VORLESUNG

Automatisierung industrieller Workflows

Teil C: Die Sprache SLX

- Einführung und Grundkonzepte -

Joachim Fischer

Position (letzte Vorlesung)

- **Teil A**
  Aspekte
  dynamis
- **Teil B**
  Die Moc
- **Teil C**
  Die ausf
  SLX
- **Teil D**
  Modellie

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  Vertiefung der SL
- **C.4**
  GPSS-Elemente
- **C.5**
  DISCO-Elemente
- **C.6**
  Basissprache (Erg

1. SLX-Überblick
2. Syntaxkonventionen
3. Elementare Datentypen
4. Klassen
5. Objekte
6. Typ Set
7. Anweisungen, Prozeduren
8. Einfache Ausgabe
9. Modellierungselemente in SLX (allgemein)
10. Prozesse und Pucks
11. Beispiel (Drucker, Druckjobs)
12. SLX- Laufzeitsystem (Simulator-Kern)
13. Zeitkonzepte
14. Scheduling-Operationen
15. Innere Parallelität: Child-Pucks per fork
16. Prioritäten
17. Zusammenfassung

Laufzeitvergleich bei Variation der Job-Queues

- Simulation für kleine und große Warteschlangenlängen

Konzept= **Set**	Kleine Länge {1}	"Mittlere" Länge	Große Länge
Zwischenankunft	[10, 20] gleichv.	[1, 14] gleichv.	[1, 10] gleichv.
gedruckte Jobs	35 040 320	700 851	95 498
durchschnittl.WZ	10.09 s	322 s [max #2.056	6 422 372 s [max# 27.610]
Ausführungszeit	4.38 s	0,26 s	0,05 s

Konzept= **Control**	Kleine Länge	Kleine Länge	Große Länge
Zwischenankunft	[10, 20] gleichv.	[1, 14] gleichv.	[1, 10] gleichv.
gedruckte Jobs	35 040 320	700 851	95 498
durchschnittl.WZ	10.09 s	5 058 949 s [max #2.056	6 422 372s [max# 27.610
Ausführungszeit	6.3 s	5.27 min	82s

Konzept= **List**	Kleine Länge	Kleine Länge	Große Länge
Zwischenankunft	[10, 20] gleichv.	[1, 14] gleichv.	[1, 10] gleichv.
gedruckte Jobs	35 040 320	700 851	95 498
durchschnittl.WZ	475.08 s	5 058 949 s [max #2.056]	6 422 372s [max# 27.610]
Ausführungszeit	3.1 s	4.63 min	45s

Warteschlangen-Implementierungsvarianten

- **Control-Variablen**
  sollten **nicht** für Warteschlangen benutzt werden
  (Konzeptmißbrauch)

- **List-Variante:**
  Wenn FIFO oder dynamische Prioritätsvergabe zum Einsatz kommt

  wait list= … und **reactivate list**=… garantieren FIFO
  (bei Prioritätsgleichheit)

  sonst nach Prioritätsklassen in LMP

- **Set-Variante:**
  - übliches Konzept,
    aber kein Pointer möglich
  - muss Attribut einer Klasse
    werden

```
type job_queue set(PrinterJob);
job_queue jq;
pointer (job_queue) Ptr_jq;
•• Semantic error: an object class name is required here;
•• "job_queue" is a set(*)

class PrinterJob ( in int in_job_num ) {
    int job_number;
    pointer ( puck ) my_puck;
```

```
pointer (JobQueue) jq;

passive class JobQueue {
    set(PrinterJob) q;
};

class PrinterJob ( in int in_job_num ) {
    int job_number;
    pointer ( puck ) my_puck;
```

Inhalt C.2

- **Teil A**
  Aspekte
  dynamis
- **Teil B**
  Die Moc
- **Teil C**
  Die ausf
  SLX
- **Teil D**
  Modellie

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  Vertiefung der SL
- **C.4**
  GPSS-Elemente
- **C.5**
  DISCO-Elemente
- **C.6**
  Basissprache (Erg

1. Charakterisierung von Zufallsgrößen
2. Pseudozufallszahlen als Approximation von Zufallszahlen
3. Die Klasse rn_stream
   SLX-Basisgenerator zur Erzeugung von [0,1)-verteilten Pseudozufallszahlen
4. Einstellung von Startwerten
5. Antithetische (0,1)-verteilte Zufallszahlen
6. SLX- Zufallszahlen verschiedener Verteilungsfunktionen
7. Mathematischer Zusammenhang der Verteilungsfunktionen
8. Behandlung empirischer Verteilungsfunktionen in SLX
9. Beispiel: PrinterJob

Rolle von Wahrscheinlichkeit u. Statistik in der Simulation

- Einflüsse der Systemumgebung oder Systemabläufe selbst unterliegen häufig dem Zufall

- Simulationsergebnisse sind dann als Stichprobe eines statistischen Experiments anzusehen

- Auf der Grundlage vieler Stichproben (Stichprobenraum) können
  - statistische Kennwertprofile und
  - Konfidenzaussagen (Aussagen zur Zuverlässigkeit der Ergebnisse)

  abgeleitet werden

- **wichtig**: der Einfluss des Zufalls muss im Simulationsexperiment wiederholbar dargestellt werden können:

  ~ Test von Simulationsmodellen,

Wiederholbarkeit eines Experiments durch Dritte

~ Prinzip guter wissenschaftlicher Praxis

Zufallszahlen im Original und Modell

1. Realität ←→ Modell

reale Zufallsgrößen werden durch mathematische Modelle (Funktionen) approximiert:

- *Verteilungsfunktion, Dichtefunktion, statistische Kenngrößen*

2. Modell ← → Modell

es bestehen mathematische Zusammenhänge zwischen einzelnen Verteilungsfunktionen:

- *beliebige Verteilungsfunktionen lassen sich durch (0,1)-gleichverteilte Verteilungsfunktionen approximieren*

3. Modell ← → Berechnungsmodell

determiniert berechnete Folgen von (0,1)-Werten lassen sich als Approximationen
von Verteilungsfunktionen realer Zufallsgrößen verwenden

Zufallsgrößen

Zufallsgrößen:= zufällige Ereignisse --> Zahlen

reale Zufallsgrößen und ihre Verteilungsfunktionen

Diskrete Zufallsgrößen:= Größen, die endliche oder abzählbar-unendlich viele verschiedene Werte annehmen können

Beispiel:

- Auszählen der Stillstände einer Maschine während einer Werksschicht
- Registrierung der Anzahl von Gesprächen in einer Telefonvermittlung

Stetige Zufallsgrößen:= Größen, die jeden beliebigen Wert innerhalb eines Intervalls der Zahlengerade annehmen können

Beispiel:

- Durchmesser von Antriebswellen (nach Bearbeitung an einem Drehautomaten): alle Werte innerhalb eines vorgeschriebenen Toleranzbereiches

Verteilungsfunktion einer Zufallsgröße

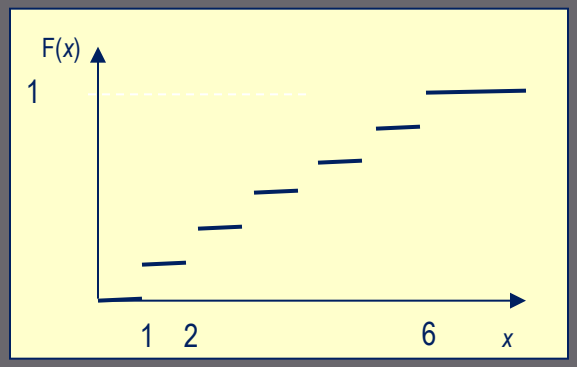
Charakterisierung einer Zufallsgröße X

- X nimmt bei jedem Versuch zufällig einen bestimmten Wert an
- Werte genügen einer Verteilungsfunktion

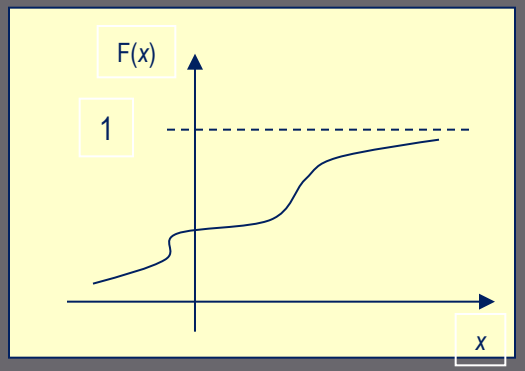
Verteilungsfunktion:= $F_X(x) = P(X \leq x)$,
der Wert von F_X (Verteilungsfunktion der Größe X) ist an der Stelle x ist **gleich der Wahrscheinlichkeit**, dass X einen Wert unterhalb von *x* annimmt.

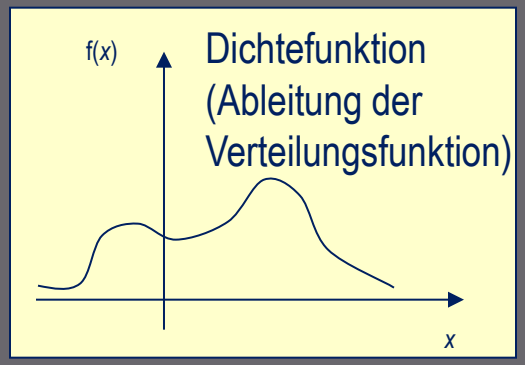
x durchläuft alle Werte der reellen Zahlengerade

diskret



kontinuierlich





Verteilungsfunktion als kumulative Dichtefunktion

Diskrete Zufallsgrößen

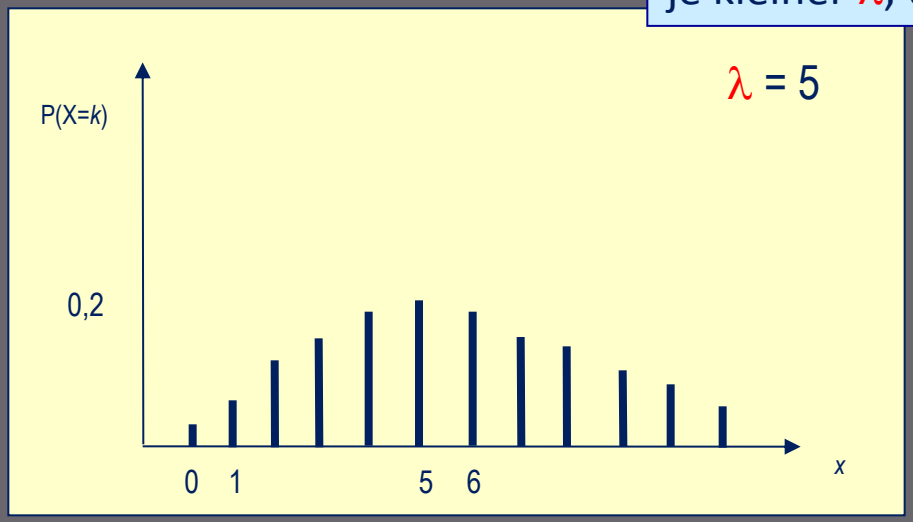
Beispiel einer diskreten Zufallsgröße X:

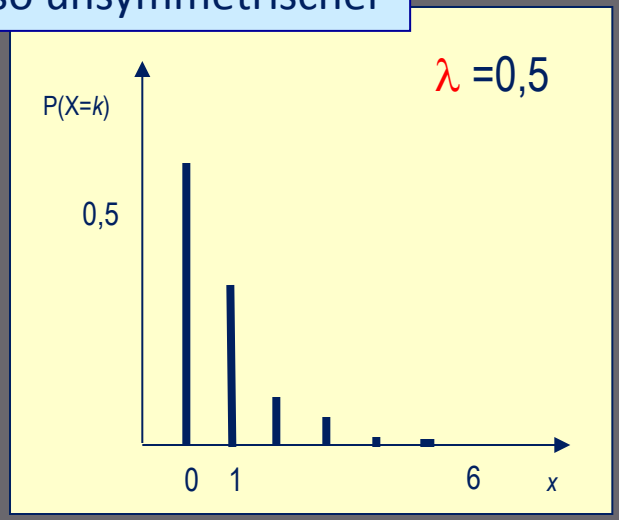
- X beschreibt Ereignisregistrierungen (Anzahl) in einem bestimmten Zeitbereich (X nimmt bei jedem Versuch zufällig einen best. Wert an)
- Werte genügen einem gleichen Typ von Verteilungsfunktion (**Poisson**)
- λ ist die mittlere Anzahl

Ereignisse

- Anzahl beobachteter Sternschnuppen in einer bestimmter Zeitspanne
- Anzahl registrierter Telefonanrufe in einer best. Zeitspanne
- Anzahl von Ankünften von Kunden einer Service-Einrichtung in best. Zeitspanne

je kleiner λ, umso unsymmetrischer





Verteilungsfunktion einer diskreten Zufallsgrößen

Poisson-Verteilung:

beschreibt Ereignisregistrierungen (Anzahl)
in einem bestimmten Zeitbereich

Verteilungsfunktion:= $F_X(x) = \sum_{k<x} P(X=k) = \sum_{k<x} \lambda^k/k! * e^{-\lambda}$,

falls x>0, sonst 0

$\mu = \lambda$, Erwartungswert
$\sigma^2 = \lambda$, Streuung

bedeutsam: Poisson-Verteilung steht im Zusammenhang mit der Exponentialverteilung:

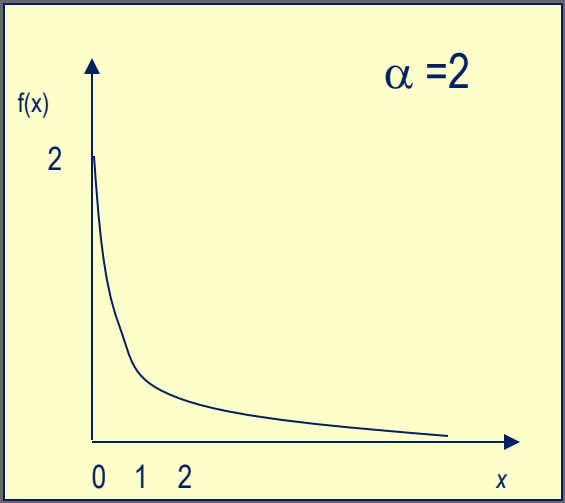
Sei X Poisson-verteilte Zufallsgröße mit Erwartungswert λ,
dann ist die **Zwischenankunftszeit** zweier aufeinanderfolgender Ereignisse
exponential-verteilt mit Erwartungswert $1/\lambda$

Stetige Zufallsgrößen

Beispiel einer stetigen Zufallsgröße X:

- X nimmt bei jedem Versuch zufällig einen bestimmten Wert an
- Werte genügen einer (negativen) Exponential-Verteilungsfunktion

Dichtefunktion	$f(x) = \alpha e^{-\alpha x}$,	falls x>=0, sonst 0
Verteilungsfunktion	$F(x) = 1 - e^{-\alpha x}$,	falls x>=0, sonst 0



- Zeitabstände zwischen Ankunftsereignissen von Ringstapeln in einer Schicht
- Dauer von Telefongesprächen
- Lebensdauer von Lebewesen, Maschinen, ...

$\mu = 1/\alpha$
$\sigma^2 = 1/\alpha^2$

Inhalt C.2

- **Teil A**
  Aspekte
  dynamis
- **Teil B**
  Die Moc
- **Teil C**
  Die ausf
  SLX
- **Teil D**
  Modellie

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  Vertiefung der SL
- **C.4**
  GPSS-Elemente
- **C.5**
  DISCO-Elemente
- **C.6**
  Basissprache (Ergänzung)

1. Charakterisierung von Zufallsgrößen
2. Pseudozufallszahlen als Approximation von Zufallszahlen
3. Die Klasse rn_stream
   SLX-Basisgenerator zur Erzeugung von [0,1)-verteilten Pseudozufallszahlen
4. Einstellung von Startwerten
5. Antithetische (0,1)-verteilte Zufallszahlen
6. SLX- Zufallszahlen verschiedener Verteilungsfunktionen
7. Mathematischer Zusammenhang der Verteilungsfunktionen
8. Behandlung empirischer Verteilungsfunktionen in SLX
9. Beispiel: PrinterJob

Pseudozufallszahlen

… als Approximationen echter Zufallsgrößen

Iterationsverfahren

x_0 – beliebig aus [1, 10], z.B.: 2

$x_{k+1} \equiv 2 * x_k$ **mod** 11 (k= 0, 1, 2, …)

➔ Zahlenfolge mit Periode p:
x_0 = 4, 8, 5, 10, 9, 7, 3, 6, 1, 2, 4, … (Wiederholung)

in Abhängigkeit vom Startwert

- ergibt sich eine periodische Folge von determinierten Zahlen, die als Zufallszahlenfolge interpretiert werden kann

Warum sind reproduzierbare Zufallszahlen enorm wichtig für die Simulation?

Ein einfacher Pseudozufallszahlengenerator

```
passive class  Random {
        int u;
        initial( int uu) {
                u= uu; if u < 1 or u>10 { error(…);  …};
        }
        procedure next () returning double {
                u:= 2*u;
                if u>11 { u= u-11; };
                return
                        double(u) / 11.0;
        }
}
```

Warum sind individuelle Startwerte
der Generatoren wichtig für die Simulation?

Individuelle Startwerte: eigene Folgen von Zufallszahlen

```
pointer (Random) s1, s2, s3;
s1= new Random(2);  s2= new Random(9);  s3 = new Random(7);
```

```
s1->next();  // .636, .273, .545, …
s2->next();  // .364, .727, .455, …
s3->next();  // .273, .545, .091, …
```

Linearer Generator

Generator für gleichverteilte 26-Bit-Zufallswerte

$q = 67.099.547 = 2^{26} - 1$

Kongruenzmethode

- multiplikative Variante (Iterationsverfahren mit Startwert x_0)

$$x_{j+1} \equiv k\, x_j \ \textbf{mod}\ q$$

$$x_{j+1} \equiv 8192\, x_j \ \textbf{mod}\ 67.099.547 \quad (j = 0, 1, 2, 3, \ldots)$$

➔ Zahlenfolge mit **sehr großer** Periode $p = 67.099.546$:

$$x_0,\ x_1,\ x_2, \ldots,\ x_{p-1},\ x_p = x_0,\ x_1,\ x_2, \ldots$$

Algorithmus für Startwerte (unabhängiger) Generatoren

$u_0 = 907$ (erster Startwert eines Generators)

$$u_{k+1} \equiv 36.855^* u_k \ \textbf{mod}\ 67.099.547$$

liefert für 500 Generatoren unabhängige Zahlenfolgen der Länge 120.000
(ohne dass eine Folge, in die andere „hineinläuft")

Inhalt C.2

- **Teil A**
  Aspekte
  dynamis
- **Teil B**
  Die Moc
- **Teil C**
  Die ausf
  SLX
- **Teil D**
  Modellie

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  Vertiefung der SL
- **C.4**
  GPSS-Elemente
- **C.5**
  DISCO-Elemente
- **C.6**
  Basissprache (Ergänzung)

1. Charakterisierung von Zufallsgrößen
2. Pseudozufallszahlen als Approximation von Zufallszahlen
3. Die Klasse **rn_stream**
   SLX-Basisgenerator zur Erzeugung von [0,1)-verteilten Pseudozufallszahlen
4. Einstellung von Startwerten
5. Antithetische (0,1)-verteilte Zufallszahlen
6. SLX- Zufallszahlen verschiedener Verteilungsfunktionen
7. Mathematischer Zusammenhang der Verteilungsfunktionen
8. Behandlung empirischer Verteilungsfunktionen in SLX
9. Beispiel: PrinterJob

SLX-Generatortyp

SLX verwendet (multiplikativen) Lehmer-Generator:

$x_{n+1} = c\, x_n \ (\mathbf{mod}\ M)$ — c, x_i ganze Zahlen aus [0, M)

$r_n = x_n / M$ — r_i aus Intervall [0,1)

$M = 2^{31} - 1$

$c = 742\,938\,285$

~ implementiert in der Klasse rn_stream

SLX-Klasse rn_stream

- vordefinierte passive Klasse
  liefert Basiszufallszahlenfolge aus [0.0, 1.0)

{passive} rn_stream		
int	seed	//aktueller Wert des Generators
int	start	//Startposition des Generators
int	count	//Anzahl bisher gelieferter Zufallszahlen
boolean	antithetic_switch	//Generatortyp
histogram	histo	// Häufigkeitsklassen [0..15]
string(*)	title	//Titel für Reportausgabe

- bei der Erzeugung lassen sich Startwert, Titel und Typ festlegen

rn_stream object_ident [**seed**= start_value] [**antithetic**] [**title**= string]

- alle erzeugten Generatoren werden in einer spezifischen internen SLX-Liste
  erfasst (automatische Erzeugung von Report-Ausgaben usw.)

Beispiele

```
rn_stream base1;
rn_stream base2;
rn_stream base3;
rn_stream base31 seed = 800000 antithetic;
rn_stream base4;
rn_stream base5;
rn_stream base6;
rn_stream base7;
```

Random Stream	Sample Count	Initial Position	Current Position	Antithetic Variates	Chi-Square Uniformity
base1	500000	400000	900000	OFF	0.59
base2	1956428	600000	2556428	OFF	0.98
base3	1000000	800000	1800000	OFF	0.95
base31	1000000	800000	1800000	ON	0.95
base4	500000	1000000	1500000	OFF	0.34
base5	500000	1200000	1700000	OFF	0.61
base6	3500621	1400000	4900621	OFF	0.85
base7	750082	1600000	2350082	OFF	0.57

Startwert:= alter Startwert + 200.000

letzter Wert

Chi-Square Uniformity

Test einer Verteilung

1. Hypothese: Daten stammen von einer bestimmten Verteilung
2. Bestimmung der Wahrscheinlichkeit, mit der die Hypothese gilt
3. Falls die Wahrscheinlichkeit unter einem bestimmten Schwellwert liegt, ist Hypothese für diesen Schwellwert zu verwerfen

Chi-Quadrat-Test:

- Unterteilung des Intervalls in Teilintervalle: [0, 1/k), [1/k, 2/k), … [(k-1)/k, 1)
- Von N Beobachtungen sollten im Mittel N/(k-1) in jedem Teilintervall liegen
- Sei n_j die Zahl der Beobachtung für das j-te Intervall
  bestimme Chi-Quadrat als

$$k/N \sum_{j=1}^{k} (n_j - N/k)$$

Inhalt C.2

- **Teil A**
  Aspekte
  dynamis
- **Teil B**
  Die Moc
- **Teil C**
  Die ausf
  SLX
- **Teil D**
  Modellie

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  Vertiefung der SL
- **C.4**
  GPSS-Elemente
- **C.5**
  DISCO-Elemente
- **C.6**
  Basissprache (Ergänzung)

1. Charakterisierung von Zufallsgrößen
2. Pseudozufallszahlen als Approximation von Zufallszahlen
3. Die Klasse **rn_stream**
   SLX-Basisgenerator zur Erzeugung von [0,1)-verteilten Pseudozufallszahlen
4. Einstellung von Startwerten
5. Antithetische (0,1)-verteilte Zufallszahlen
6. SLX- Zufallszahlen verschiedener Verteilungsfunktionen
7. Mathematischer Zusammenhang der Verteilungsfunktionen
8. Behandlung empirischer Verteilungsfunktionen in SLX
9. Beispiel: PrinterJob

Dynamische Generatoränderungen

… individuell für jeden Generator durch Ausführung einer Anweisung möglich: Setzen einer neuen Position

rn_seed object_ident = { **new** | value } [**antithetic**]

automatische Wahl sichert Folgen, die mind. 1 Mill Stellen unabhängig von anderen ist

Iterationsverfahren mit Startwert x_0

$$x_{j+1} \equiv k\, x_j \ \textbf{mod}\ q \quad (j = 0, 1, 2, \ldots)$$

liefert Zahlenfolge mit Periode p:

$$x_0,\ x_1,\ x_2,\ \ldots,\ x_{p-1},\ x_p = x_0,\ x_1,\ x_2,\ \ldots$$

Generator für **gleichverteilte** (31 Bit-) Zufallswerte mit **maximaler Periode**

$q = 2^{31} - 1$,
$k = 742\,938\,285$,
$p = 2^{31} - 2 = 2.147.483.646$

Generator für **(0,1)-gleichverteilte Zufallswerte**

per Transformation : $y_i = x_i / q$

Inhalt C.2

- **Teil A**
  Aspekte
  dynamis
- **Teil B**
  Die Moc
- **Teil C**
  Die ausf
  SLX
- **Teil D**
  Modellie

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  Vertiefung der SL
- **C.4**
  GPSS-Elemente
- **C.5**
  DISCO-Elemente
- **C.6**
  Basissprache (Ergänzung)

1. Charakterisierung von Zufallsgrößen
2. Pseudozufallszahlen als Approximation von Zufallszahlen
3. Die Klasse **rn_stream**
   SLX-Basisgenerator zur Erzeugung von [0,1)-verteilten Pseudozufallszahlen
4. Einstellung von Startwerten
5. Antithetische (0,1)-verteilte Zufallszahlen
6. SLX- Zufallszahlen verschiedener Verteilungsfunktionen
7. Mathematischer Zusammenhang der Verteilungsfunktionen
8. Behandlung empirischer Verteilungsfunktionen in SLX
9. Beispiel: PrinterJob

Antithetische Zufallszahlen

- Auf dem Intervall [0.0, 1.0) wird der Zufallszahl ξ
  mittels $\xi \rightarrow 1-\xi$

  ihre antithetische Zahl zugeordnet.

- Für streng monotone Funktionen kann durch Verwendung von antithetischen Zufallszahlen die Varianz verkleinert werden.

  Achtung: Wir werden dies später aufgreifen

Inhalt C.2

- **Teil A**
  Aspekte
  dynamis
- **Teil B**
  Die Moc
- **Teil C**
  Die ausf
  SLX
- **Teil D**
  Modellie

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  Vertiefung der SL
- **C.4**
  GPSS-Elemente
- **C.5**
  DISCO-Elemente
- **C.6**
  Basissprache (Ergänzung)

1. Charakterisierung von Zufallsgrößen
2. Pseudozufallszahlen als Approximation von Zufallszahlen
3. Die Klasse **rn_stream**
   SLX-Basisgenerator zur Erzeugung von [0,1)-verteilten Pseudozufallszahlen
4. Einstellung von Startwerten
5. Antithetische (0,1)-verteilte Zufallszahlen
6. SLX- Zufallszahlen verschiedener Verteilungsfunktionen
7. Mathematischer Zusammenhang der Verteilungsfunktionen
8. Behandlung empirischer Verteilungsfunktionen in SLX
9. Beispiel: PrinterJob

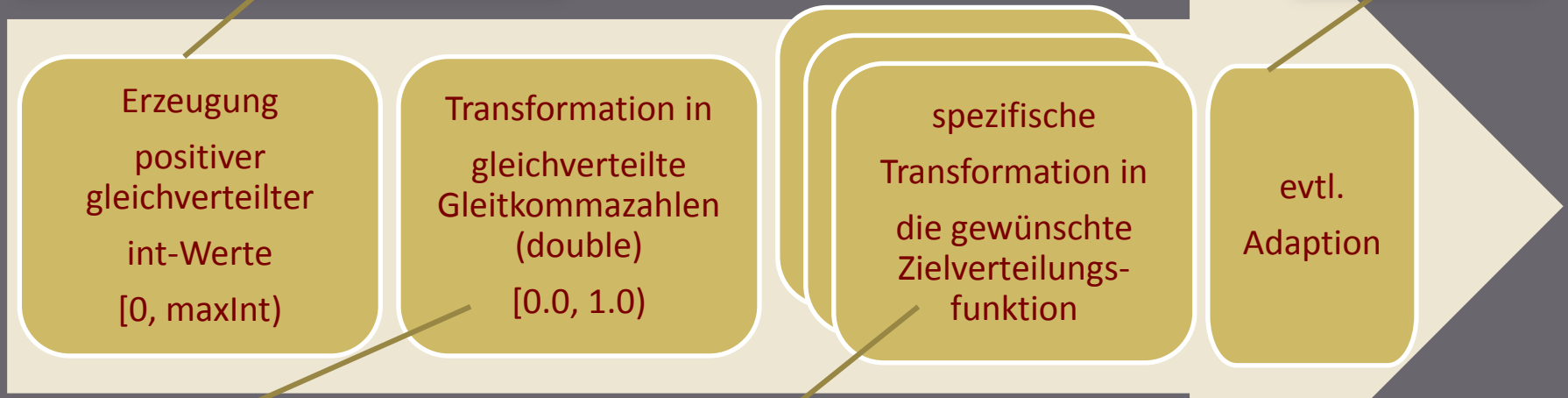
Schema zur Berechnung von Zufallszahlen

rn_stream arrive **seed**= 123456;

mit spezieller Anweisung

$x_{j+1} \equiv k\, x_j$ **mod** q (j= 0, 1, 2, ...)

Filter
Histogramme



frn (arrive); // liefert bei jedem Aufruf
// neue [0,1)-Zufallszahl

- Theoretische Verteilungsfunktionen
- Empirische Verteilungsfunktionen
- Bezier-Verteilungsfunktionen

Beispiel:
rv_expo (arrive); //liefert bei jedem Aufruf
// neue Zufallszahl entspr. Exponentialverteilung

Zufallsfunktion	SLX-Bezeichner	Parametertyp und Parametername	Typ des Rück-gabewertes
[0,1]	frn	rn_stream r	**float**
Bernoulli-Verteilung	rv_bernoulli	rn_stream r **double** probability	**int**
Beta-Verteilung	rv_beta	rn_stream r **double** alpha1 **double** alpha2	**float**
Bezier-Verteilung	rv_bezier	rn_stream r rv_bezier_data b	**double**
Binomial-Verteilung	rv_binomial	rn_stream r **int** trials **double** probability	**int**
Negative **Binomial-**Verteilung	rv_negative_binomial	rn_stream r **int** successes **double** probability	**int**
Cauchy-Verteilung	rv_cauchy	rn_stream r **double** location **double** scale	**double**
χ^2-Verteilung	rv_chi_square	rn_stream r **int** dof	**double**
Dreiecks-Verteilung	rv_triangular	rn_stream r **double** xmin **double** xmode **double** xmax	**float**
Erlang-Verteilung	rv_erlang	rn_stream r **int** m **double** beta	**float**
Extremwert-Verteilung A	rv_extreme_value_a	rn_stream r **double** gamma **double** beta	**float**
Extremwert-Verteilung B	rv_extreme_value_b	rn_stream r **double** gamma	**float**
		double beta	
Exponential-Verteilung	rv_expo	rn_stream r **double** xmean	**float**
Exponential-Power-Verteilung	rv_expo_power	rn_stream r **double** location **double** scale **double** shape	**double**
F-Verteilung	rv_F	rn_stream r **int** dof1 **int** dof2	**double**
Fehler-Verteilung	rv_error	rn_stream r **double** precision	**double**
Gamma-Verteilung	rv_gamma	rn_stream r **double** alpha **double** beta	**float**
Inverse **Gauß-**Verteilung	rv_inverse_gaussian	rn_stream r **double** alpha **double** beta	**float**
Geometrische-Verteilung	rv_geometric	rn_stream r **double** probability	**int**
Gleichverteilung	rv_uniform	rn_stream r **double** mean **double** spread	**float**
Gleichverteilung, diskret	rv_discrete_uniform	rn_stream r **int** lower_endpt **int** upper_endpt	**int**
Bounded **Johnson-**Verteilung	rv_bounded_johnson	rn_stream r **double** alpha1 **double** alpha2	**float**
Unbounded **Johnson-**Verteilung	rv_unbounded_johnson	rn_stream r **double** alpha1 **double** alpha2	**float**
Laplace-Verteilung	rv_laplace	rn_stream r **double** gamma **double** beta	**float**

Log-**Laplace**-Verteilung	rv_log_laplace	rn_stream r **double** alpha **double** beta	**float**
Logistic-Verteilung	rv_logistic	rn_stream r **double** gamma **double** beta	**float**
Log-**Logistic**-Verteilung	rv_log_logistic	rn_stream r **double** location **double** scale **double** shape	**double**
Log-Series-Verteilung	rv_log_series	rn_stream r **double** shape	**int**
Normal-Verteilung	rv_normal	rn_stream r **double** xmean **double** xstd	**float**
Log-**Normal**-Verteilung	rv_log_normal	rn_stream r **double** mean **double** variance	**float**
Pareto-Verteilung	rv_pareto	rn_stream r **double** location **double** scale	**double**
Pearson5-Verteilung	rv_pearson5	rn_stream r **double** alpha **double** beta	**float**
Pearson6-Verteilung	rv_pearson6	rn_stream r **double** alpha1 **double** alpha2 **double** beta	**float**
Poisson-Verteilung	rv_poisson	rn_stream r **double** mean	**int**
Power-Verteilung	rv_power	rn_stream r **double** lower_bound **double** upper_bound **double** shape	**double**

Random-Walk	rv_random_walk	rn_stream r **double** alpha **double** beta	**float**
Rayleigh-Verteilung	rv_rayleigh	rn_stream r **double** location **double** scale	**double**
t-Verteilung	rv_t	rn_stream r **int** dof	**double**
Wald-Verteilung	rv_wald	rn_stream r **double** location **double** shape	**double**
Weibull-Verteilung	rv_weibull	rn_stream r **double** alpha **double** beta	**float**
Invertierte **Weibull**-Verteilung	rv_inverted_weibull	rn_stream r **double** alpha **double** beta **double** gamma	**float**

```
rn_stream s;// [0,1]-Zufallszahlen
```

```
double val= rv_normal (s, 0.0, +10.0);
                    // normal-verteilte Zufallszahlen
```

Bearbeitung, Aufruf, Erfassung von Zufallszahlen

- Häufig wird nur ein bestimmter Bereich aus dem Gesamtbereich einer Verteilungsfunktion benötigt

```
rn_stream s;   //Objekt zur Erzeugung von [0,1]-Zufallszahlen
```

```
random_input name = rv_normal  (s, 2.0, 1.5) //genutzte Verteilungsfunktion
        accept ( 0, 10 )                      //eingeschränkter Teilbereich
        histogram start=0 width=1 count=10
        title= "..." ;
```

Anweisung

- generiert wird:

```
procedure sample_name returning double {
                              //Prozedur zum Aufruf
}
```

- Aufruf:

```
double val= sample_name()
```

Beispiel: Quelltext

```
//*****************************************************************************
// Example EX-Zufall
//*****************************************************************************
import <stats>
module basic {
        rn_stream base2;
        rn_stream base3;
        rn_stream base31 seed = 800000 antithetic;

        random_input norm_half=  rv_normal( base2 ,0.5,5.0 )
                accept(0.0,10.0)
                histogram start=0.0 width=1.0 count=10;

        random_input norm_full=  rv_normal( base3 ,0.5,5.0 )
                histogram start=-15 width=1 count=30;

        int aufrufe= 100000;
        double random;

        procedure main() {
                print ( aufrufe) "Versuchsanzahl : _\n";

                while (aufrufe>0) {
                        aufrufe--;
                        random= sample_norm_half();
                        random= sample_norm_full();
                }

                report (system);
        }// main
}
```

Random Variable	#Observed	Mean or ~Value	Std Dev or ~Error	Sig. Digits	Minimum	Maximum
norm_full	100000	0.483	5.014		-21.79	22.82

Lower	Upper	Frequency	Percent	
-15.0	-14.0	76	0.076	|
-14.0	-13.0	160	0.160	|
-13.0	-12.0	255	0.255	| *
-12.0	-11.0	469	0.469	| **
-11.0	-10.0	735	0.735	| ****
-10.0	-9.0	1099	1.099	| ******
-9.0	-8.0	1619	1.619	| **********
-8.0	-7.0	2275	2.275	| **********
-7.0	-6.0	3092	3.092	| **********
-6.0	-5.0	3874	3.874	| **********
-5.0	-4.0	4800	4.800	| **********
-4.0	-3.0	5829	5.829	| **********
-3.0	-2.0	6635	6.635	| **********
-2.0	-1.0	7313	7.313	| **********
-1.0	0.0	7748	7.748	| **********
0.0	1.0	8087	8.087	| **********
1.0	2.0	7802	7.802	| **********
2.0	3.0	7342	7.342	| **********
3.0	4.0	6552	6.552	| **********
4.0	5.0	5885	5.885	| **********
5.0	6.0	4759	4.759	| **********
6.0	7.0	3785	3.785	| **********
7.0	8.0	2926	2.926	| **********
8.0	9.0	2210	2.210	| **********
9.0	10.0	1653	1.653	| **********
10.0	11.0	1127	1.127	| ******
11.0	12.0	677	0.677	| ****
12.0	13.0	468	0.468	| **
13.0	14.0	290	0.290	| *
14.0	15.0	152	0.152	|

Underflow:	109	**Average Underflow:**	-16.50
Overflow:	197	**Average Overflow:**	16.44

Random Variable	#Observed	Mean or ~Value	Std Dev or ~Error	Sig. Digits	Minimum	Maximum
norm_half	100000	3.732	2.551		0.00	10.00

Lower	Upper	Frequency	Percent	
0.0	1.0	15859	15.859	| **********
1.0	2.0	15387	15.387	| **********
2.0	3.0	14290	14.290	| **********
3.0	4.0	12936	12.936	| **********
4.0	5.0	11246	11.246	| **********
5.0	6.0	9397	9.397	| **********
6.0	7.0	7514	7.514	| **********
7.0	8.0	5805	5.805	| **********
8.0	9.0	4389	4.389	| **********
9.0	10.0	3177	3.177	| **********

Random Variable	#Observed	Mean or ~Value	Std Dev or ~Error	Sig. Digits	Minimum	Maximum
norm_full	500000	0.501	5.005		-21.92	25.48

	Lower	Upper	Frequency	Percent	
	-15.0	-14.0	76	0.076	|
	-14.0	-13.0	160	0.160	|
	-13.0	-12.0	255	0.255	| *
	-12.0	-11.0	469	0.469	| **
	-11.0	-10.0	735	0.735	| ****
	-10.0	-9.0	1099	1.099	| ******
	-9.0	-8.0	1619	1.619	| **********
	-8.0	-7.0	2275	2.275	| **********
	-7.0	-6.0	3092	3.092	| **********
	-6.0	-5.0			
	-5.0	-4.0			
	-4.0	-3.0			
	-3.0	-2.0			
	-2.0	-1.0			
	-1.0	0.0			
	0.0	1.0			
	1.0	2.0			
	2.0	3.0			
	3.0	4.0			
	4.0	5.0			
	5.0	6.0			
	6.0	7.0			
	7.0	8.0			
	8.0	9.0			
	9.0	10.0	1653	1.653	| **********
	10.0	11.0	1127	1.127	| ******
	11.0	12.0	677	0.677	| ****
	12.0	13.0	468	0.468	| **
	13.0	14.0	290	0.290	| *
	14.0	15.0	152	0.152	|

Random Stream	Sample Count	Initial Position	Current Position	Antithetic Variates	Chi-Square Uniformity
base1	500000	400000	900000	OFF	0.59
base2	1956428	600000	2556428	OFF	0.98
base3	1000000	800000	1800000	OFF	0.95
base31	1000000	800000	1800000	ON	0.95
base4	500000	1000000	1500000	OFF	0.34
base5	500000	1200000	1700000	OFF	0.61
base6	3500621	1400000	4900621	OFF	0.85
base7	750082	1600000	2350082	OFF	0.57

Random Variable	#Observed	Mean or ~Value	Std Dev or ~Error	Sig. Digits	Minimum	Maximum
norm_full_antith	500000	0.499	5.005		-24.48	22.92

	Lower	Upper	Frequency	Percent	
	-15.0	-14.0	413	0.083	|
	-14.0	-13.0	847	0.169	| *
	-13.0	-12.0	1410	0.282	| *
	-12.0	-11.0	2333	0.467	| **
	-11.0	-10.0	3599	0.720	| ****
	-10.0	-9.0			
	-9.0	-8.0			
	-8.0	-7.0			
	-7.0	-6.0			
	-6.0	-5.0			
	-5.0	-4.0			
	-4.0	-3.0			
	-3.0	-2.0			
	-2.0	-1.0			
	-1.0	0.0			
	0.0	1.0			
	1.0	2.0			
	2.0	3.0			
	3.0	4.0			
	4.0	5.0			
	5.0	6.0			
	6.0	7.0			
	7.0	8.0			
	8.0	9.0	11254	2.251	| **********
	9.0	10.0	8034	1.607	| **********
	10.0	11.0	5434	1.087	| ******
	11.0	12.0	3612	0.722	| ****
	12.0	13.0	2264	0.453	| **
	13.0	14.0	1426	0.285	| *
	14.0	15.0	819	0.164	| *

```
rn_stream base2;
        rn_stream base3;
        rn_stream base31 seed = 800000 antithetic;

        random_input norm_full=  rv_normal( base3 ,0.5,5.0 )
                histogram start=-15 width=1 count=30;
        random_input norm_full_antithetic=  rv_normal( base31 ,0.5,5.0 )
                histogram start=-15 width=1 count=30;
```

Underflow:	451	Average Underflow:	-16.30
Overflow:	886	Average Overflow:	16.48

Inhalt C.2

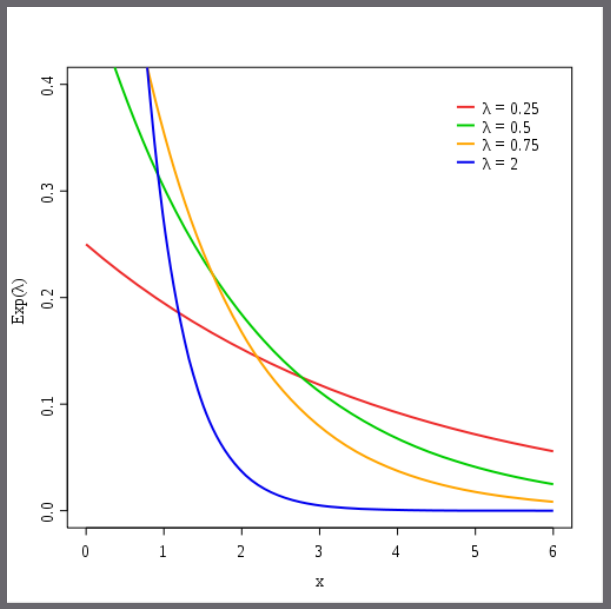
- **Teil A**
  Aspekte
  dynamis
- **Teil B**
  Die Moc
- **Teil C**
  Die ausf
  SLX
- **Teil D**
  Modellie

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  Vertiefung der SL
- **C.4**
  GPSS-Elemente
- **C.5**
  DISCO-Elemente
- **C.6**
  Basissprache (Ergänzung)

1. Charakterisierung von Zufallsgrößen
2. Pseudozufallszahlen als Approximation von Zufallszahlen
3. Die Klasse **rn_stream**
   SLX-Basisgenerator zur Erzeugung von [0,1)-verteilten Pseudozufallszahlen
4. Einstellung von Startwerten
5. Antithetische (0,1)-verteilte Zufallszahlen
6. SLX- Zufallszahlen verschiedener Verteilungsfunktionen
7. Mathematischer Zusammenhang der Verteilungsfunktionen
8. Behandlung empirischer Verteilungsfunktionen in SLX
9. Beispiel: PrinterJob

Transformationsalgorithmen

exponentialverteilte Pseudo-Zufallszahlen



rn_stream arrive **seed**= 123456;

rv_expo (arrive, 2.0); // liefert bei jedem Aufruf
// neue Zufallszahl mit
// Erwartungswert 2.0

$\{y_i\}$ sei (0, 1)- verteilte Zufallszahlenfolge

$x_i = (-1/\lambda) * \ln (1 - y_i)$ (i= 0, 1, 2, 3, ...)

$x_i = (-1/\lambda) * \ln (y_i)$ (i= 0, 1, 2, 3, ...)

➔ $x_0, x_1, x_2, ..., x_{p-1}, x_p = x_0, x_1, x_2, ...$
exponential-verteilte Zufallswerte mit Erwartungswert λ

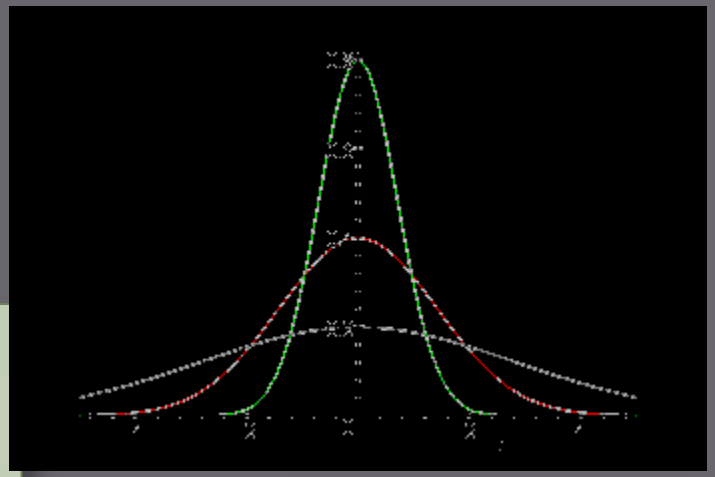
Transformationsalgorithmen

normalverteilte Pseudo-Zufallszahlen

seien y_1 und y_2 zwei aufeinander folgende Wert einer (0, 1)- gleichverteilten Zufallszahlenfolge

$$x_1 = \sqrt{-2 \ln (y_1)} \sin(2\pi y_2)$$

$$x_2 = \sqrt{-2 \ln (y1)} \cos(2\pi y2)$$



verschiedene Standardabweichungen
0.5
1.0
2.0
bei Mittelwert 0.0

➔ $x_0, x_1, x_2, \ldots, x_{p-2}, x_p = x_0, x_1, x_2, \ldots$
normal-verteilte Zufallswerte
mit Erwartungswert 0.0 und
Standardabweichung 1.0

$$z_i = m + s x_i$$

rn_stream fehler;

rv_normal (fehler, 5.2, 3.32);

normal-verteilte Zufallswerte
mit gewünschtem Erwartungswert m und gewünschter Standardabweichung s

Transformationsalgorithmen

ganzzahlige Pseudo-Zufallszahlen aus einem vorgeg. Intervall (z.B.: [1, 6] ~ Würfel)

$\{y_i\}$ sei (0, 1)- verteilte Zufallszahlenfolge

$$x_i = a + (b-a) * y_i$$

$\{x_i\}$ nimmt gleichverteilte Werte des Intervalls [a, b] an

rn_stream wuerfel;

rv_discrete_uniform (wuerfel, 1, 6);

Inhalt C.2)

- **Teil A** Aspekte dynamis
- **Teil B** Die Moc
- **Teil C** Die ausf SLX
- **Teil D** Modellie

- **C.1** Einführung und Ba
- **C.2** Stochastische Pro
- **C.3** Vertiefung der SL
- **C.4** GPSS-Elemente
- **C.5** DISCO-Elemente
- **C.6** Basissprache (Ergänzung)

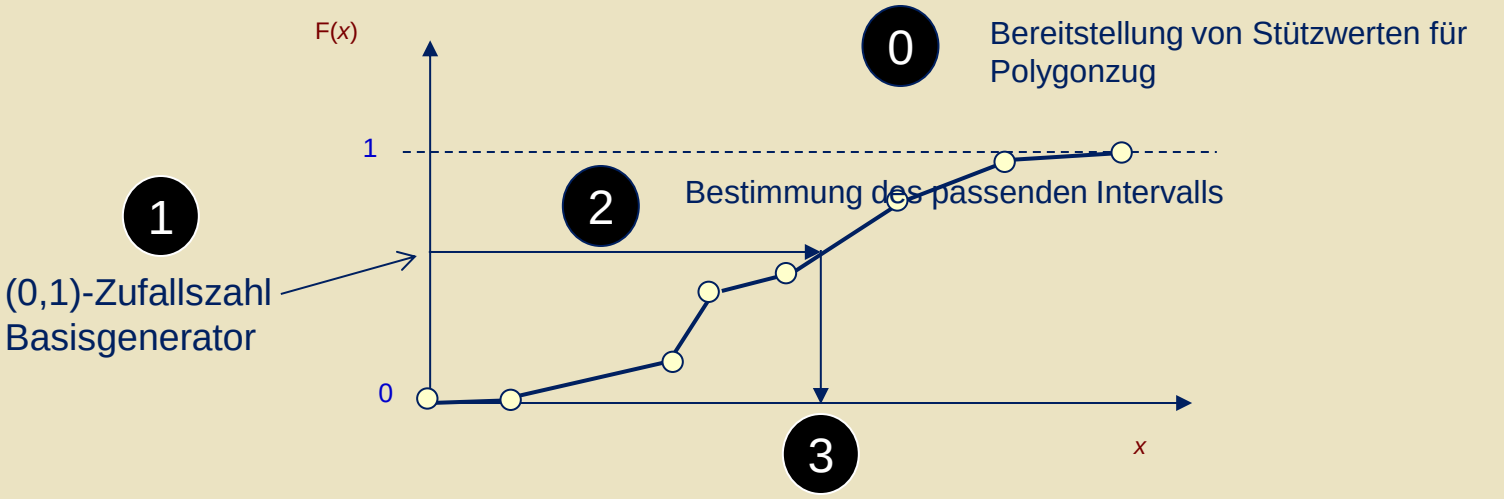
1. Charakterisierung von Zufallsgrößen
2. Pseudozufallszahlen als Approximation von Zufallszahlen
3. Die Klasse **rn_stream**
   SLX-Basisgenerator zur Erzeugung von [0,1)-verteilten Pseudozufallszahlen
4. Einstellung von Startwerten
5. Antithetische (0,1)-verteilte Zufallszahlen
6. SLX- Zufallszahlen verschiedener Verteilungsfunktionen
7. Mathematischer Zusammenhang der Verteilungsfunktionen
8. Behandlung empirischer Verteilungsfunktionen in SLX
9. Beispiel: PrinterJob

Empirische Verteilungsfunktionen

Vor.: aufgezeichnetes Histogramm einer beobachteten Größe
Häufigkeit für Werte in Werteklassen, daraus: kumulative Häufigkeit F(x)

→ Polygonzug über (x, F(x))-Stützwerte als Verteilungsfunktion:

***Methode** zur Ermittlung einer Zufallszahl entsprechend einer empirischen Verteilung F(x): Schritte 0 bis 3*



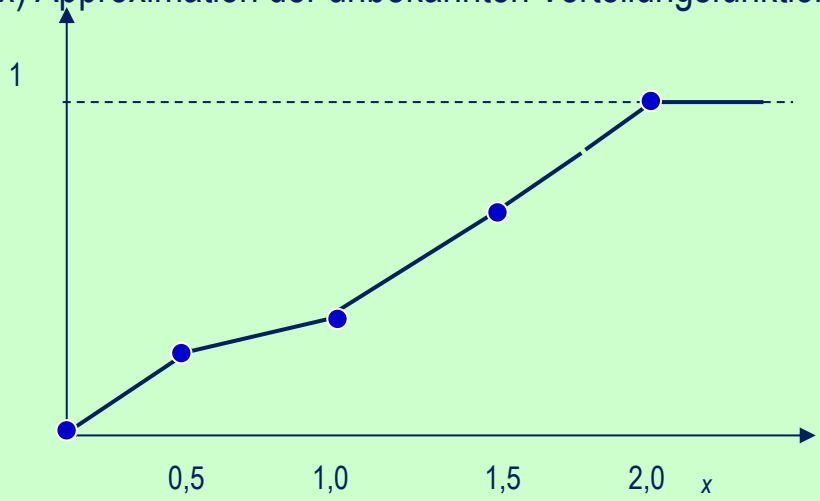
Bestimmung des x-Wertes (Zufallsgröße, die der empirischen F(x) genügt)

Empirische Verteilungen

Beispiel: Aufzeichnung von 100 Reparaturzeiten

Intervall(h)	Häufigkeit	relative Häufigkeit	kummulative Häufigkeit
0 <=x<=0.5	31	0.31	0.31
0.5 < x<=1.0	10	0.10	0.41
1.0 < x<=1.5	25	0.25	0.66
1.5 < x<=2.0	34		

F(*x*) Approximation der unbekannten Verteilungsfunktion



Vorgehensweise:
- Erzeugung einer (0,1)- verteilten Zufallszahl
- Bestimmung des passenden Intervalls (Funktionsgleichung)
- Bestimmung der Zufallsgröße (Reparaturzeit)

Diskrete Empirische Verteilungen in SLX

1. Variante: Definition von Wertepaaren

```
rn_stream s;
```

```
discrete_empirical myEmpiricalFunct (
                0.10    450,
                0.29    750,
                0.61    1000,
                0.85    1500,
                 0.9    2000,
                 1.0    3000
);
```

2. Variante: Definition per Datei mit Wertepaaren

```
discrete_empirical myEmpiricalFunct file= file_name;
```

```
procedure main() {
        int i;
        for (i=1; i<=5; i++) {
                print (i, myEmpiricalFunct(s)) "__ _____\n";
        }
}
```

```
1    3000
2    1000
3    1500
4    1000
5    1500
```

Kontinuierliche Empirische Verteilungen in SLX

1. Variante: Definition von Wertepaaren

```
rn_stream s;
```

```
continuous_empirical myEmpiricalFunct (
            0.10    450,
            0.29    750,
            0.61    1000,
            0.85    1500,
            0.9     2000,
            1.0     3000
);
```

2. Variante: Definition per Datei mit Wertepaaren

```
continuous_empirical myEmpiricalFunct file= file_name;
```

```
procedure main() {
        int i;
        for (i=1; i<=5; i++) {
                print (i, myEmpiricalFunct(s)) "__ _____\n";
        }
}
```

```
1   2852
2   872
3   1136
4   879
5   1467
```

Inhalt C.2

- **Teil A**
  Aspekte
  dynamis
- **Teil B**
  Die Moc
- **Teil C**
  Die ausf
  SLX
- **Teil D**
  Modellie

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  Vertiefung der SL
- **C.4**
  GPSS-Elemente
- **C.5**
  DISCO-Elemente
- **C.6**
  Basissprache (Ergänzung)

1. Charakterisierung von Zufallsgrößen
2. Pseudozufallszahlen als Approximation von Zufallszahlen
3. Die Klasse **rn_stream**
   SLX-Basisgenerator zur Erzeugung von [0,1)-verteilten Pseudozufallszahlen
4. Einstellung von Startwerten
5. Antithetische (0,1)-verteilte Zufallszahlen
6. SLX- Zufallszahlen verschiedener Verteilungsfunktionen
7. Mathematischer Zusammenhang der Verteilungsfunktionen
8. Behandlung empirischer Verteilungsfunktionen in SLX
9. Beispiel: PrinterJob

```
module basic {
    rn_stream arrivals, service ;
    int shutdown_time = 5*8*60, jobs_in;
    control integer  jobs_printed ;
    float total_queueing_time;
    pointer( cl_printer ) printer;
    set ( cl_printer_job ) waiting_line; // Queue for arriving jobs
```

```
class cl_printer {
   boolean printer_busy;
   pointer ( puck ) my_puck; // Puck for Printer Process
   pointer ( cl_printer_job ) owner; // Current Print Job
   actions {
     my_puck = ACTIVE; // Store the Pointer of the Puck
    forever {
      wait; // Wait for Print Jobs
         // While Contents of Job Queue != 0
      while ( (first cl_printer_job in waiting_line) != NULL ) {
          // Take First Print Job
          owner = first cl_printer_job in waiting_line;
       remove owner from waiting_line ;
          printer_busy = TRUE;
          total_queueing_time += (time –
                                  owner->my_puck->mark_time);
       advance rv_uniform ( service , 0.5, 15.0 ) ;
                                  // printing time
          printer_busy = FALSE;
          jobs_printed ++;
          // Wake up the Sleeping Print Job Process
       reactivate owner->my_puck;
          owner = NULL;
      } // while
     } // forever
   } // actions
}
```

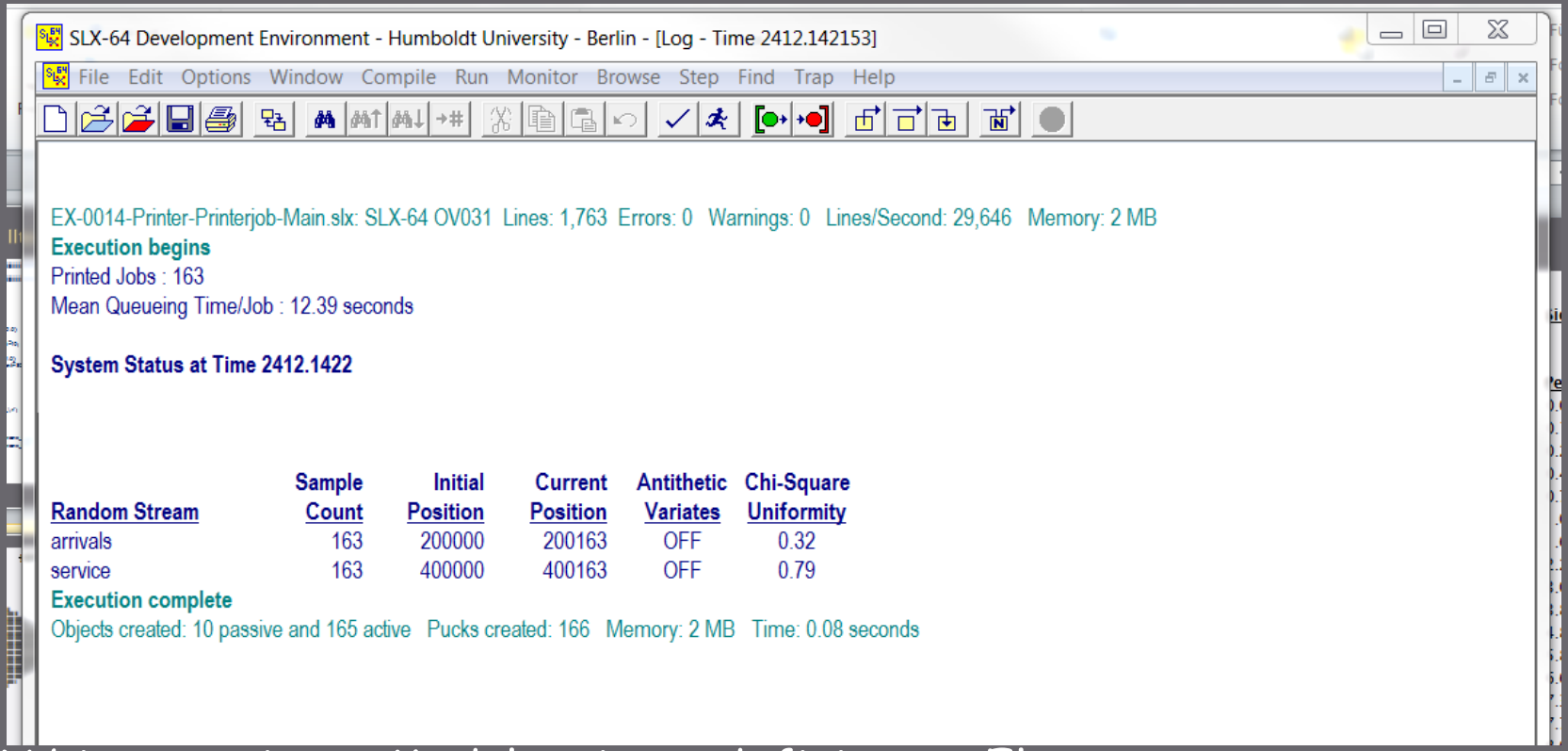
```
class cl_printer_job ( in integer in_job_num ) {
        int job_number;
        pointer ( puck ) my_puck; // Puck for printer_job process
        initial {
          job_number = in_job_num;
        }
        actions {
             my_puck = ACTIVE; // Store the pointer of the puck
             place ME into waiting_line; // Place in Job Queue
             if ( not printer->printer_busy ) // Printer process sleeping?
                  reactivate printer->my_puck;
             wait; // Sleep until wake up
             my_puck = NULL;
             terminate ;
        }
    } // cl_printer_job
```

```
procedure main() {
     printer = new cl_printer ;
    activate printer;
    while (time < shutdown_time) {
       jobs_in++;
      activate new cl_printer_job ( jobs_in ); // Create new jobs
     advance rv_uniform( arrivals , 10.0, 20.0 ) ;// interarrivel time
    }
    wait until ( jobs_in == jobs_printed);
    print ( jobs_printed ,
              total_queueing_time / jobs_printed * 60 )
              "Printed Jobs : _"
              "Mean Queueing Time/Job : _.__ seconds";
     report (system);
    }// main
}
```

Modifizierte Quelle: Thomas Schulze, James Henriksen:“Simulation needs SLX”, EX 0014

Report

- Statistik über alle verwendeten vordefinierte System-Elemente von SLX
  hier: nur Zufallszahlengeneratoren



- SLX bietet weitere Module mit vordefinierten Elementen (als Klassen implementiert)