-3.5

Tabelle 1: Ausgewählte eigene LCG

Name	Autokorrelation	Sequenz Up-Down	Eigene Qualitätsfunktion
Ansi-C	-3.934162525530557E-5	980.5321565104077	-130714.58854603767
Minimal Standard	3.946961585210038E-5	213.6657482114089	-35452.95018982887
RANDU	6.437755903578125E-5	4238.527762911732	-117881.36328101158
SIMSCRIPT	2.0249551855690006E-5	781.1936664137095	-91928.37710237503
NAG's LCG	0.4285933657796326	691.0461040713435	2.88225319436136448E17
Maple's LCG	0.4287608997611043	8937.141743645336	4.999958534339282E11

Tabelle 2: Qualität bei einhundert Millionen

Diese Werte können aber auch verschieden gewählt werden. Als Beispiel setzte ich m fest auf 10 und ändere die Werte a und c von 1 bis m-1 und den Startwert ändere ich jeweils auch von 0 bis n. Danach Teste ich jedes LCG und gebe den Gütegrad aller vorhandenen Gütefunktionen aus. Bei den Gütefunktionen werde ich das n der Generierten Zahlen auf 100 setzten und bei Autokorrelation und Sequenz Up-Down jeweils den Durchschnitt nehmen. Bei der Autokorrelation gehe ich wähle ich die k's von 1 bis 99. Ausgewählte Ergebnisse sind in dieser Tabelle raus geschrieben. Wenn eine Detailliertere Tabelle gebraucht wird ist eine nebenbei angelegt.

Wie man an diesem Auszug bereits erkennen kann sind einige gute Beispiele dabei. Betrachtet man den ersten Eintrag sieht man, das die Autokorrelation annähernd null ist oder sehr nah an null dran ist. Dies bedeutet das die Folge sehr zufällig wirkt. Dazu im Gegensatz der zweite Eintrag dieser besitzt eine Autokorrelation von 1 was weit entfernt von der null ist. Dies bedeutet das die Zahlen die entstehen keine Zufallszahlen sind. Der Sequenz Test und mein Eigener Qualitätstest sind von der Aussage her irrelevant da diese bei hohen zahlen erst Wirkung tragen. Dies sind zwar gute Veranschaulichungen aber dennoch kein guter Test um die Qualität von LCG zu Zeigen.

Um einen Guten Test präsentieren zu können betrachten wir die implementierten festen Beispiele bei 50'000'000 erstellten Zufallszahlen und deren Qualität.

In dieser Tabelle lässt sich bei 50'000'000 gut erkennen welcher von den obenstehenden ein guter Zufallszahlen Generator ist. Bei der Autokorrelation geht es darum so nah wie möglich an der 0 zu gelangen. Wie man unschwer erkennen kann sind bis auf NAG's LCG und Maple's LCG jeder bei 0.0000.... was sehr nah an 0 geht. Dadurch lässt sich deuten das diese Funktionen ein

sehr guter Zufallsgenerator ist. NAG's LCG und Maple's LCG jedoch liegen bei 0.4, was bedeutet das man diese zwar verwenden können aber nicht gerade die besten Zufallszahlen liefern. Wenn man nun den Sequenz Up-Down Test betrachtet, erkennt man die durchschnittliche Differenz vom erwarteten Wert und vom gezählten Wert wie in 2.2.2 erwähnt. Die Werte der einzelnen LCG's zeigen das sich die Werte nicht zu weit von den Erwarteten Werten Wegbewegt. Die Differenz der Einzelnen N(k)'s ist dabei entscheidend und nicht hoch. Daraus lässt sich schlussfolgern das die Reihenfolge auch dem Zufall gut überlassen ist.

Um die LCG's vollständig zu untersuchen muss man noch auf die Einzelnen Parameter (a,c,x0,m) eingehen und diese im Zusammenhang mit der Periodizität und der Qualität betrachten. Als ersten Test betrachte ich den Parameter m. Der Parameter m des LCG gibt die Modularität des Generators an. Dies bedeutet er gibt an was die höchste zu erreichende Zufallszahl des Generators ist des weiteren gilt als Regel dass, wenn wir nur a betrachten und kein Parameter c in Betracht nehmen darf a kein Teiler von m sein da sonnst eine unendlich lange Periode auftritt. Die Periodizität ist aber nicht nur von m abhängig auch die Wahl der Parameter a, c und x0 sind hier ausschlaggebend. Um das genauer zu prüfen werde ich eine Tabelle anlegen die alle Parameter bis 100 festsetzt dabei aber die regeln für ein gute Folge befolgt. Die Ergebnisse dieser Tabelle werde ich als Datei nebenbei stellen. Der Startwert des LCG's beeinflusst die Periodenlänge nicht. Nur die Parameter a, c und m haben Einfluss auf die Periodenlänge. Die Periodenlänge hat auch direkten Einfluss auf die Qualität der Funktion. Umso größer die Periodizität der Funktion ist umso höher ist die Qualität. Das liegt daran das eine Mustererkennung die richtig eingestellt ist schnell ein Muster in einer niedrigen Periodizität finden kann und damit die Qualität beeinflussen kann.

3.2 Polar Methode/Generator

Der Polar Generator beziehungsweise die Polar Methode dient dazu die 2 Zufallszahlen zu standardisieren. Um den Autokorrelationstest, den Sequenz Test und den Eigenen Qualitätstest durchzuführen werde ich diese mit 10^n Zufallszahlen testen. Bei der Autokorrelation nehme ich k = 1 bis 5. Ergebnisse dafür befinden sich in der neben gelegten Textdatei „polar_test_1.txt“. Der Generator der für die Polar Methode Verwendet wird ist ein Generator der die Standardimplementation von Java benutzt und diese auf -1 bis 1 begrenzt. Auch die Korrelationsergebnisse sprechen für sich. Die werte der Korrelation gehen immer näher zur 0 umso höher man das n wählt. Schaut man sich den Sequenz Test genauer an so kann man feststellen das bei kleinem n eine kleine Differenz der erwarteten Qualität und der für unkorrelierte Zufallszahlen. Umso höher mann das n jedoch wählt umso höher wird die Differenz zwischen erwarteten und Allgemeinen Wert. Dadurch lässt sich schlussfolgern das bei hohen n die Qualität der Zufallszahlen zunimmt. Der eigene Qualitätstest ergibt bei jeder Anzahl von getestetem n die gleiche Qualität zurück. Dies bedeutet das die Distanz vom Erwarteten Mittelwert und die des Tatsächlichen Mittelwertes eine

Große Distanz aufweist und dadurch zeigt das die werte die aus dem Generator rauskommen sehr stark voneinander abweichen.

3.3 Eigener Generator

Mein eigens implementierter Zufallszahlen Generator ist in der Lage jede Zahl zwischen 0 und 1 als Zufällige Zahl auszugeben. Dabei spielt die aktuelle Zeit in Nanosekunden eine Große Rolle. Um die Qualität zu Testen benutze ich die gleichen Voraussetzungen wie für den Polar Generator. Ich benutze für den Autokorrelationstest, den Sequenz Test und den Eigenen Qualitätstest eine Anzahl von 10^n Zufallszahlen und bei der Autokorrelation die k Werte von 1 bis 5. Die Ergebnisse dafür befinden sich in der daneben gelegten Textdatei „sinus_test_1.txt“. Die Ergebnisse der Autokorrelation kann man ähnlich wie die der Polarmethode sehen. Jeder der 10^n Tests ergibt einen wert nahe der 0 was bedeutet sie Korrelieren nicht miteinander und sind unabhängig, was eine gute zufällige Zahleneinfolge bedeutet. Auch der Sequenztest zeigt ähnliche Werte wie der des Polar Generators. Umso höher man das n wählt umso zufälliger scheint die Reihenfolge zu sein. Schlechter als bei der Polar Methode fällt der Eigene Qualitätstest aus. Dieser Zeigt das sich die Werte der Zufallszahlen im Gesamtbild nicht vom Erwartungswert des Generators unterscheidet.

3.4 Auswertung

Allgemein kann man sagen das sich jeder der implementierten Generatoren für eine Zufallszahlengenerierung eignen. jedoch gibt es einige Unterschiede in der Qualität der Generatoren. Betrachtet man den Polar Generator so stellt man fest das die Qualität der Zufallszahlen sehr gut ist aber von der genutzten Generator Funktion abhängig ist. Hat man eine Funktion mit einem Schlechten Qualität so wirkt sich dies auch auf die Qualität der Polar Methode aus. Auch der LCG ist abhängig davon wie die Parameter gesetzt werden. Dabei sind möglichst hohe Zahlen im Modulator sehr hilfreich um die Periodizität und auch die Qualität der Generatoren gut zu halten. Auch die Faktoren a und c spielen bei der Qualität eine große Rolle. Gerade wenn man qualitativ gute Zufallsgeneratoren möchte so ist es von Nöten bestimmte regeln einzuhalten um die Periodizität und damit auch die Qualität hoch zu halten. Wenn man nun zum schluss den Eigenen Generator (Sinus Generator) mal genauer anschaut erkennt man konsistente unterschiedliche Zahlenfolge mit keine oder geringer Muster Erkennung. Dies liegt an dem nutzen der aktuellen Zeit. Dadurch lassen sich auf kleinerer Anzahl von Zufallszahlen zwar kaum bis keine Muster erkennen jeoch bei höherem n und passender Prozessorgeschwindigkeit könnte es vorkommen das sich eine Reihenfolge oder sogar ein Muster sich erkennen lässt.

4 Zusammenfassung und Ausblick

4.1 Zusammenfassung

Jeder der implementierten Verfahren eignet sich gut zur Erstellung und Bereitstellung von Zufallszahlen. Durch die bereits vor-implementierten LCG's ist es möglich auch diese einfach zu benutzen und gute Zufallszahlen zu bekommen. Sollte man jedoch hohe Anzahlen von Zufallszahlen gebrauchen so ist die Funktion besser die keine Periodizität aufweist. Da wäre die Polarfunktion oder der Sinus Generator die bessere Wahl. Die Bibliothek fasst dies alles zusammen und stellt diese mit den Gütefunktoneinen zur Verfügung. Man kann diese Bibliothek einbinden und dann selbst Generatoren und Gütefunktionen erstellen aber auch die bereits erstellten nutzen um diese in einem Algorithmus einzubinden oder selbst geschriebenen Generatoren zu Testen. Die bereitgestellte Tabellen Klasse ist ein guter Helfer um schnell und einfach Tabellen für vergleiche zu erstellen. Des weiteren bindet sich auch in der Bibliothek Methoden zum schreiben von Dateien oder Konsolen ausgaben. Dies wird in den vor-implementierten Gütefunktionen genutzt um die schritte die getätigt werden zu verdeutlichen.

4.2 Ausblick

Eine Bibliothek bietet viele Möglichkeiten um sie zu verwenden oder zu verbessern. Unter anderem könnte man die Bibliothek visuell darstellen, indem man die verschiedensten Generatoren und ihre Zufallszahlen Graphisch in Diagrammen zeigt und auch die verschiedenen Gütefunktionen mit ihren einzelnen schritten in Diagramm Form zeigen kann. Zusätzlich könnte man ein LCG Test Generator Oberfläche hinzufügen die einen beliebigen LCG erstellen lässt und diese auf Qualität und Periodizität testet und Visuell darstellen. Des weiteren wäre um Performance zu verbessern eine Parallelisierung der Gütefunktionen angebracht. Da die Gütetests erst bei einem sehr hohen n eine gute aussage treffen können ist eine Parallelisierung um die Geschwindigkeit zu erhöhen angebracht. Beispielsweise die Autokorrelation beinhaltet in der Mathematischen Formel die Berechnung zweier Summen die jeweils bis n-k gehen. Summen der Mathematik lassen sich leicht Parallelisieren. Eine Weitere Möglichkeit wäre das erweitern der Bibliothek mit Stochatischen Modellen die die Zufallsgeneratoren nutzen können. Beispielsweise ein Urnen Modell mit und Ohne Zurücklegen, Münzwurf oder Würfelwurf könnte man als zusätzliche Funktionen in die Bibliothek einbinden.