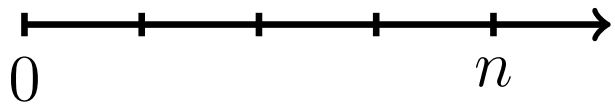
Alternativen zur Linear Kongruenten Methode

Oder: Eine kleine Reise durch die Welt der unüblichen Generatoren für Zufallszahlen

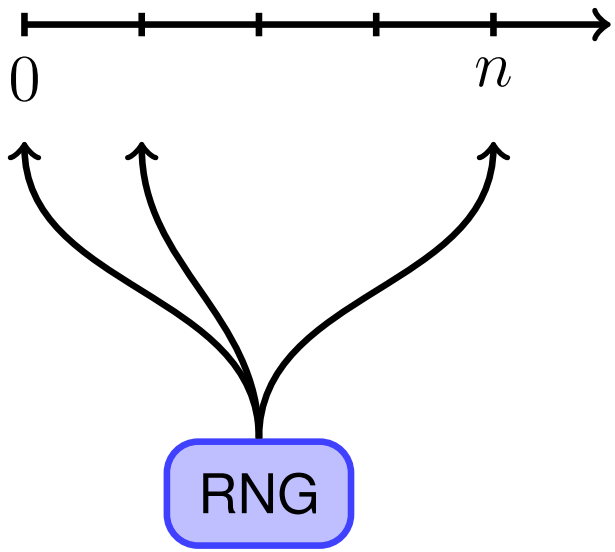
Philip Kaludercic
Hardware-Software-Co-Design, Friedrich-Alexander-Universität Erlangen-Nürnberg
13. August 2019

Problemstellung



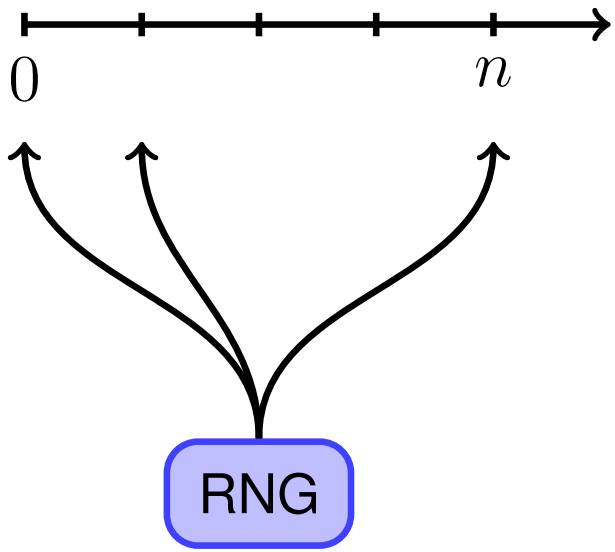
- Suche nach Wegen zufällige Zahlen zwischen $[0; n]$ zu generieren.

Problemstellung



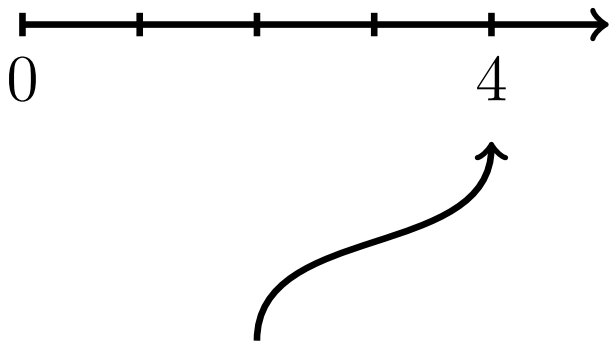
- Suche nach Wegen zufällige Zahlen zwischen $[0; n]$ zu generieren.
- Für meiste Anwendungen eignet sich (Pseudo) Random Number Generator.

Problemstellung



- Suche nach Wegen zufällige Zahlen zwischen $[0; n]$ zu generieren.
- Für meiste Anwendungen eignet sich (Pseudo) Random Number Generator.
- Fokus auf was *gute* RNGs ausmacht...

Problemstellung




```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

- Suche nach Wegen zufällige Zahlen zwischen $[0; n]$ zu generieren.
- Für meiste Anwendungen eignet sich (Pseudo) Random Number Generator.
- Fokus auf was *gute* RNGs ausmacht. . . und was *schlechte*.

Teil I

Linear Kongruente Methode

Linear Kongruente Methode

Zahlenwerte $X_1, X_2, \ldots$ werden generiert mittels[2, S.9][1, S.10]:

$$X_{n+1} = (aX_n + c) \bmod m$$

wobei

Linear Kongruente Methode

Zahlenwerte $X_1, X_2, \ldots$ werden generiert mittels[2, S.9][1, S.10]:

$$X_{n+1} = (aX_n + c) \bmod m$$

wobei

X_0 Initialwert

Linear Kongruente Methode

Zahlenwerte $X_1, X_2, \ldots$ werden generiert mittels[2, S.9][1, S.10]:

$$X_{n+1} = (aX_n + c) \bmod m$$

wobei

- X_0 Initialwert
- X_i i'te generierte Zufallswert

Linear Kongruente Methode

Zahlenwerte $X_1, X_2, \ldots$ werden generiert mittels[2, S.9][1, S.10]:

$$X_{n+1} = (aX_n + c) \bmod m$$

wobei

- X_0 Initialwert
- X_i i'te generierte Zufallswert
- a multiplikator

Linear Kongruente Methode

Zahlenwerte $X_1, X_2, \ldots$ werden generiert mittels[2, S.9][1, S.10]:

$$X_{n+1} = (aX_n + c) \bmod m$$

wobei

- X_0 Initialwert
- X_i i'te generierte Zufallswert
- a multiplikator
- c Zuwachs

Linear Kongruente Methode

Zahlenwerte $X_1, X_2, \ldots$ werden generiert mittels[2, S.9][1, S.10]:

$$X_{n+1} = (aX_n + c) \bmod m$$

wobei

- X_0 Initialwert
- X_i i'te generierte Zufallswert
- a multiplikator
- c Zuwachs
- m Modulus

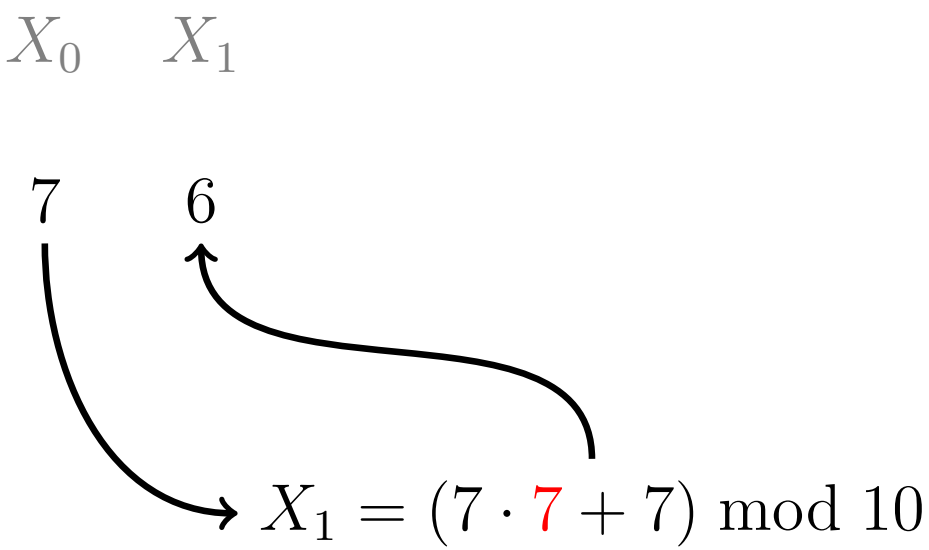
LCM Beispiel

Mit $m = 10, X_0 = 7, a = 7, c = 7$:

X_0
7

LCM Beispiel

Mit $m = 10, X_0 = 7, a = 7, c = 7$:



LCM Beispiel

Mit $m = 10, X_0 = 7, a = 7, c = 7$:

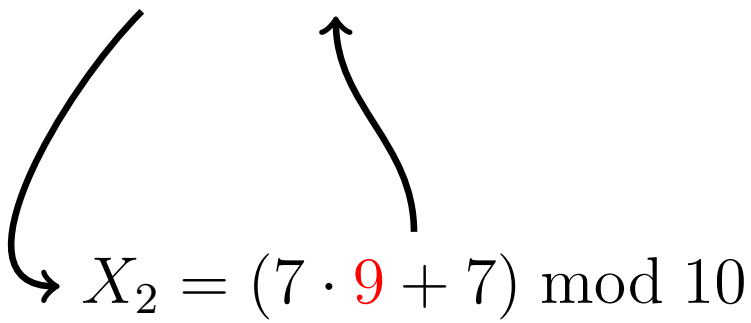
X_0	X_1	X_2
7	6	9

$$X_2 = (7 \cdot 6 + 7) \bmod 10$$

LCM Beispiel

Mit $m = 10, X_0 = 7, a = 7, c = 7$:

X_0	X_1	X_2	X_3
7	6	9	0



LCM Beispiel

Mit $m = 10, X_0 = 7, a = 7, c = 7$:

X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	$\cdots$
7	6	9	0	7	6	9	0	

Kriterien

X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	$\cdots$
7	6	9	0	7	6	9	0	7	

Kriterien

$$\underbrace{\underbrace{X_0 \quad X_1 \quad X_2 \quad X_3}_{7 \quad 6 \quad 9 \quad 0} \quad \underbrace{X_4 \quad X_5 \quad X_6 \quad X_7}_{7 \quad 6 \quad 9 \quad 0}}_{\text{Periode 1}} \quad \underbrace{X_8 \quad \cdots}_{\text{Periode 2}} \quad 7$$

Periode Länge der *kürzesten* sich insgesamt wiederholenden *Untersequenz*[2, S.9][1, S.10]

Kriterien

$$X_0 \quad X_1 \quad X_2 \quad X_3 \quad X_4 \quad X_5 \quad X_6 \quad X_7 \quad X_8 \quad \cdots$$

$$7 \quad 6 \quad 9 \quad 0 \quad 7 \quad 6 \quad 9 \quad 0 \quad 7$$

Periode

$$\frac{|\{0,6,7,9\}|}{m} = \frac{4}{10} = 0.4$$

Periode Länge der *kürzesten* sich insgesamt wiederholenden *Untersequenz*[2, S.9][1, S.10]

Füllrate Verhältnis der *möglich* erreichbaren zu *erreichten* Zahlen

Kriterien

X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	$\cdots$
7	6	9	0	7	6	9	0	7	

Periode Länge der *kürzesten* sich insgesamt wiederholenden *Untersequenz*[2, S.9][1, S.10]

Füllrate Verhältnis der *möglich* erreichbaren zu *erreichten* Zahlen

⋮

LCM Visualisierung I: Spektral-Methode-Test

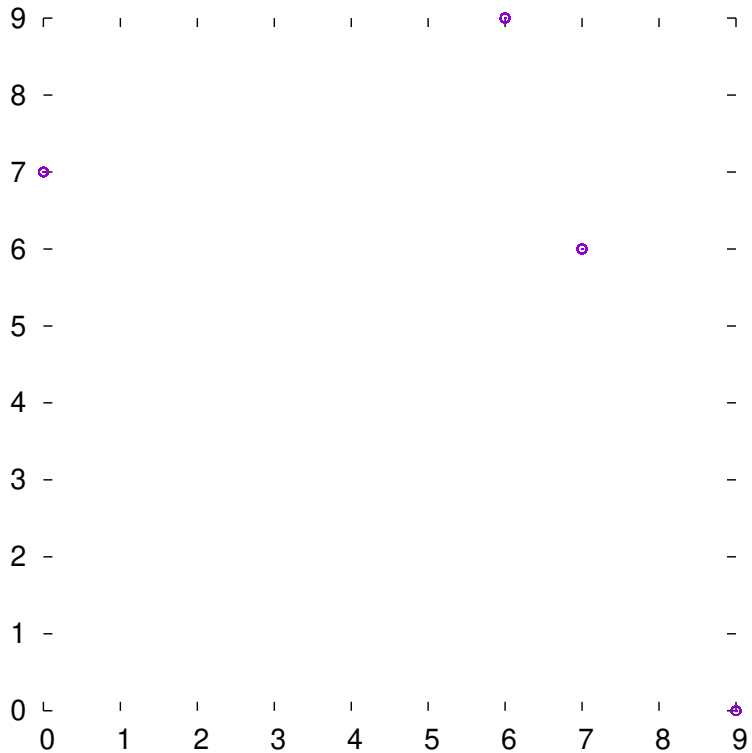


Abbildung: LCM von Vorhin

- Spektral Test[2, S.89][1, S.93] plottet X_n zu X_{n+1} Werte

LCM Visualisierung I: Spektral-Methode-Test

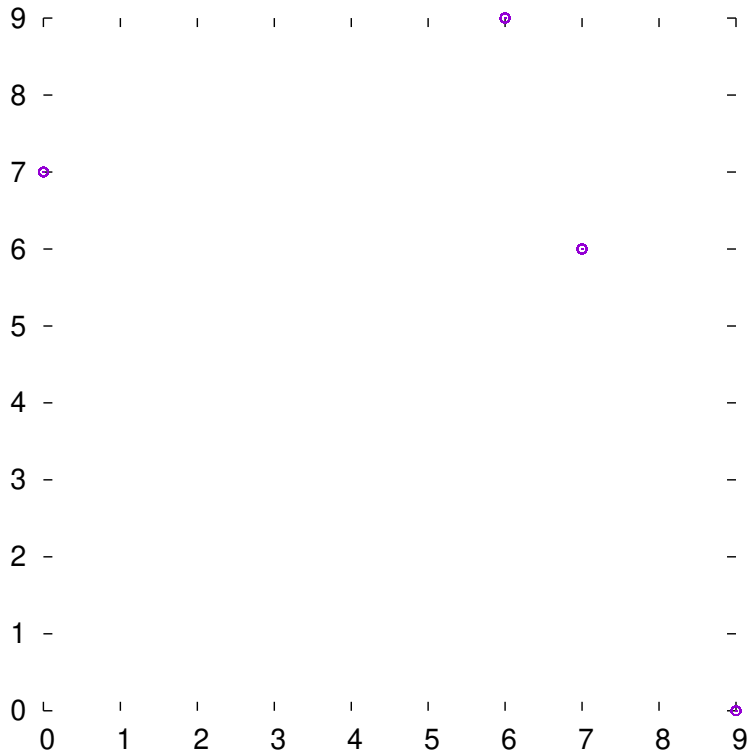


Abbildung: LCM von Vorhin

- Spektral Test[2, S.89][1, S.93] plottet X_n zu X_{n+1} Werte
- Graphisch kann gesehen werten ob ein Generator schlecht ist

$$X_0 = a = c = 7$$
$$m = 10$$

LCM Visualisierung I: Spektral-Methode-Test

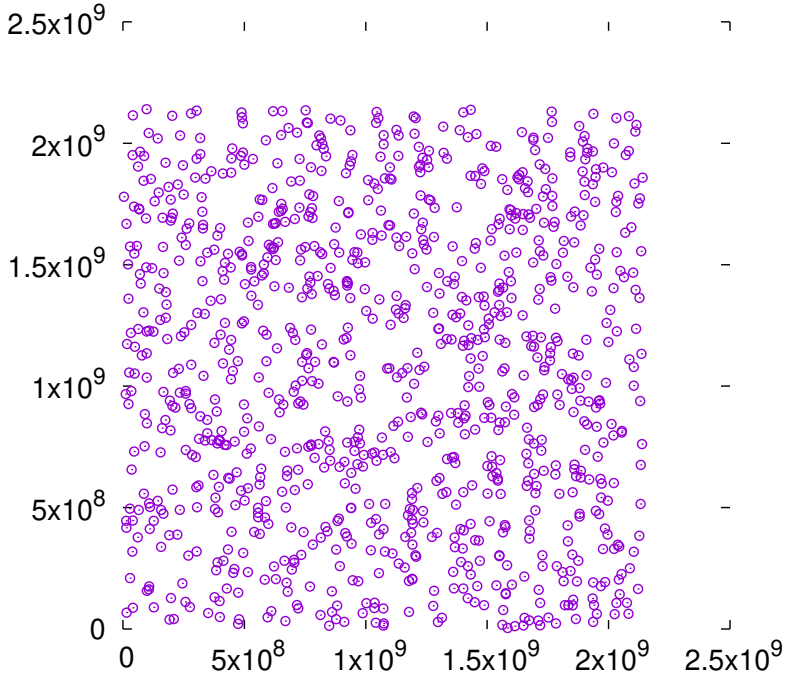


Abbildung: Glibc LCM[6, Z.364]

- Spektral Test[2, S.89][1, S.93] plottet X_n zu X_{n+1} Werte
- Graphisch kann gesehen werten ob ein Generator schlecht ist

$$X_0 = a = c = 7$$
$$m = 10$$

- . . . oder gut

$$X_0 = 0$$
$$a = 1103515245$$
$$c = 12345$$
$$m = 2^{31}$$

LCM Visualisierung II: Monte-Carlo-Test

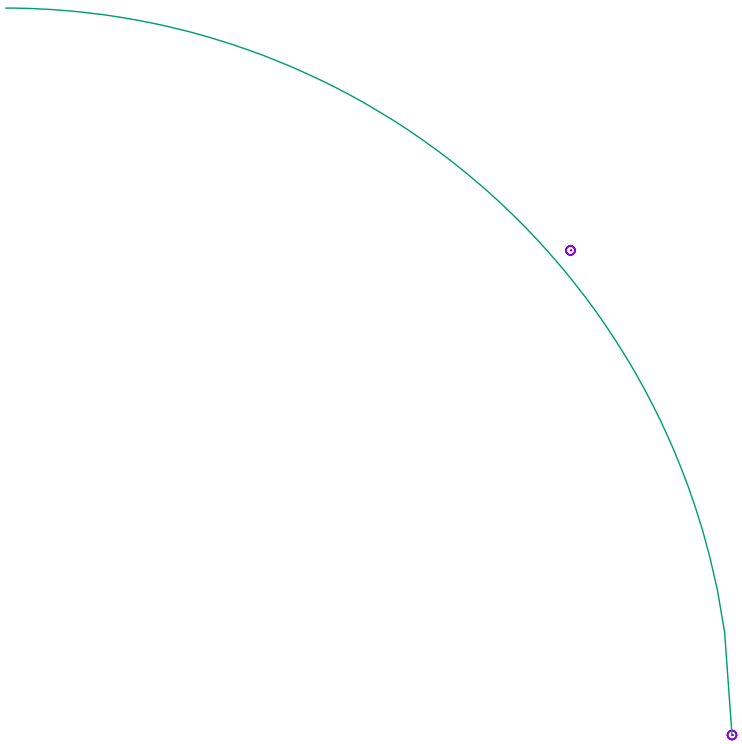


Abbildung: $n = 10$

Monte-Carlo Test zum Approximieren von π:

n	Treffer	Δ

LCM Visualisierung II: Monte-Carlo-Test

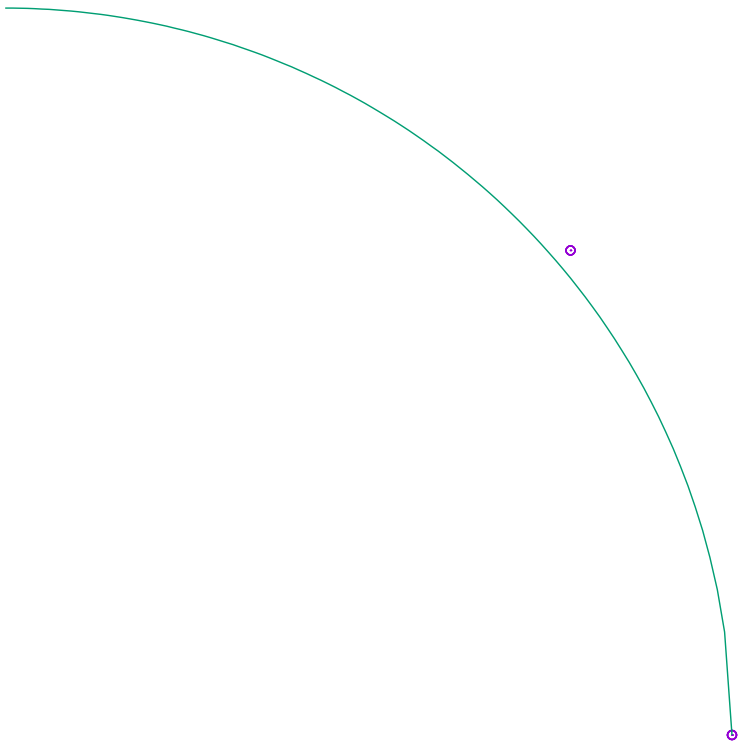


Abbildung: $n = 10$

Monte-Carlo Test zum Approximieren von π:

n	Treffer	Δ
10	0	3.14159265358

LCM Visualisierung II: Monte-Carlo-Test

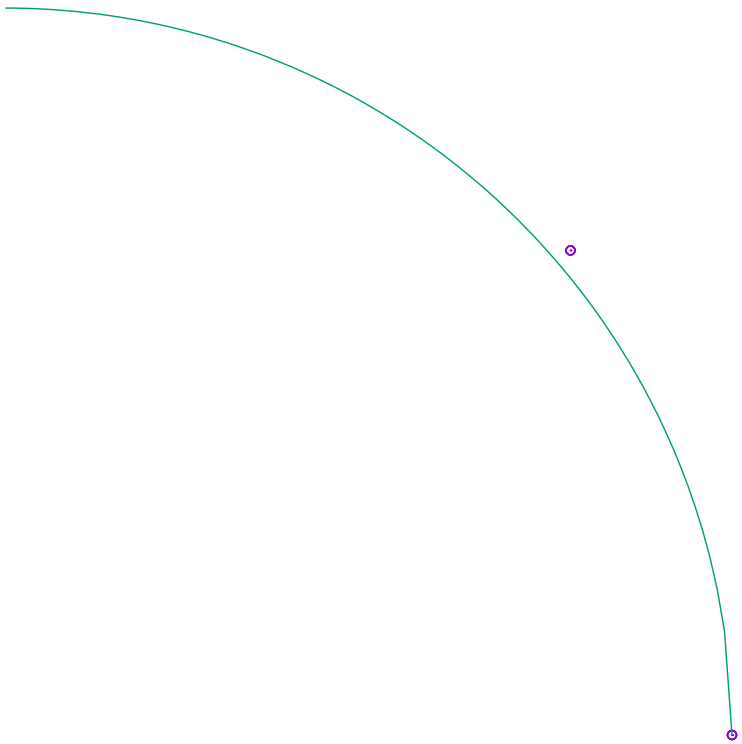


Abbildung: $n = 100$

Monte-Carlo Test zum Approximieren von π:

n	Treffer	Δ
10	0	3.14159265358
100	0	3.14159265358

LCM Visualisierung II: Monte-Carlo-Test

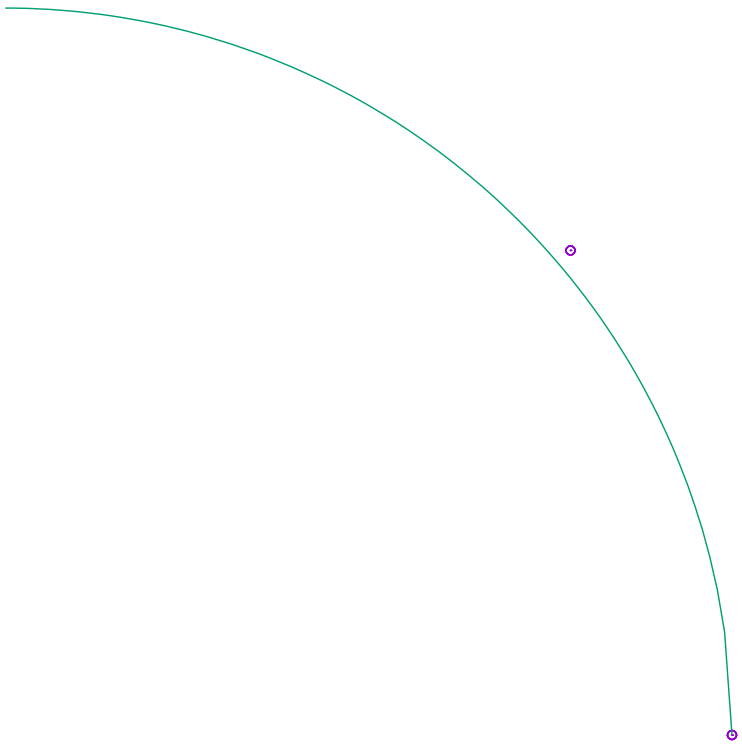


Abbildung: $n = 1000$

Monte-Carlo Test zum Approximieren von π:

n	Treffer	Δ
10	0	3.14159265358
100	0	3.14159265358
1000	0	3.14159265358

LCM Visualisierung II: Monte-Carlo-Test

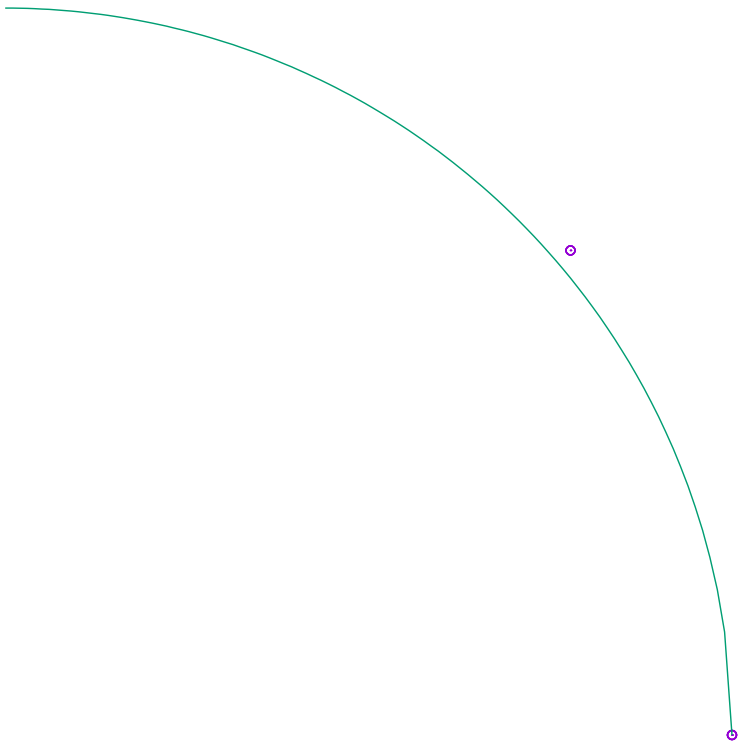


Abbildung: $n = 10000$

Monte-Carlo Test zum Approximieren von π:

n	Treffer	Δ
10	0	3.14159265358
100	0	3.14159265358
1000	0	3.14159265358
10000	0	3.14159265358

LCM Visualisierung II: Monte-Carlo-Test

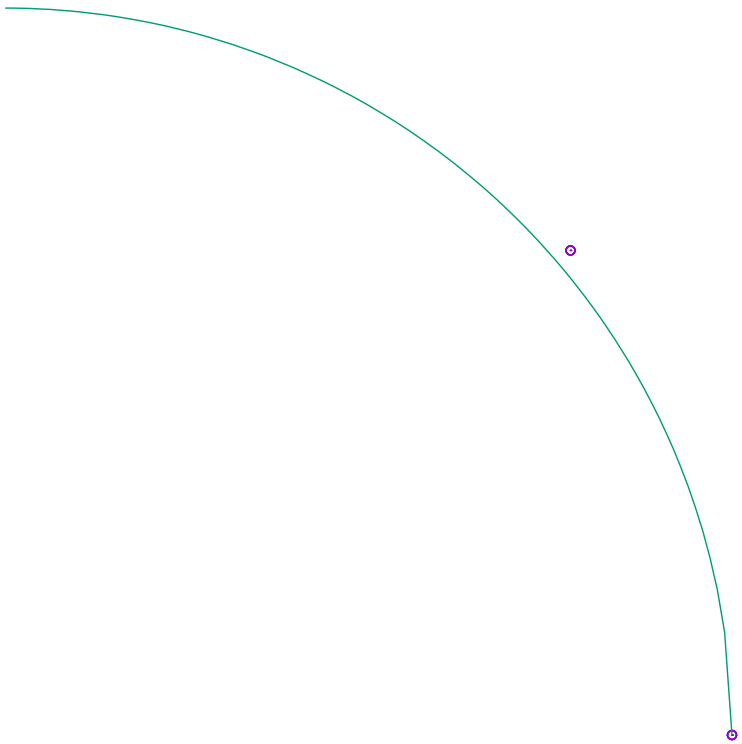


Abbildung: $n = 100000$

Monte-Carlo Test zum Approximieren von π:

n	Treffer	Δ
10	0	3.14159265358
100	0	3.14159265358
1000	0	3.14159265358
10000	0	3.14159265358
100000	0	3.14159265358

LCM Visualisierung II: Monte-Carlo-Test (Glibc)

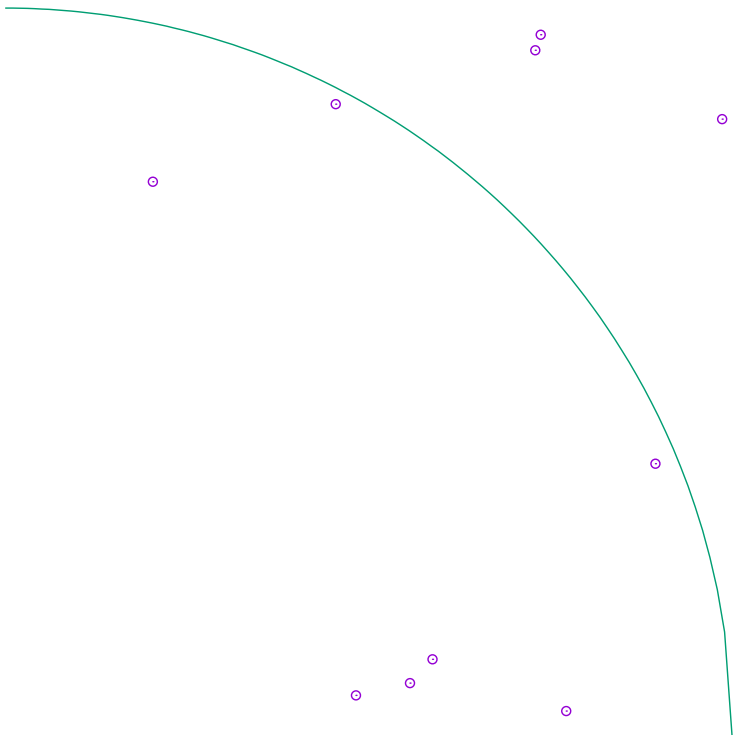


Abbildung: $n = 10$

Monte-Carlo Test zum Approximieren von π:

n	Treffer	Δ
10	7	0.3415926535

LCM Visualisierung II: Monte-Carlo-Test (Glibc)

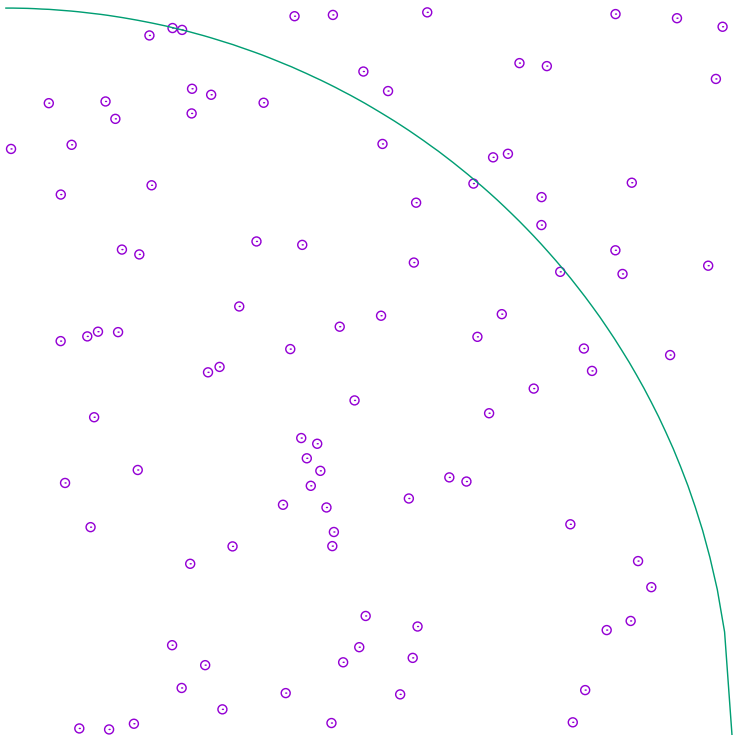


Abbildung: $n = 100$

Monte-Carlo Test zum Approximieren von π:

n	Treffer	Δ
10	7	0.3415926535
100	79	0.0184073464

LCM Visualisierung II: Monte-Carlo-Test (Glibc)

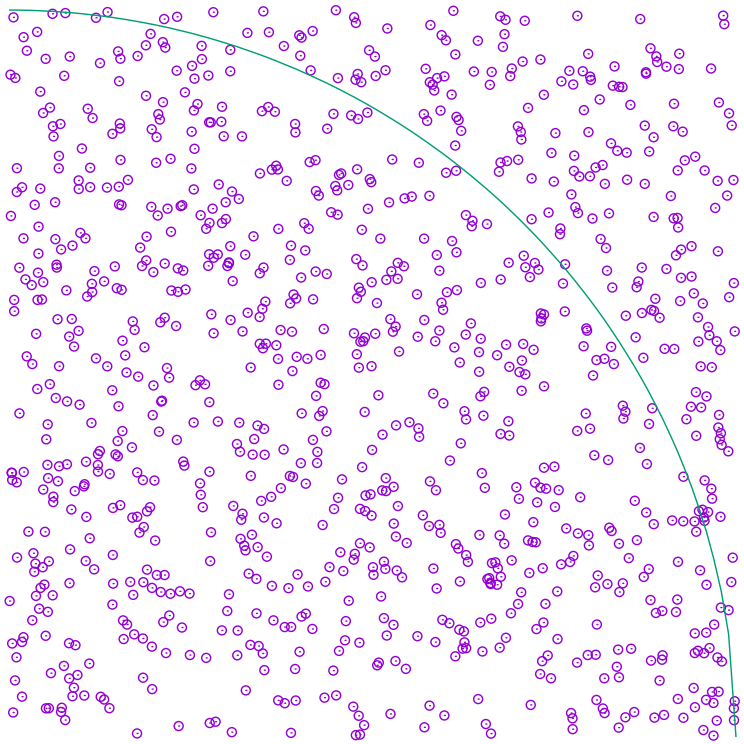


Abbildung: $n = 1000$

Monte-Carlo Test zum Approximieren von π:

n	Treffer	Δ
10	7	0.3415926535
100	79	0.0184073464
1000	797	0.0464073464

LCM Visualisierung II: Monte-Carlo-Test (Glibc)

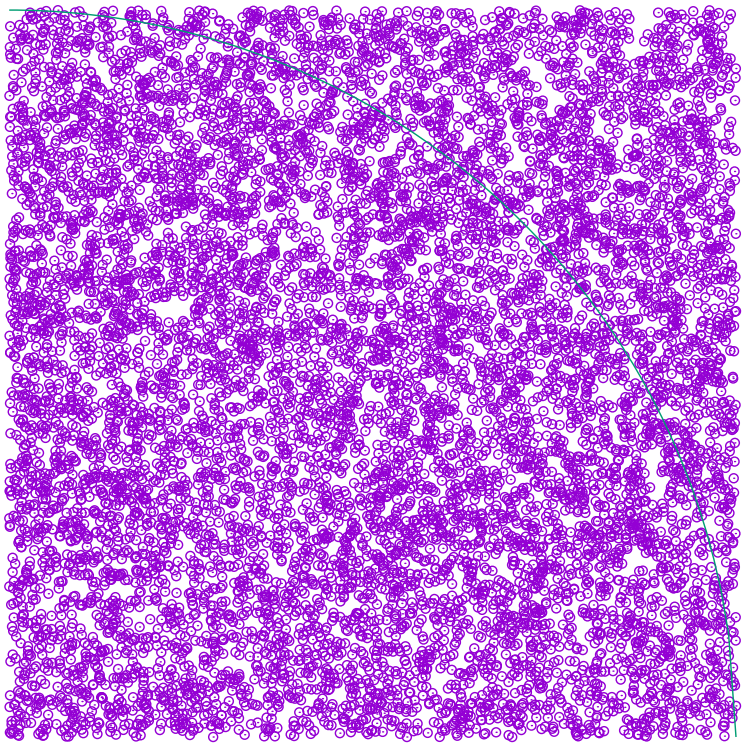


Abbildung: $n = 10000$

Monte-Carlo Test zum Approximieren von π:

n	Treffer	Δ
10	7	0.3415926535
100	79	0.0184073464
1000	797	0.0464073464
10000	7995	0.0124073461

LCM Visualisierung II: Monte-Carlo-Test (Glibc)

Abbildung: $n = 100000$

Monte-Carlo Test zum Approximieren von π:

n	Treffer	Δ
10	7	0.3415926535
100	79	0.0184073464
1000	797	0.0464073464
10000	7995	0.0124073461
100000	78627	0.0034873461

LCM Visualisierung III: Rausch-Test

RNG benutzen um zufällige Pixelwerte zu generieren:

LCM Visualisierung III: Rausch-Test

RNG benutzen um zufällige Pixelwerte zu generieren:

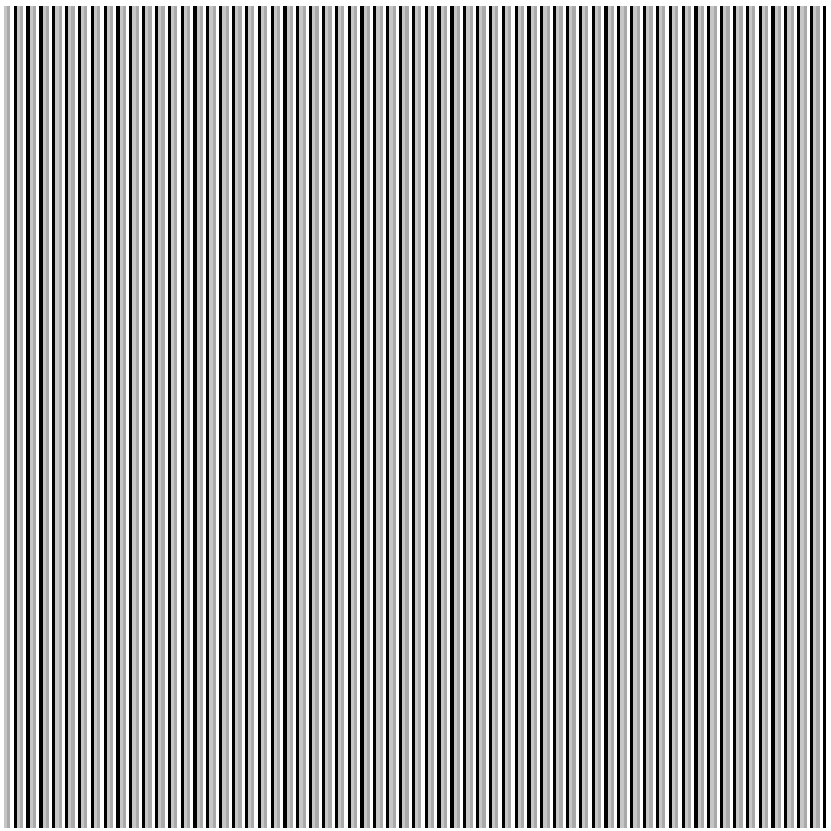


Abbildung: Einfacher LCM

LCM Visualisierung III: Rausch-Test

RNG benutzen um zufällige Pixelwerte zu generieren:

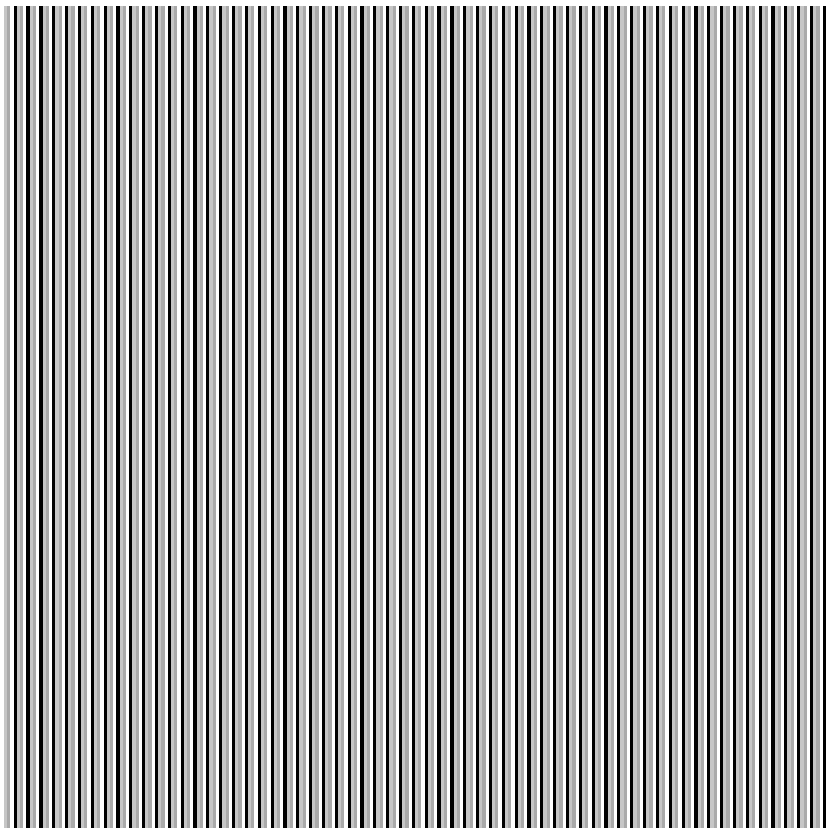


Abbildung: Einfacher LCM

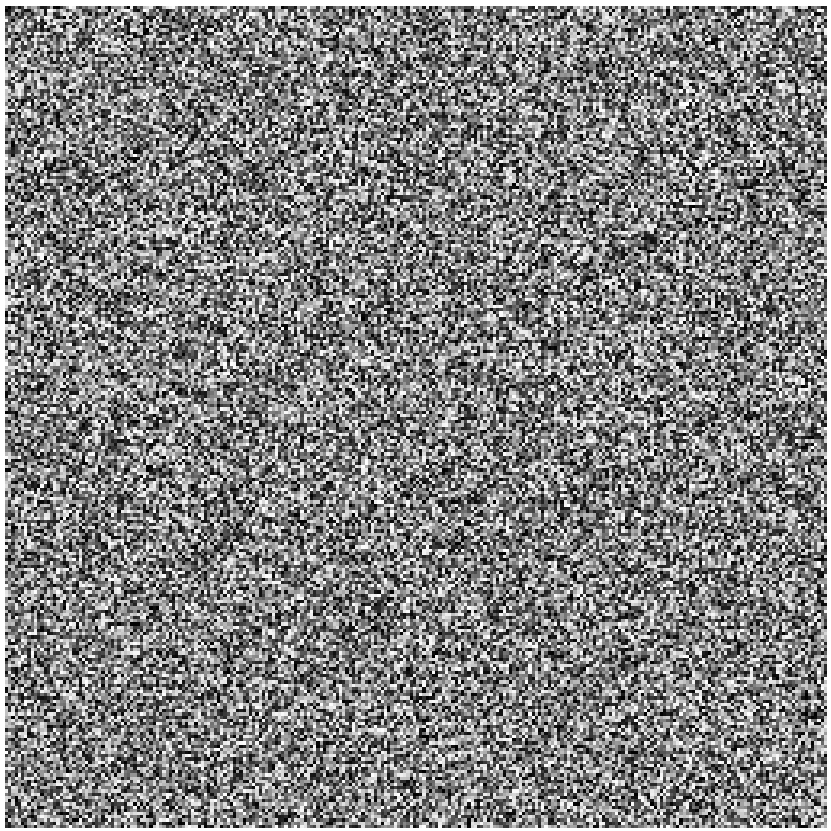


Abbildung: Glibc LCM

Teil II

Alternativen zur *Linear Kongruente Methode*

Fibonacci-Generator

Fibonacci-Generator[2, S.26][1, S.27]

Zahlenwerte $X_2, X_3, \ldots$ werden generiert mittels:

$$X_{n+1} = (X_n + X_{n-1}) \bmod m$$

wobei

$$X_0 = 1 \qquad X_1 = 1$$

Fibonacci-Generator[2, S.26][1, S.27]

Zahlenwerte $X_2, X_3, \ldots$ werden generiert mittels:

$$X_{n+1} = (X_n + X_{n-1}) \bmod m$$

wobei

$$X_0 = 1 \qquad X_1 = 1$$

X_0	X_1	X_2	X_3	X_4
1	1	2	3	5

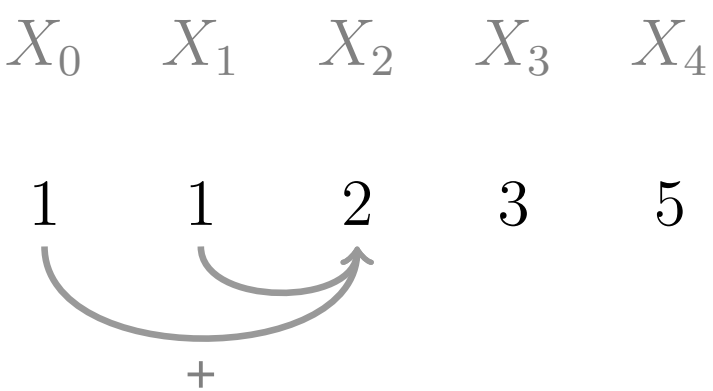
Fibonacci-Generator[2, S.26][1, S.27]

Zahlenwerte $X_2, X_3, \ldots$ werden generiert mittels:

$$X_{n+1} = (X_n + X_{n-1}) \bmod m$$

wobei

$$X_0 = 1 \qquad X_1 = 1$$



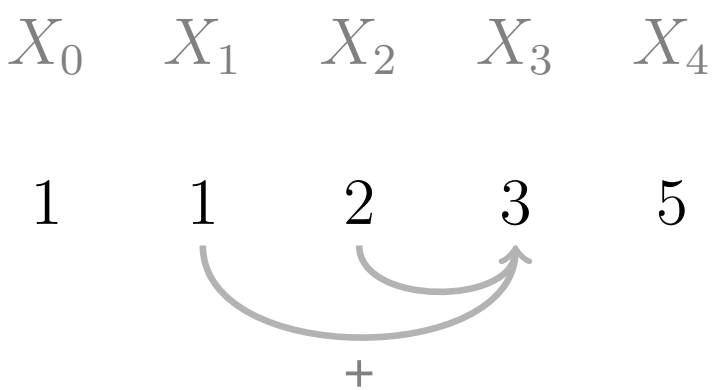
Fibonacci-Generator[2, S.26][1, S.27]

Zahlenwerte $X_2, X_3, \ldots$ werden generiert mittels:

$$X_{n+1} = (X_n + X_{n-1}) \bmod m$$

wobei

$$X_0 = 1 \qquad X_1 = 1$$



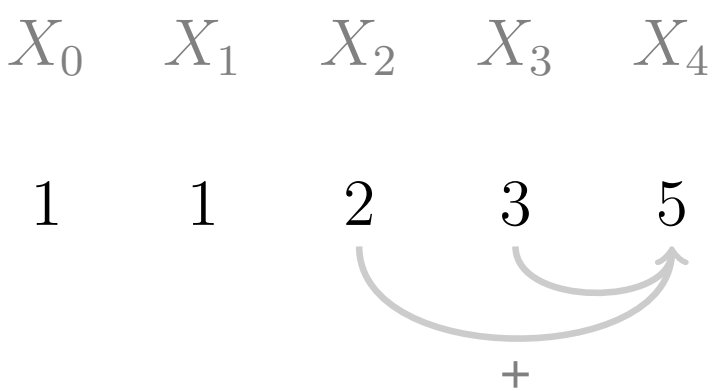
Fibonacci-Generator[2, S.26][1, S.27]

Zahlenwerte $X_2, X_3, \ldots$ werden generiert mittels:

$$X_{n+1} = (X_n + X_{n-1}) \bmod m$$

wobei

$$X_0 = 1 \qquad X_1 = 1$$



Fib. Visualisierung I: Spektral-Methode-Test

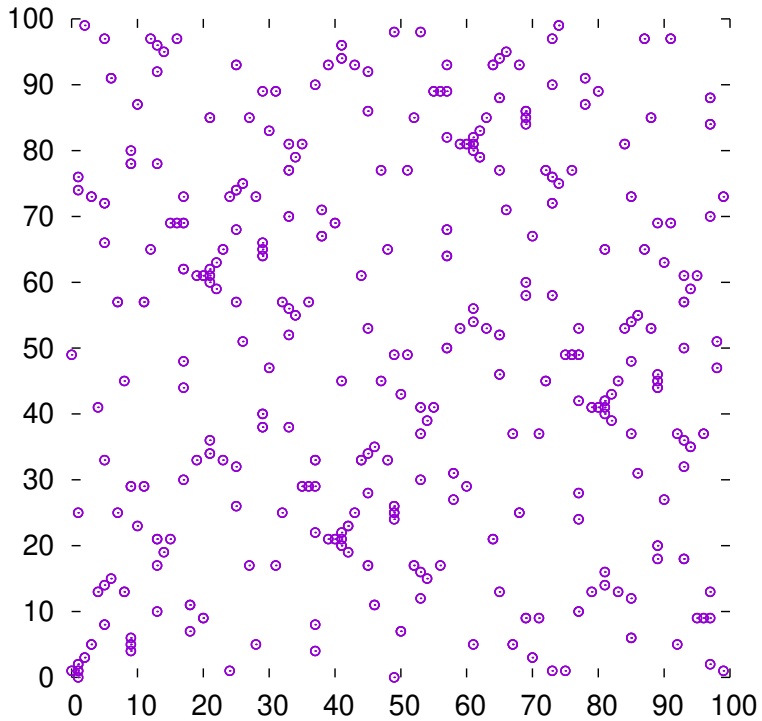


Abbildung: $m = 100$

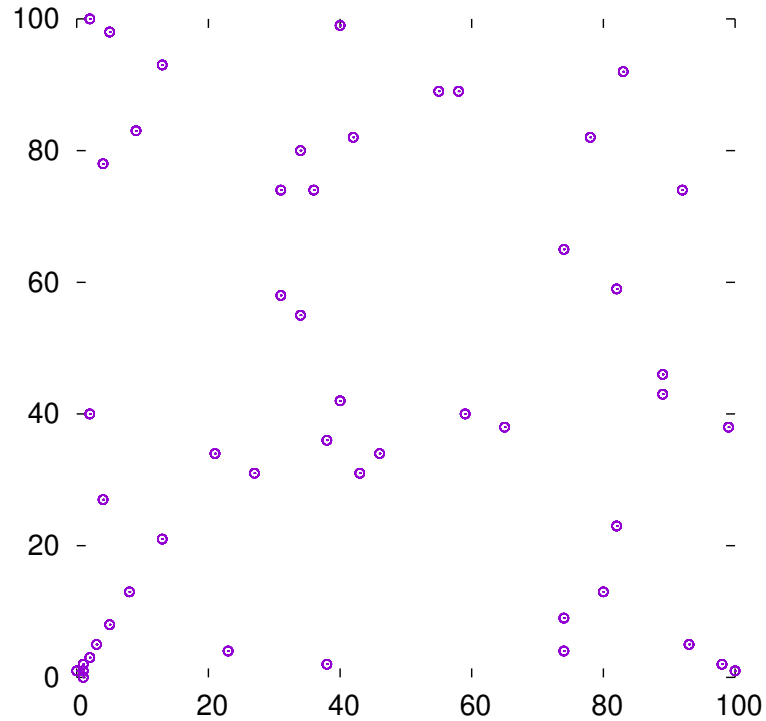


Abbildung: $m = 101$

Fib. Visualisierung II: Monte-Carlo-Test ($m = 100$)

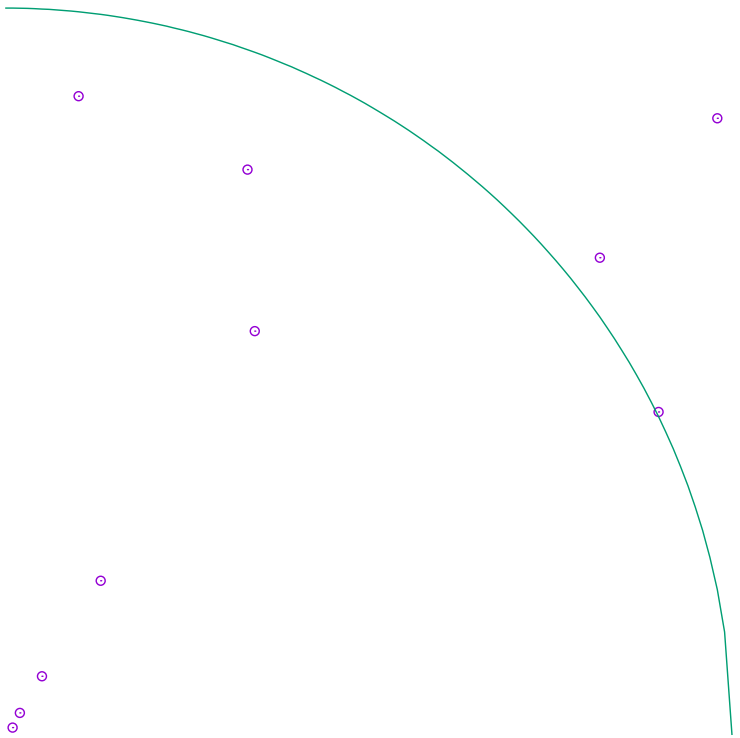


Abbildung: $n = 10$

Monte-Carlo Test zum Approximieren von π:

n	Treffer	Δ
10	7	0.34159265358

Fib. Visualisierung II: Monte-Carlo-Test ($m = 100$)

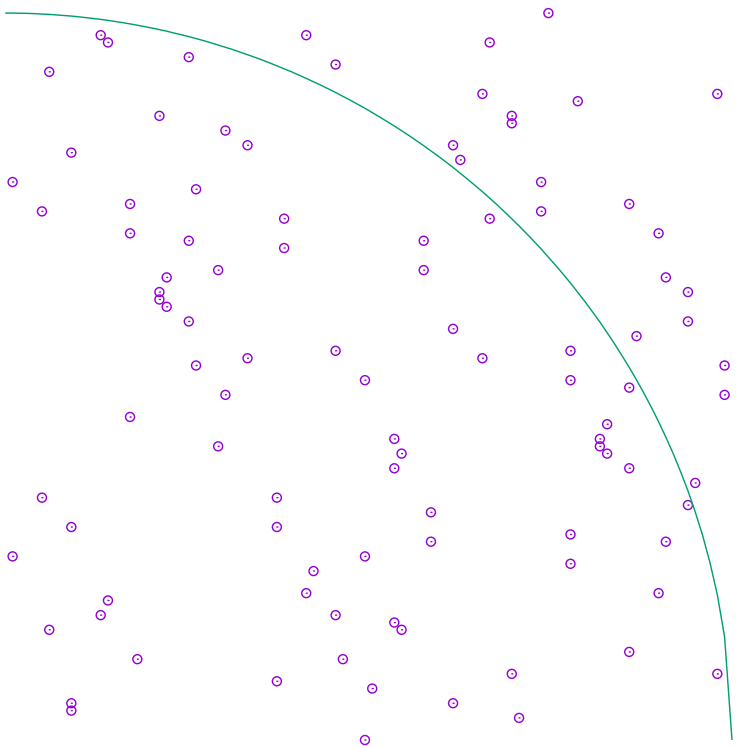


Abbildung: $n = 100$

Monte-Carlo Test zum Approximieren von π:

n	Treffer	Δ
10	7	0.34159265358
100	78	0.02159265358

Fib. Visualisierung II: Monte-Carlo-Test ($m = 100$)

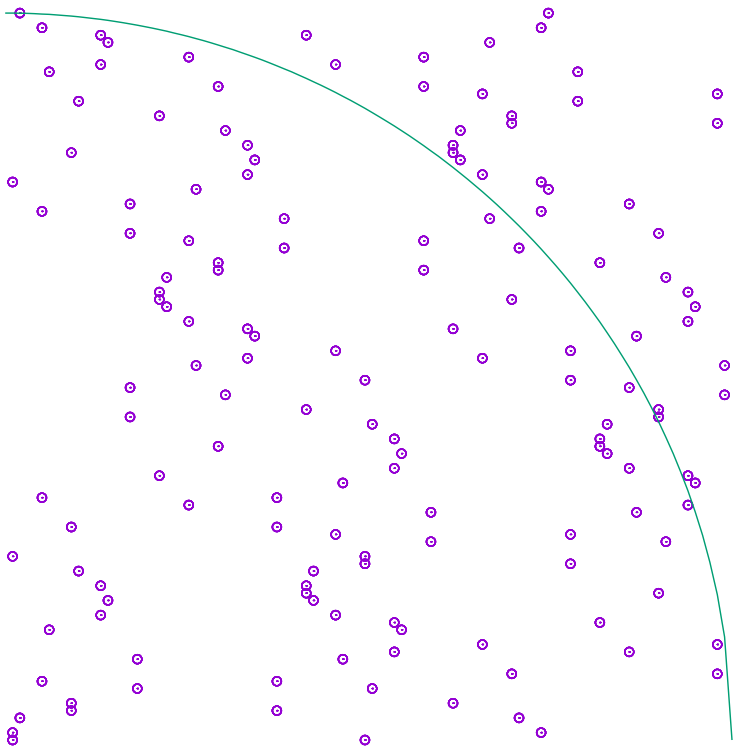


Abbildung: $n = 1000$

Monte-Carlo Test zum Approximieren von π:

n	Treffer	Δ
10	7	0.34159265358
100	78	0.02159265358
1000	750	0.14159265358

Fib. Visualisierung II: Monte-Carlo-Test ($m = 100$)

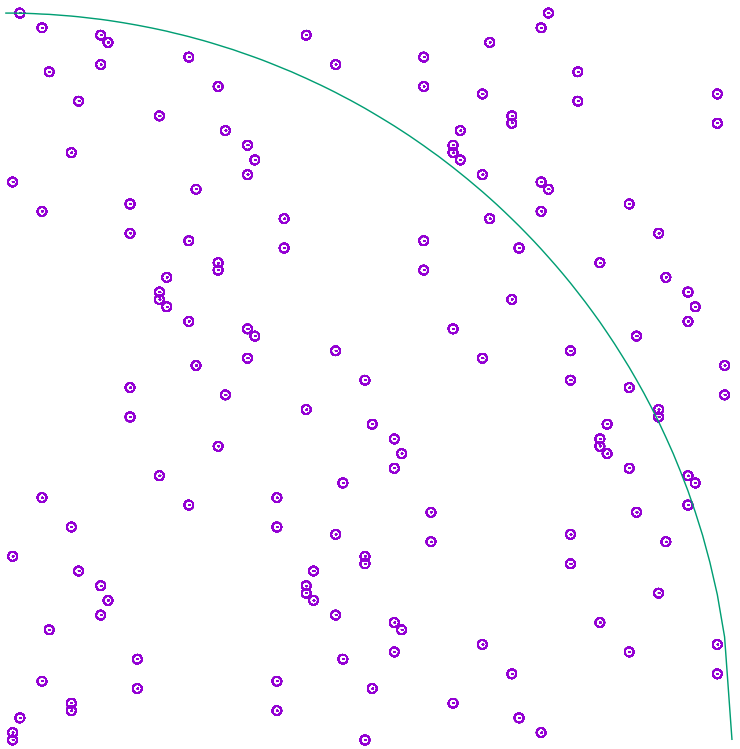


Abbildung: $n = 10000$

Monte-Carlo Test zum Approximieren von π:

n	Treffer	Δ
10	7	0.34159265358
100	78	0.02159265358
1000	750	0.14159265358
10000	7534	0.12799265358

Fib. Visualisierung II: Monte-Carlo-Test ($m = 100$)

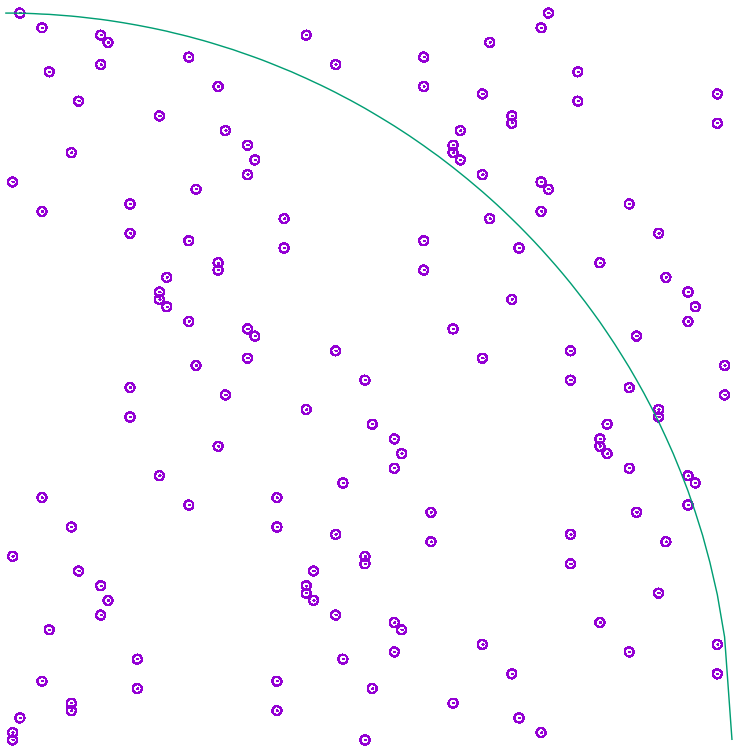


Abbildung: $n = 100000$

Monte-Carlo Test zum Approximieren von π:

n	Treffer	Δ
10	7	0.34159265358
100	78	0.02159265358
1000	750	0.14159265358
10000	7534	0.12799265358
100000	75336	0.12815265358

Fib. Visualisierung III: Rausch-Test

RNG benutzen um zufällige Pixelwerte zu generieren:

Fib. Visualisierung III: Rausch-Test

RNG benutzen um zufällige Pixelwerte zu generieren:

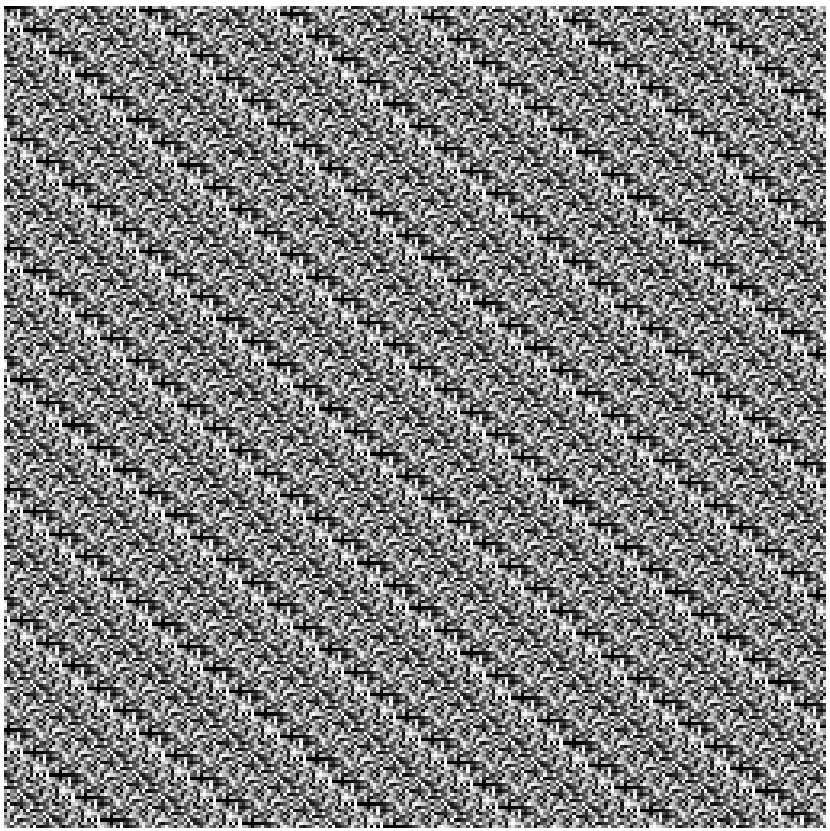


Abbildung: $m = 100$

Fib. Visualisierung III: Rausch-Test

RNG benutzen um zufällige Pixelwerte zu generieren:

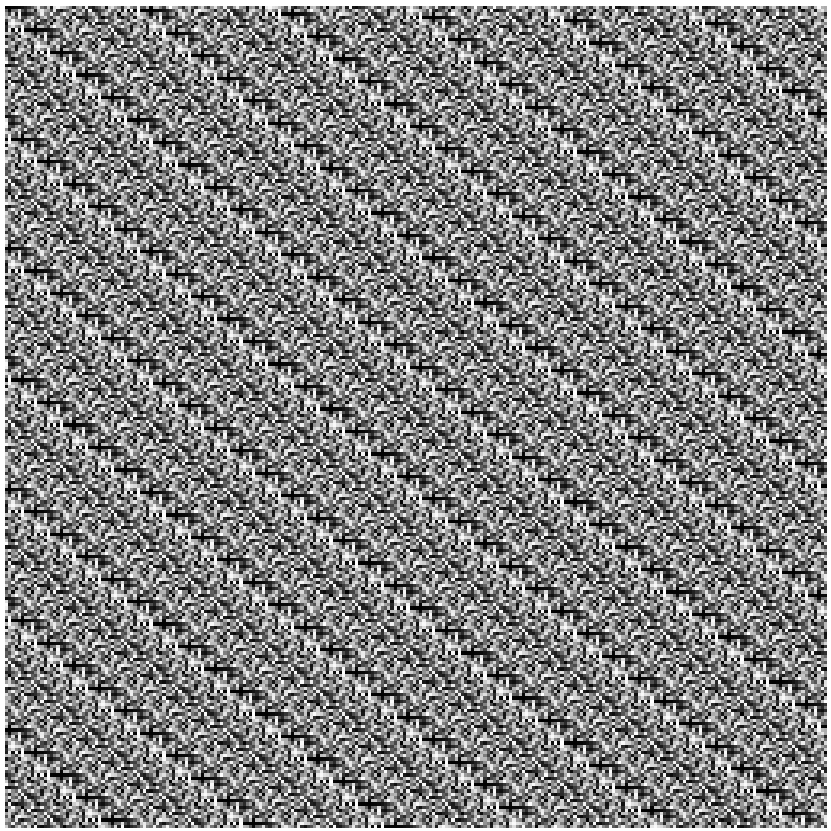


Abbildung: $m = 100$

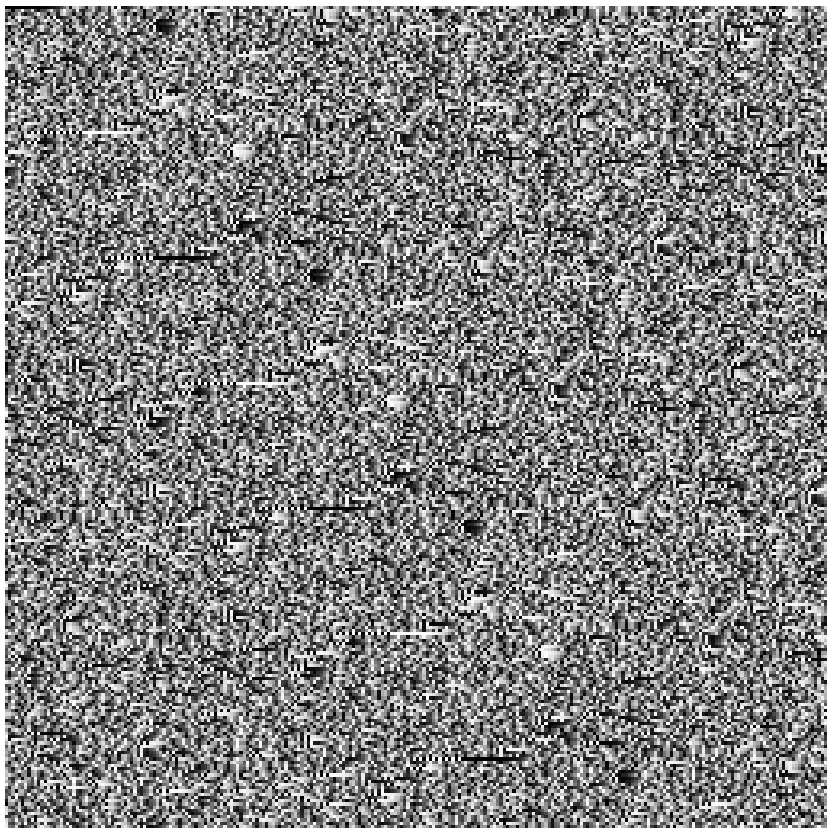


Abbildung: $m = 10007$

Beobachtungen und Anmerkungen I

Mit zwei Eingaben können hohe Perioden ($\leq m^2$) erreicht werden[2, S.26][1, S.27].

Beobachtungen und Anmerkungen II

Relative Einfachheit der Formel verhindert gute Qualität und führt zu vielen „Mustern“.

Beobachtungen und Anmerkungen III

Bspw. $X_{n-1} < X_{n+1} < X_n$ wird nie erfüllt[2, S.33, Aufgabe 2][1, S.36, ibid].

Beobachtungen und Anmerkungen IV

Obwohl früher benutzt, ist es jetzt ein Negativbeispiel!

„Mitchell-Moore“ Generator

„Mitchell-Moore“ Generator[2, S.26][1, S.27]

Zahlenwerte $X_{55}, X_{56}, \ldots$ werden generiert mittels:

$$X_n = (X_{n-24} + X_{n-55}) \bmod m$$

wobei

$$X_0, X_1, \ldots, X_{55}$$

beliebige Werte.

„Mitchell-Moore“ Generator[2, S.26][1, S.27]

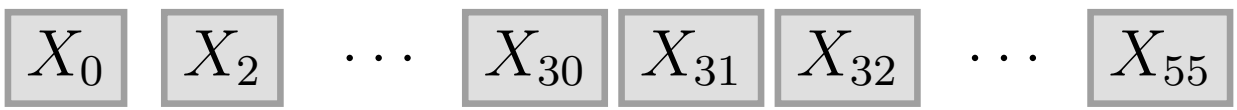
Zahlenwerte $X_{55}, X_{56}, \ldots$ werden generiert mittels:

$$X_n = (X_{n-24} + X_{n-55}) \bmod m$$

wobei

$$X_0, X_1, \ldots, X_{55}$$

beliebige Werte.



„Mitchell-Moore“ Generator[2, S.26][1, S.27]

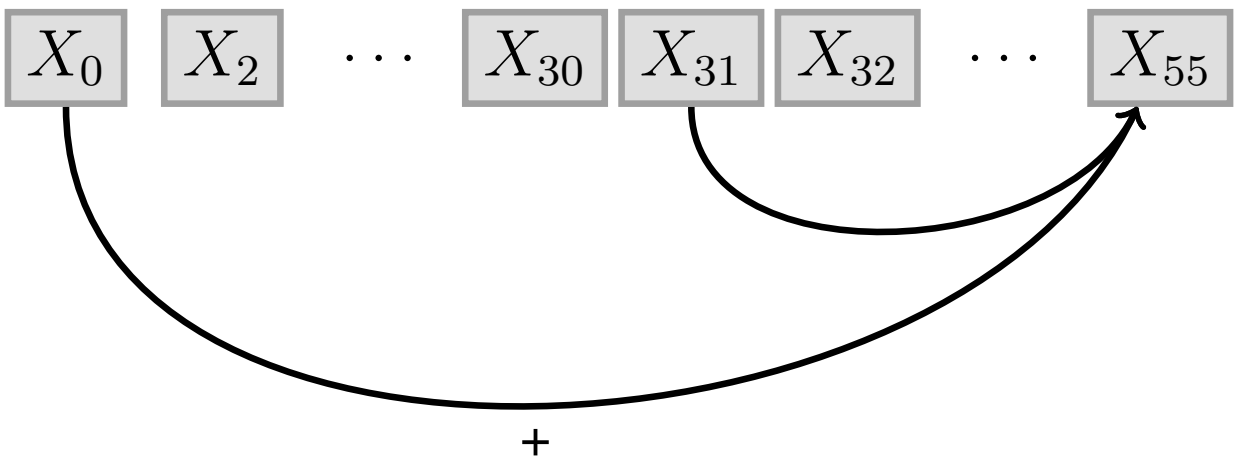
Zahlenwerte $X_{55}, X_{56}, \ldots$ werden generiert mittels:

$$X_n = (X_{n-24} + X_{n-55}) \bmod m$$

wobei

$$X_0, X_1, \ldots, X_{55}$$

beliebige Werte.



„Mitchell-Moore“ Generator[2, S.26][1, S.27]

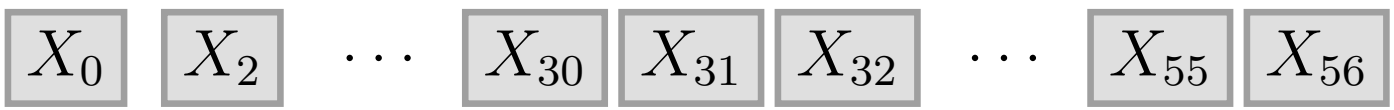
Zahlenwerte $X_{55}, X_{56}, \ldots$ werden generiert mittels:

$$X_n = (X_{n-24} + X_{n-55}) \bmod m$$

wobei

$$X_0, X_1, \ldots, X_{55}$$

beliebige Werte.



„Mitchell-Moore“ Generator[2, S.26][1, S.27]

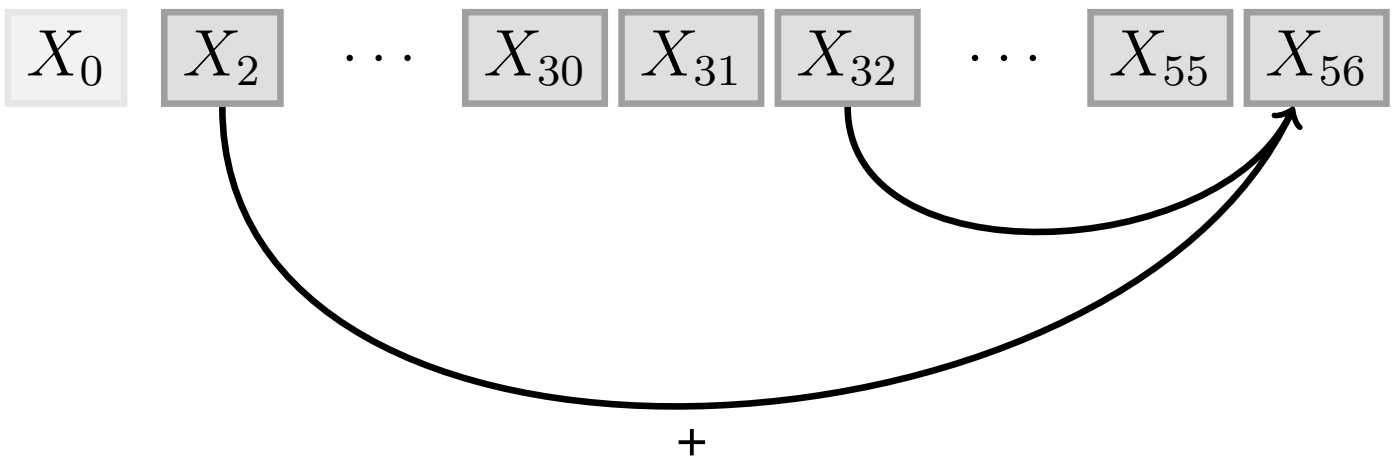
Zahlenwerte $X_{55}, X_{56}, \ldots$ werden generiert mittels:

$$X_n = (X_{n-24} + X_{n-55}) \bmod m$$

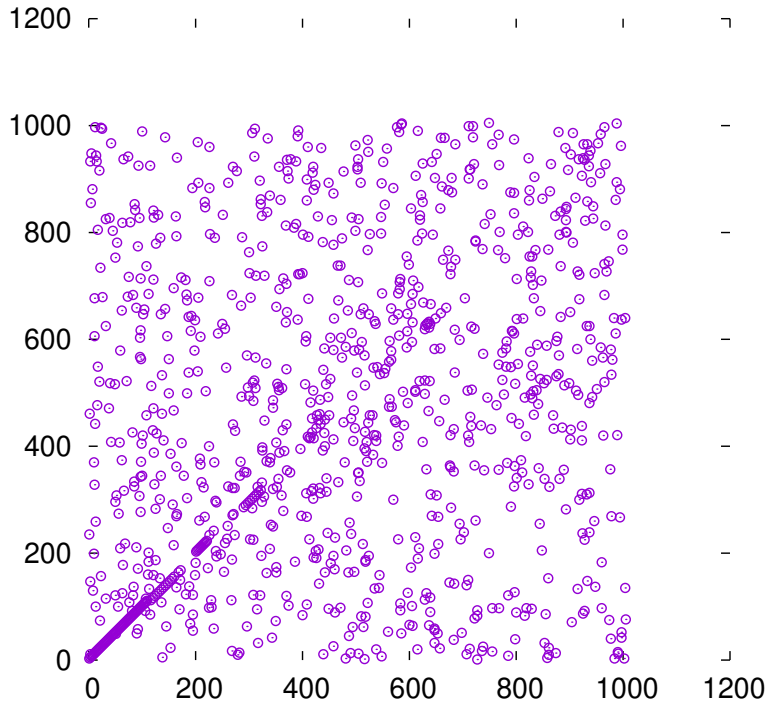
wobei

$$X_0, X_1, \ldots, X_{55}$$

beliebige Werte.



„Mitchell-Moore“ Visualisierung I: Spektral-Methode-Test



- „Aufwärmphase“ zu erkennen.

Abbildung: $X_{0,\ldots,56} = 1, 3, \ldots, m = 1007$

„Mitchell-Moore“ Visualisierung II: Monte-Carlo-Test

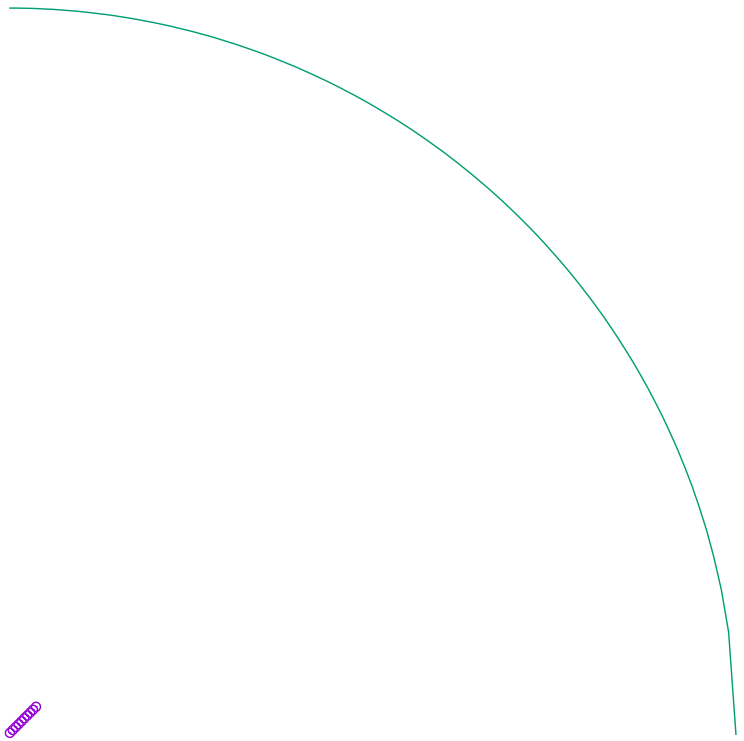


Abbildung: $n = 10$

Monte-Carlo Test zum Approximieren von π:

n	Treffer	Δ
10	10	0.858407346410

„Mitchell-Moore“ Visualisierung II: Monte-Carlo-Test

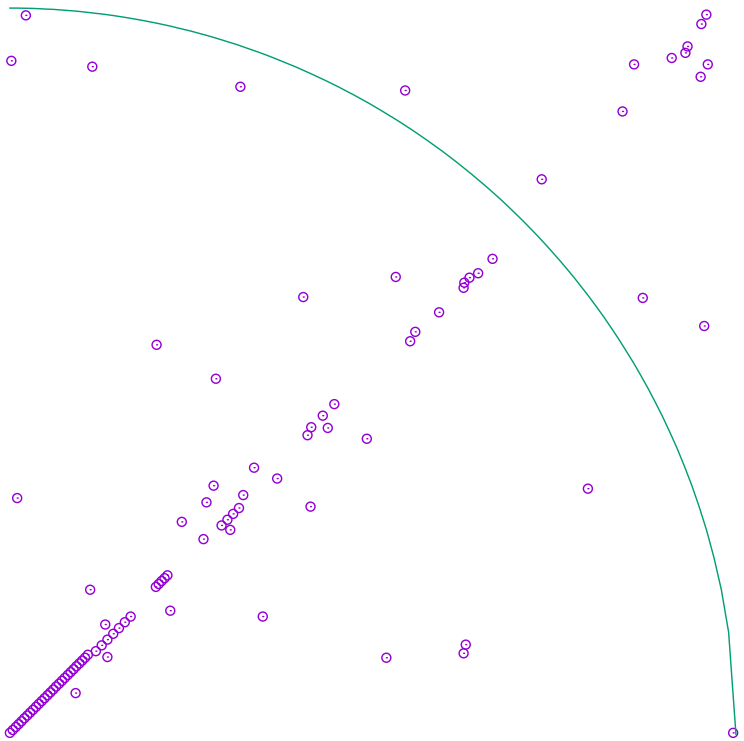


Abbildung: $n = 100$

Monte-Carlo Test zum Approximieren von π:

n	Treffer	Δ
10	10	0.858407346410
100	84	0.218407346410

„Mitchell-Moore“ Visualisierung II: Monte-Carlo-Test

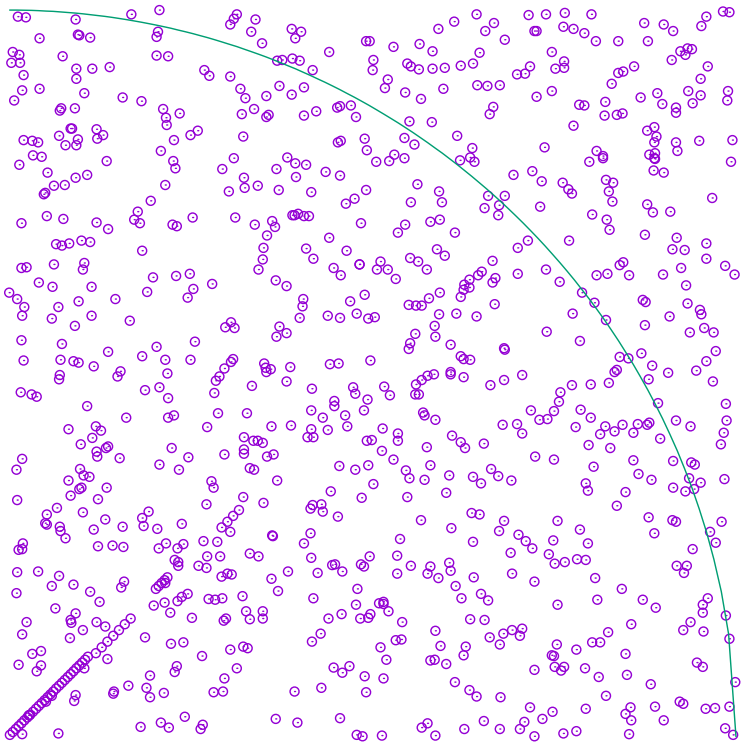


Abbildung: $n = 1000$

Monte-Carlo Test zum Approximieren von π:

n	Treffer	Δ
10	10	0.858407346410
100	84	0.218407346410
1000	793	0.030407346410

„Mitchell-Moore“ Visualisierung II: Monte-Carlo-Test

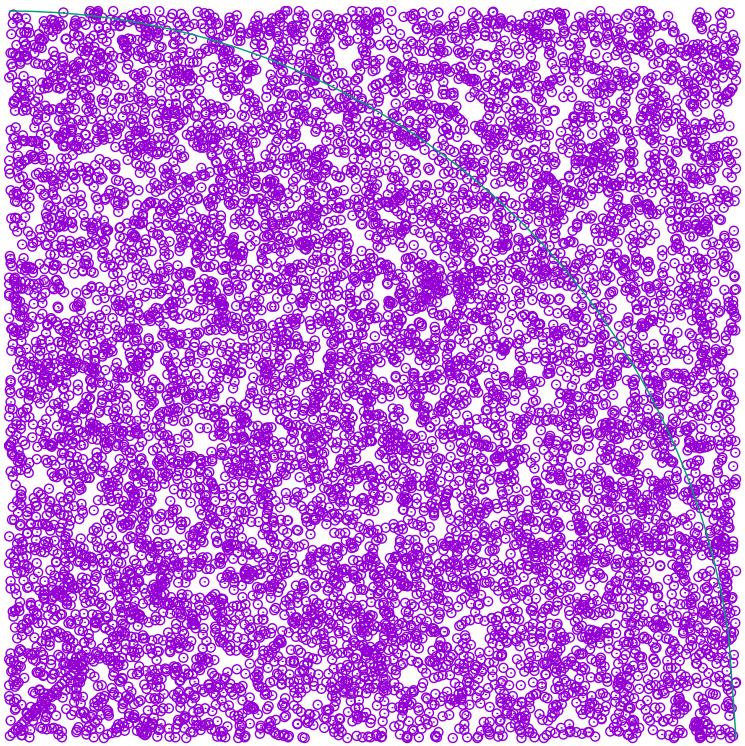


Abbildung: $n = 10000$

Monte-Carlo Test zum Approximieren von π:

n	Treffer	Δ
10	10	0.858407346410
100	84	0.218407346410
1000	793	0.030407346410
10000	7936	0.032807346410

„Mitchell-Moore“ Visualisierung II: Monte-Carlo-Test

Abbildung: $n = 100000$

Monte-Carlo Test zum Approximieren von π:

n	Treffer	Δ
10	10	0.858407346410
100	84	0.218407346410
1000	793	0.030407346410
10000	7936	0.032807346410
100000	78527	0.000512643589

„Mitchell-Moore“ Visualisierung III: Rausch-Test

RNG benutzen um zufällige Pixelwerte zu generieren:

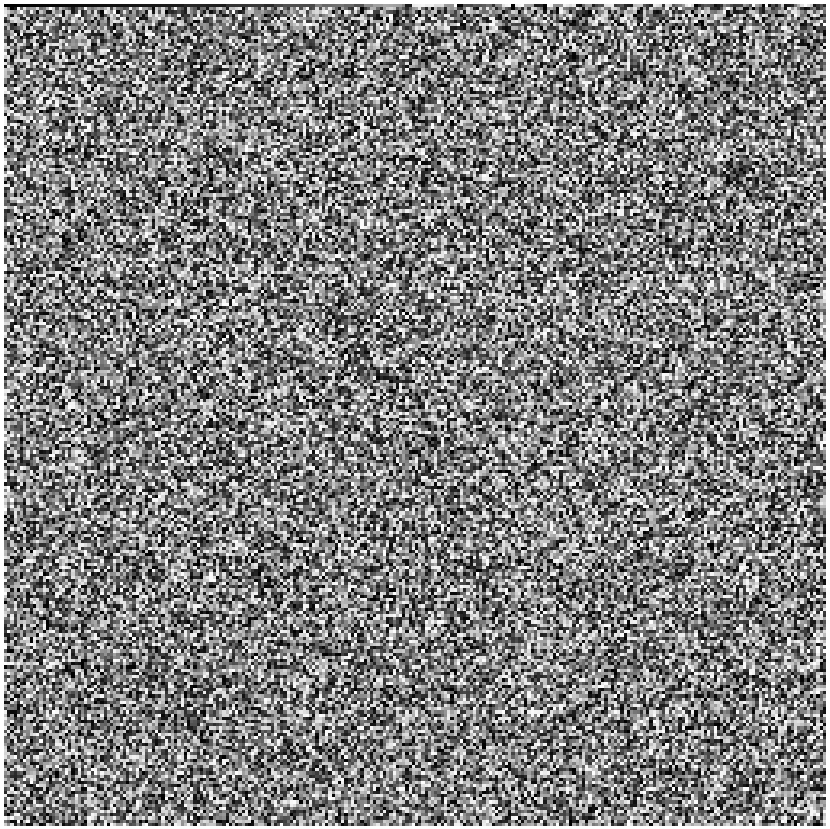


Abbildung: „Mitchell-Moore“ mit Parametern wie vorhin

Beobachtungen und Anmerkungen I

Basiert auf der Theorie von primitiven Polynomen

$$X^{55} + X^{24} + 1,$$

dh. mit Nullstelle α kann ein endlicher Körper generiert werden[2, S.30][1, S.32].

Beobachtungen und Anmerkungen II

Werte 24, 55 (lags) können mit anderen werten Ersetzt werden[2, S.26][1, S.30]:

$$(37, 100), (273, 607), (9739, 23209), \ldots$$

Beobachtungen und Anmerkungen III

Für $m = 2^e$ erreicht der Generator eine Periode von $2^{e-1}(2^{55} - 1)$[2, S.28][1, S.28].

Beobachtungen und Anmerkungen IV

Obwohl *praktisch* geeignet fehlt Theorie um dieses zu fundieren.

Beobachtungen und Anmerkungen V

Große Monte-Carlo Experimente weisen sogar *Biase* nach[1, S.29]!

Variationen auf dem bisher Gesehenem

Quadratic Congruent Method [2, S.9][1, S.10]

$$X_{n+1} = (dX_n^2 + aX_n + c) \bmod m$$

XOR Fibonacci „[2, S.30][1, S.32]“

$$X_{n+1} = (X_n \otimes X_{n-1}) \bmod m$$

Lineare Kombinationen[2, S.28][1, S.29]

$$X_{n+1} = (a_1 X_n + \ldots + a_k X_{n-k}) \bmod m$$

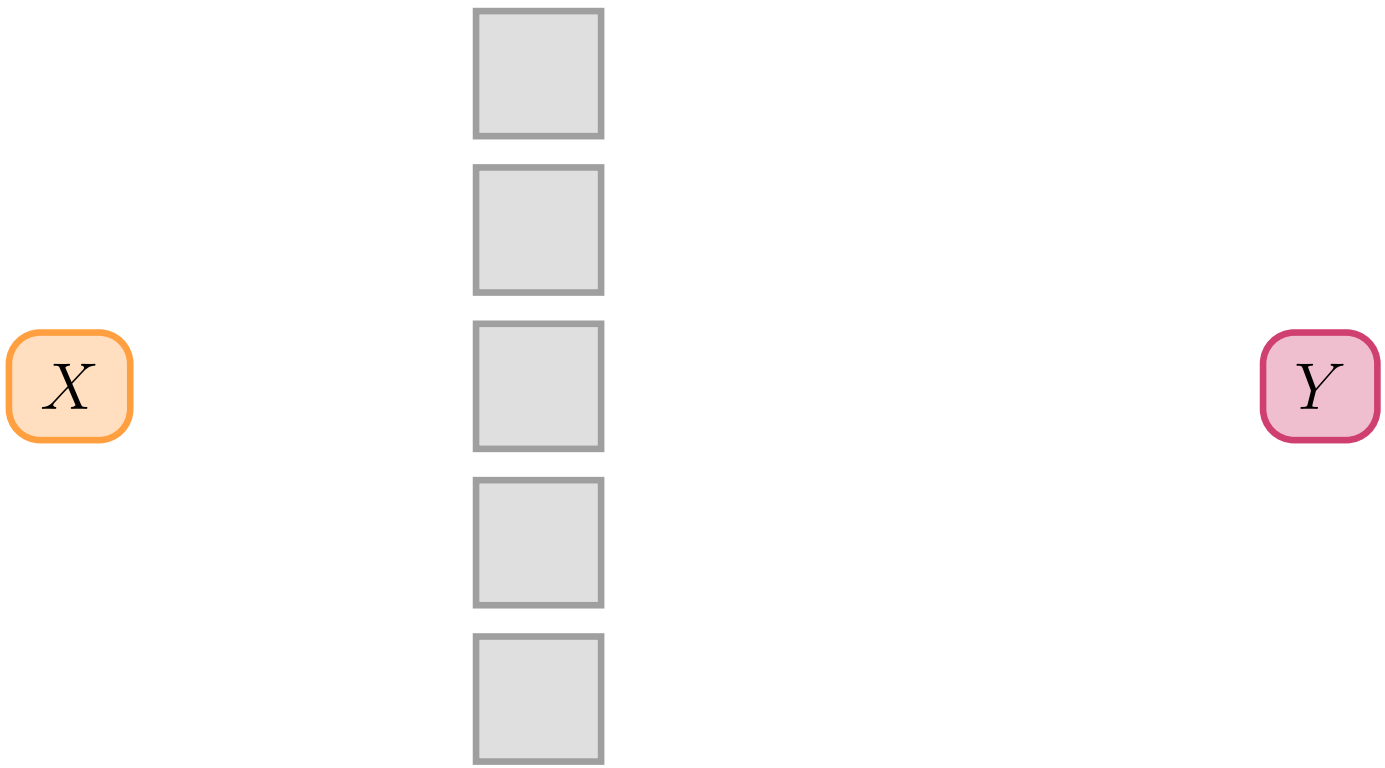
Algo. M Generator

„Algorithmus M“ Generator[2, S.32][1, S.33]

Mit zwei Generatoren X und Y:

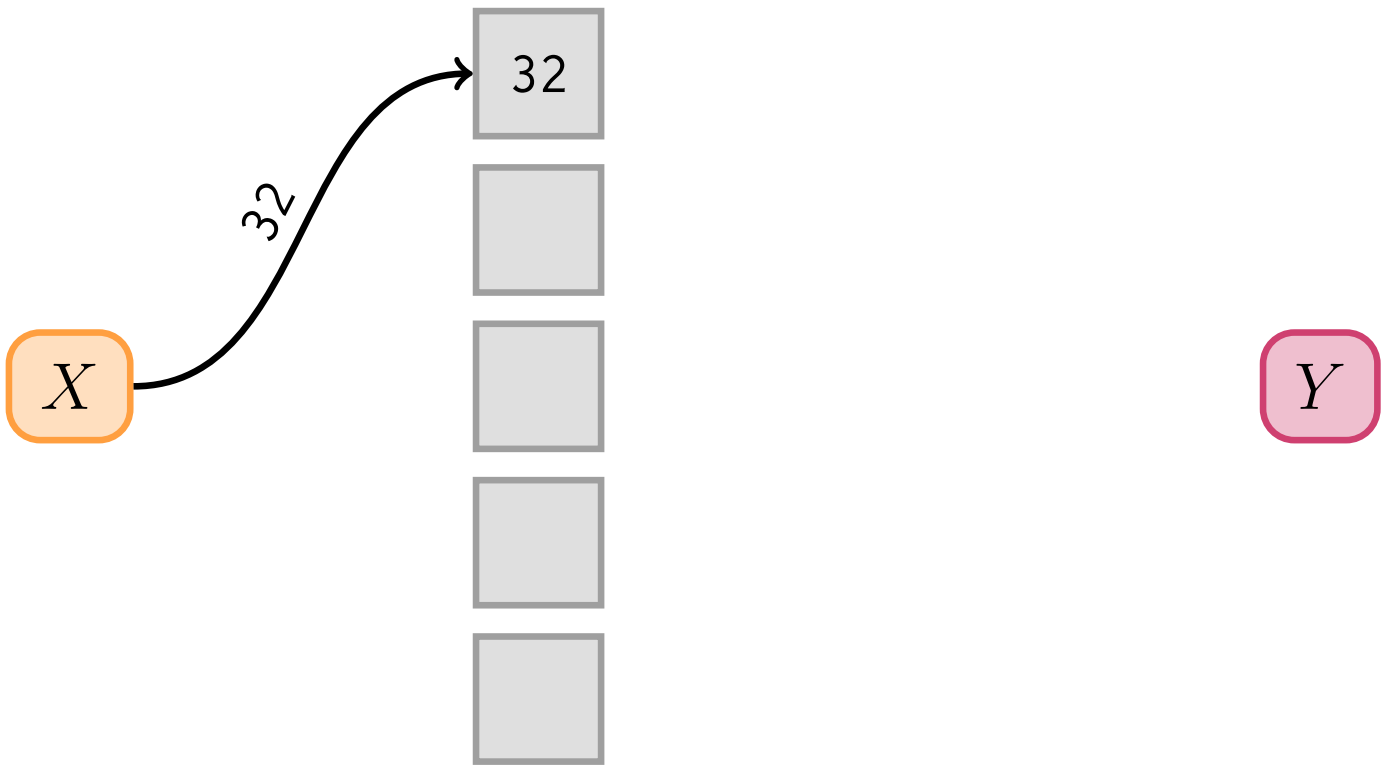
„Algorithmus M“ Generator[2, S.32][1, S.33]

Mit zwei Generatoren X und Y:



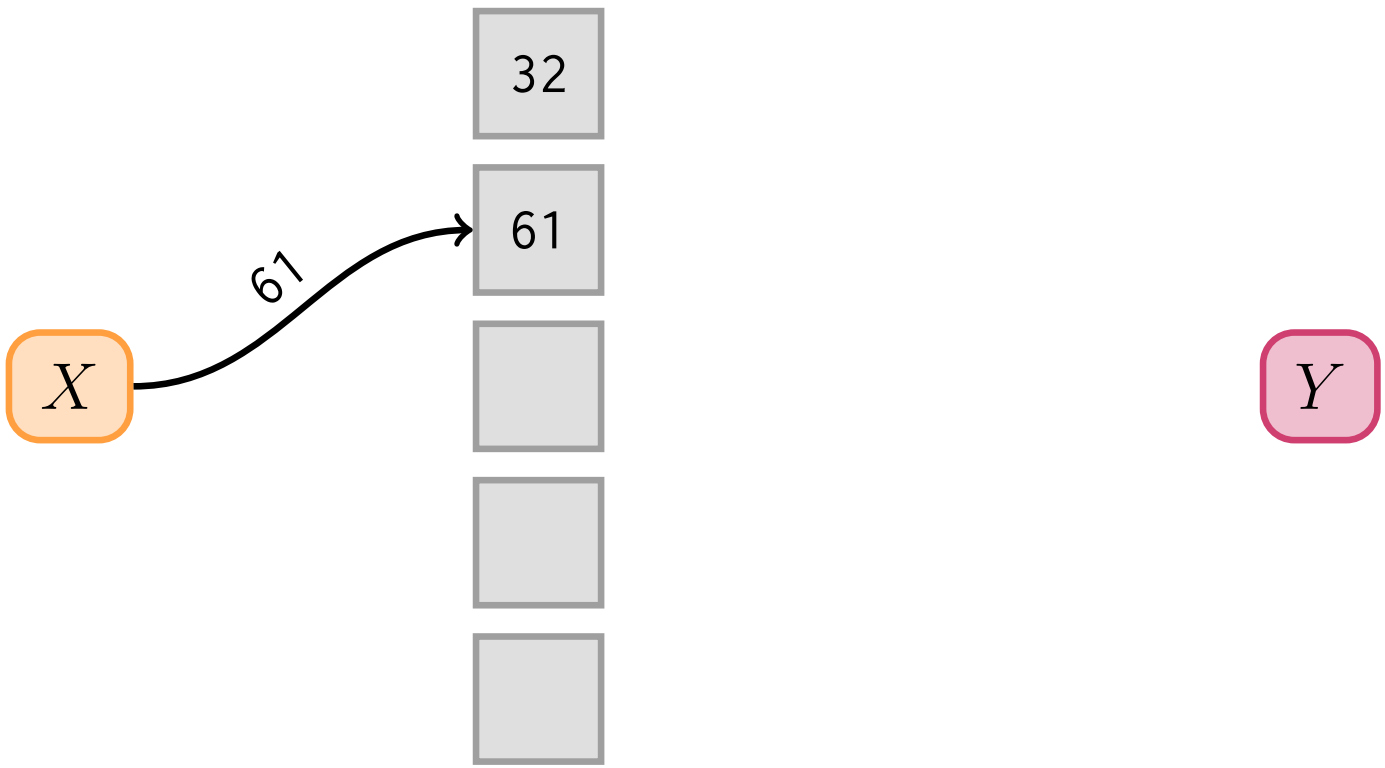
„Algorithmus M“ Generator[2, S.32][1, S.33]

Mit zwei Generatoren X und Y:



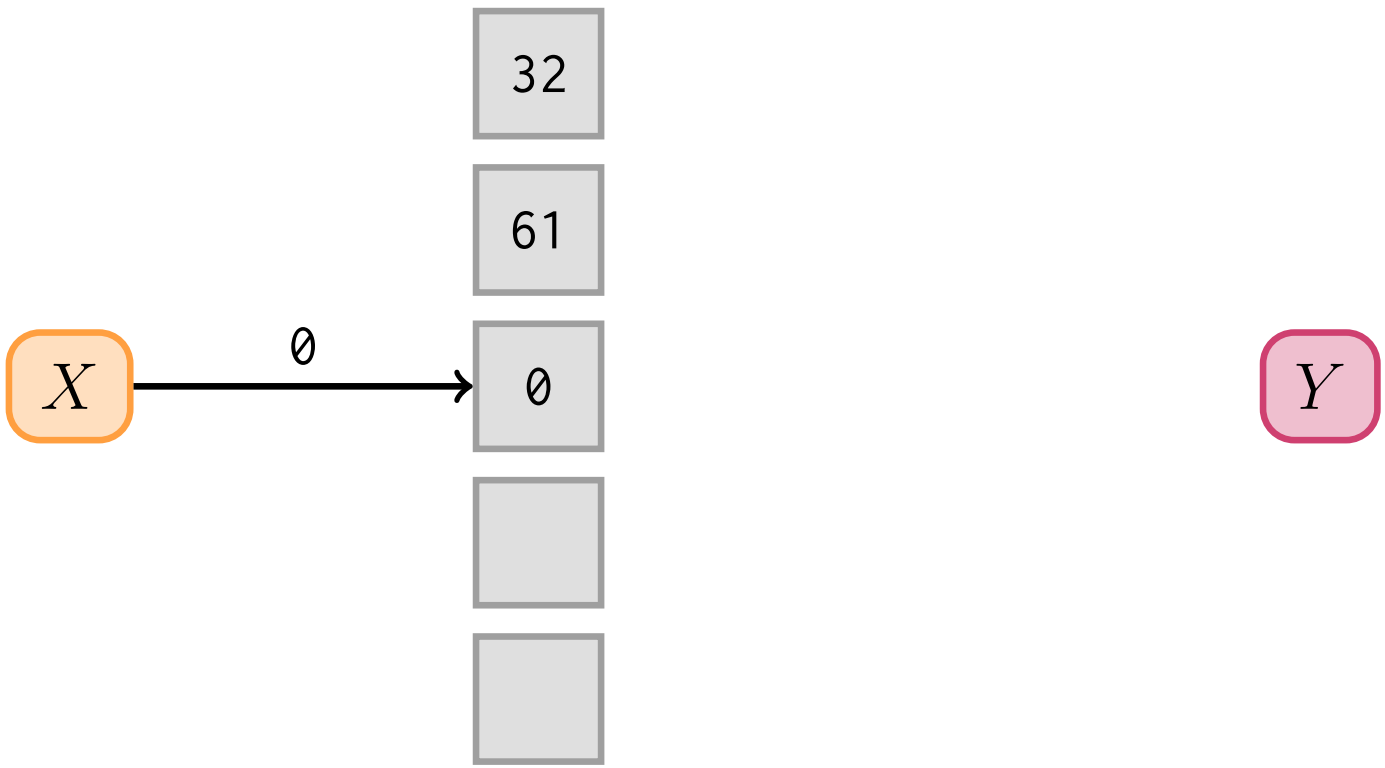
„Algorithmus M“ Generator[2, S.32][1, S.33]

Mit zwei Generatoren X und Y:



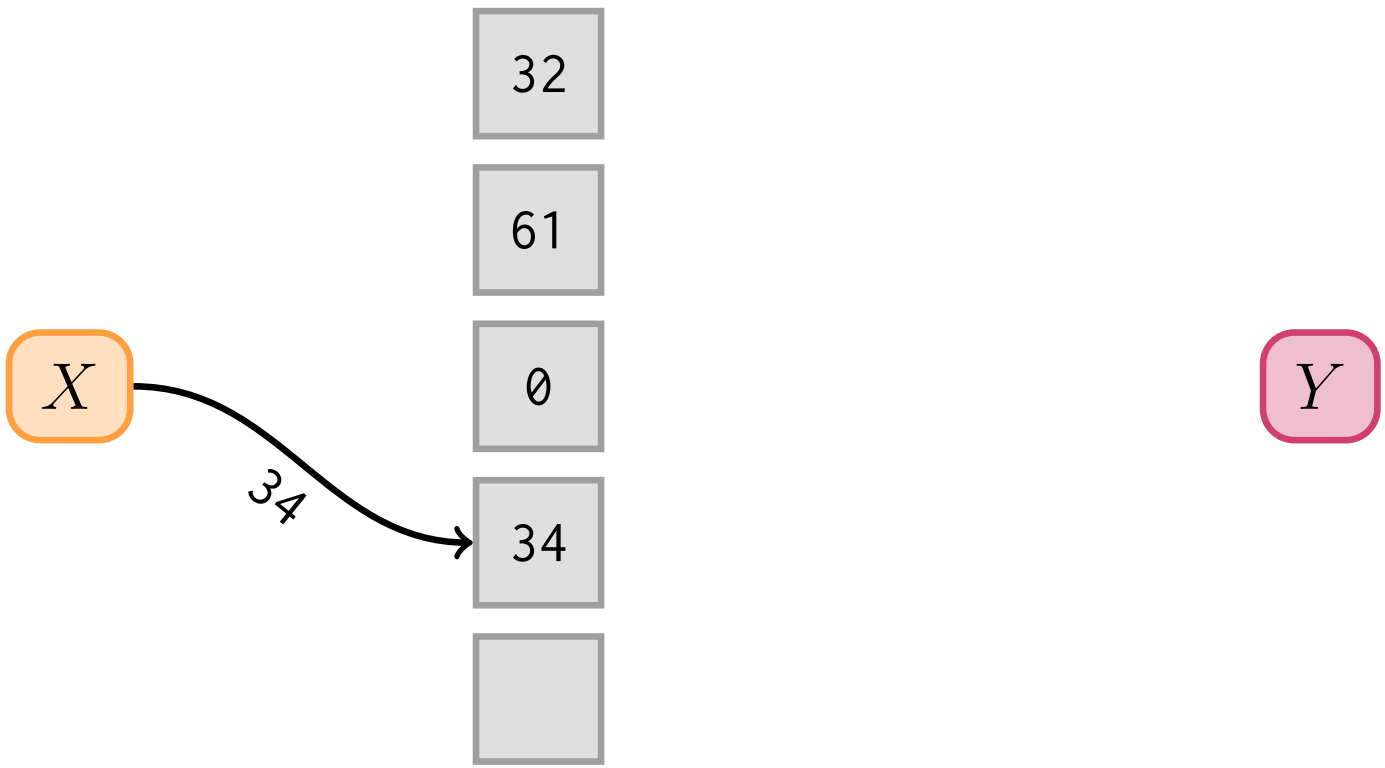
„Algorithmus M“ Generator[2, S.32][1, S.33]

Mit zwei Generatoren X und Y:



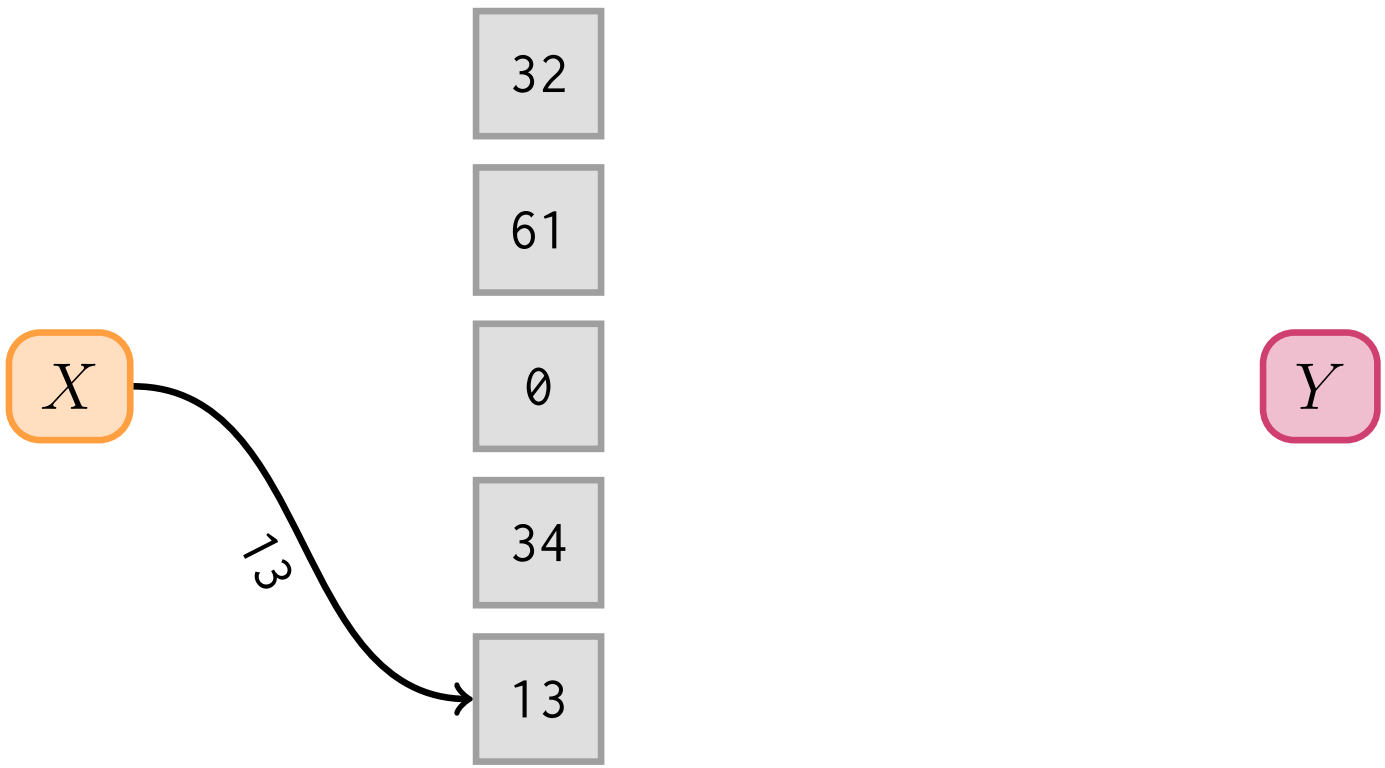
„Algorithmus M“ Generator[2, S.32][1, S.33]

Mit zwei Generatoren X und Y:



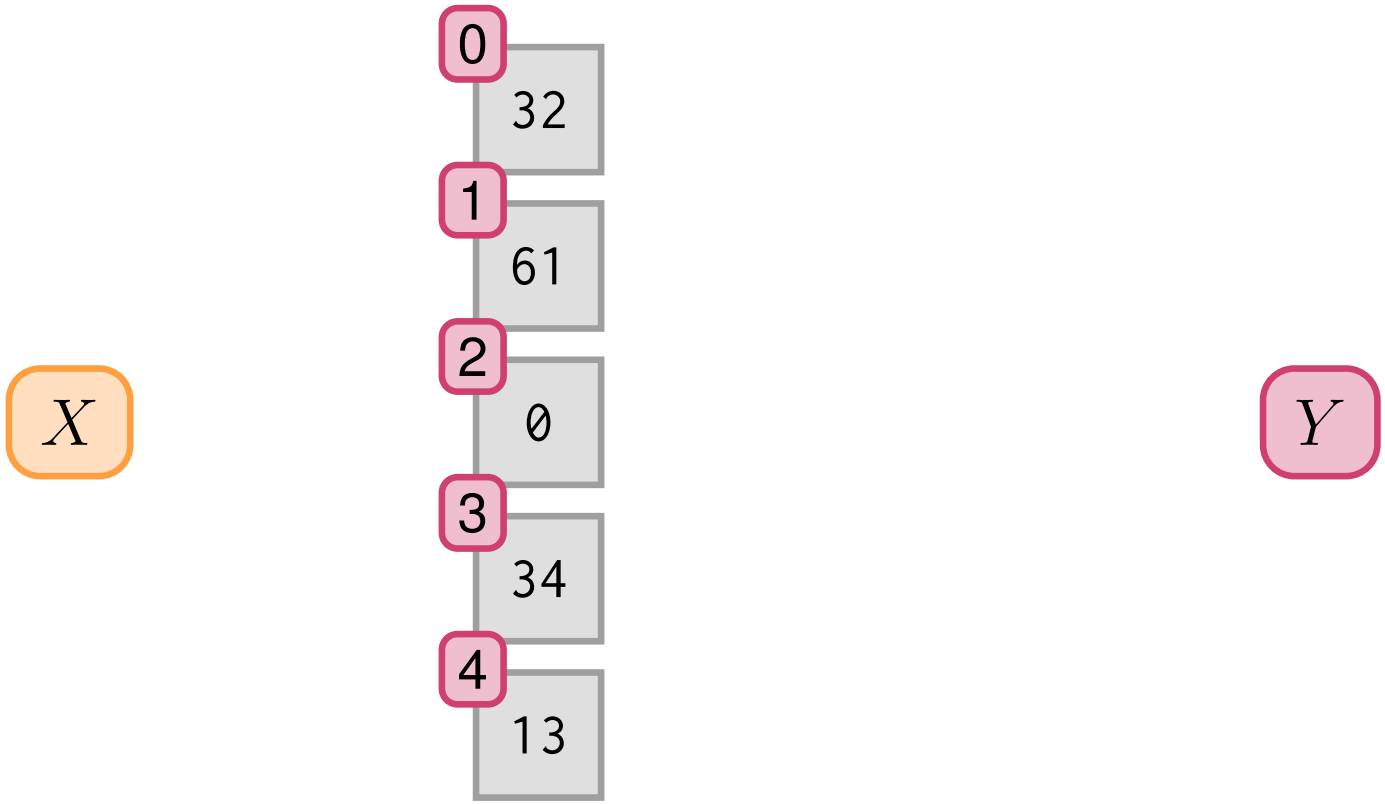
„Algorithmus M“ Generator[2, S.32][1, S.33]

Mit zwei Generatoren X und Y:



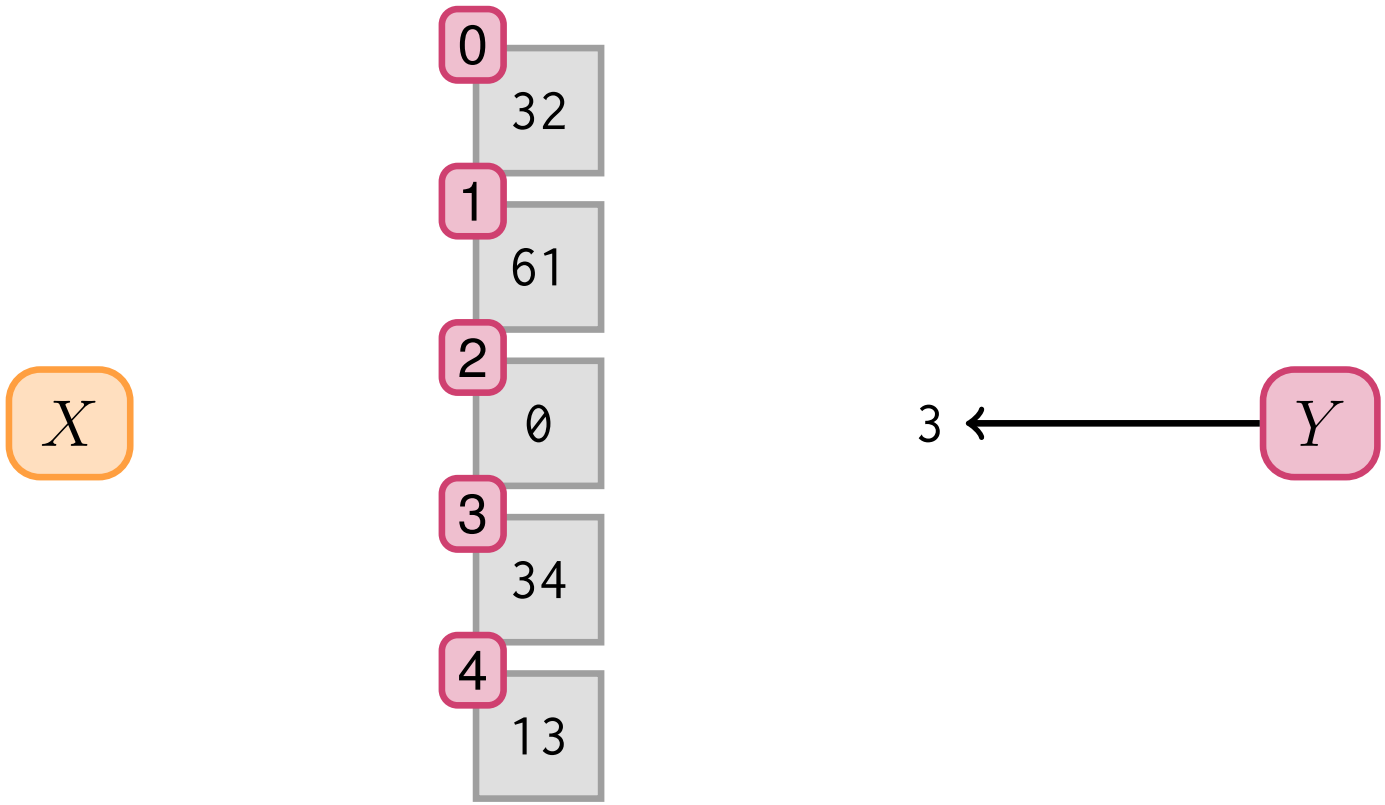
„Algorithmus M“ Generator[2, S.32][1, S.33]

Mit zwei Generatoren X und Y:



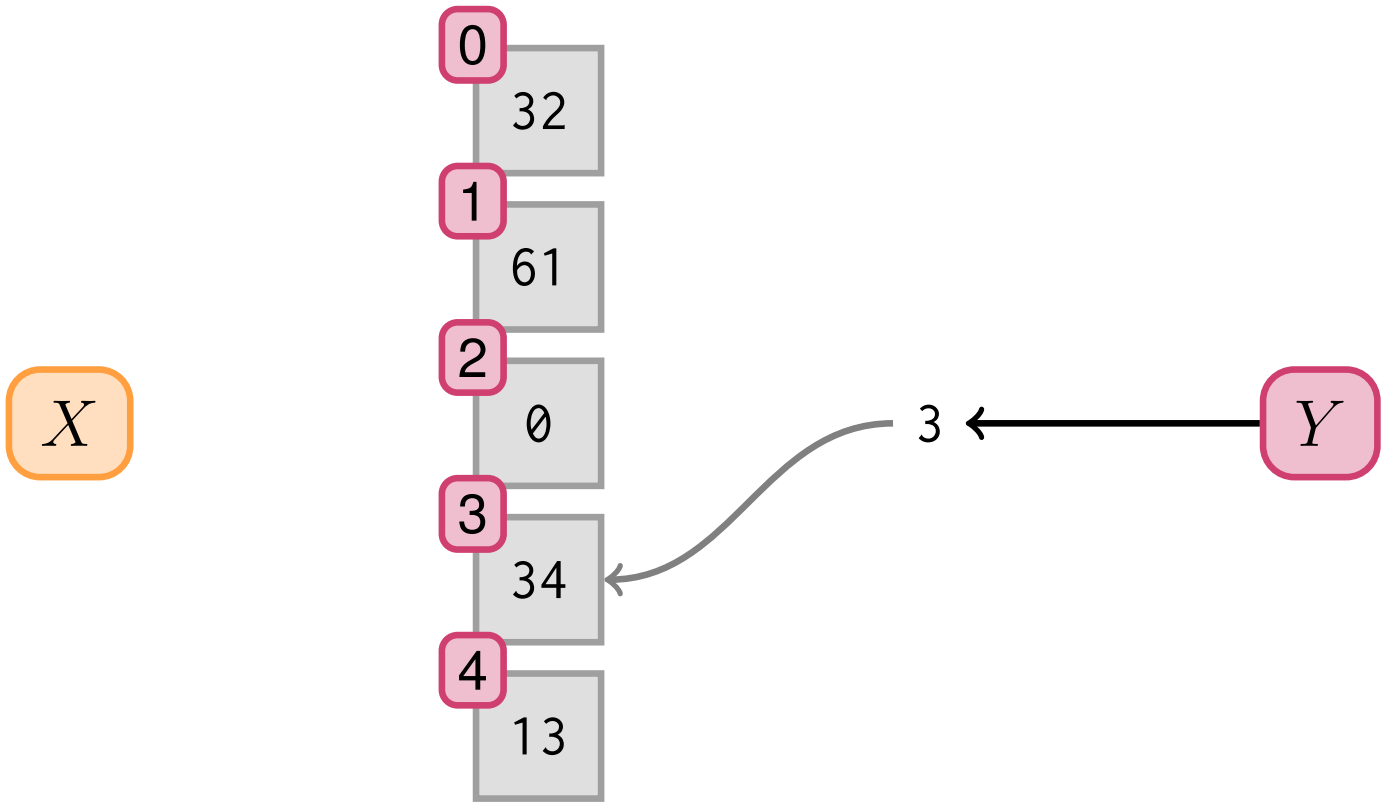
„Algorithmus M“ Generator[2, S.32][1, S.33]

Mit zwei Generatoren X und Y:



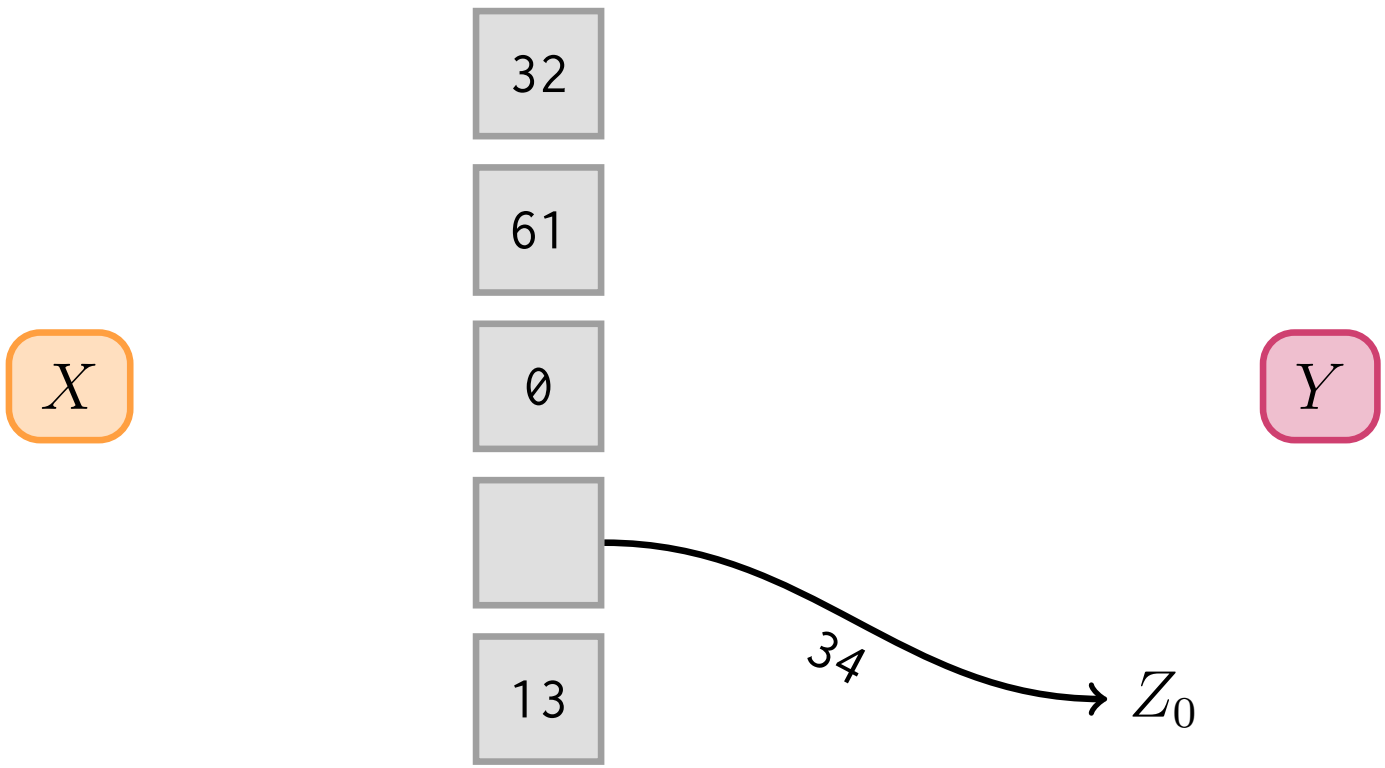
„Algorithmus M“ Generator[2, S.32][1, S.33]

Mit zwei Generatoren X und Y:



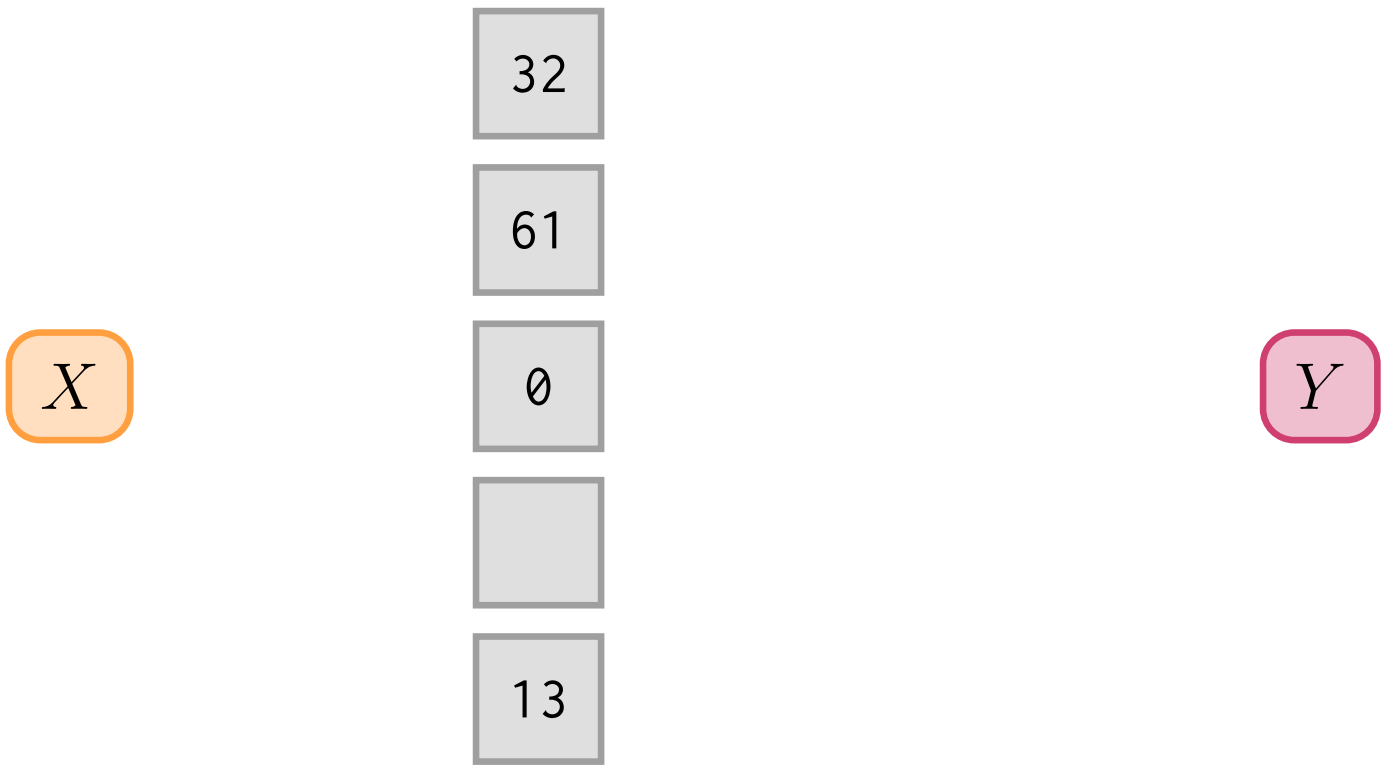
„Algorithmus M“ Generator[2, S.32][1, S.33]

Mit zwei Generatoren X und Y:



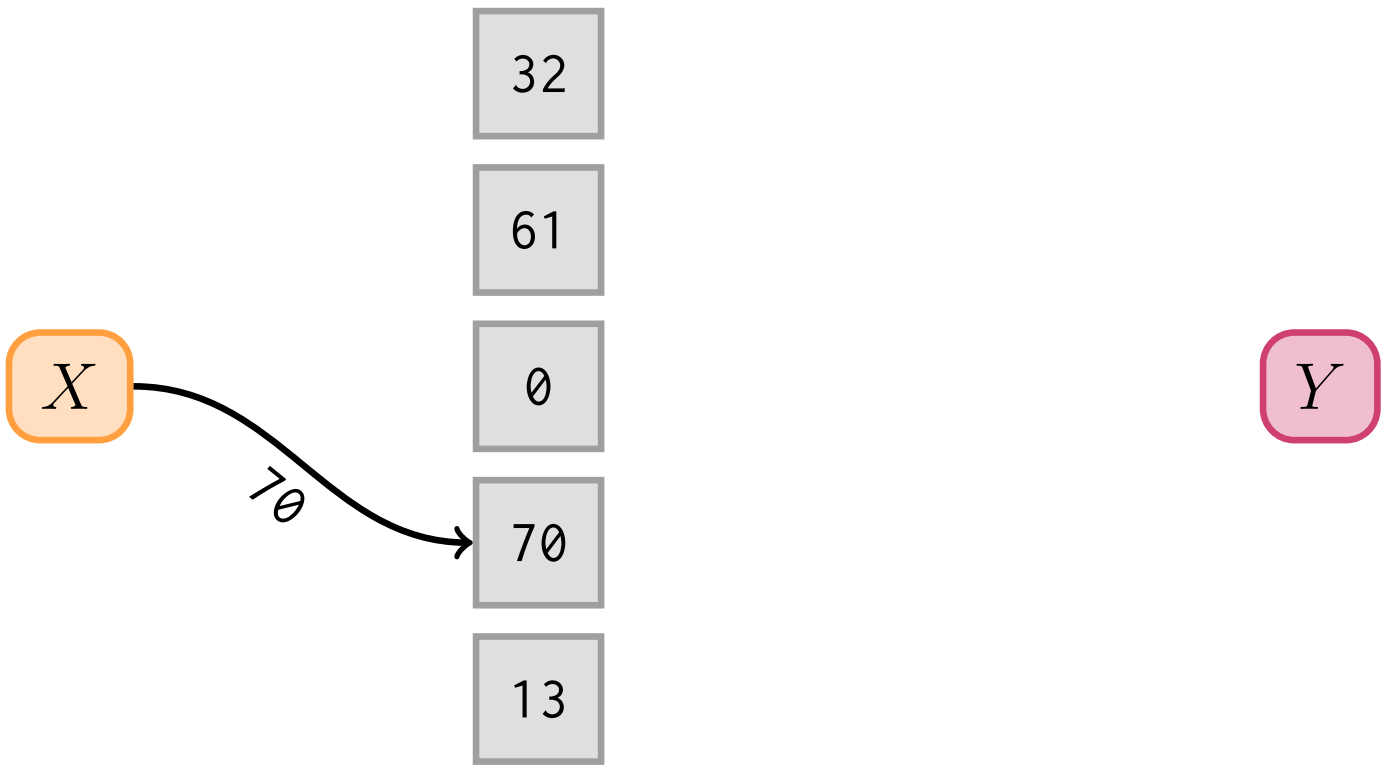
„Algorithmus M“ Generator[2, S.32][1, S.33]

Mit zwei Generatoren X und Y:



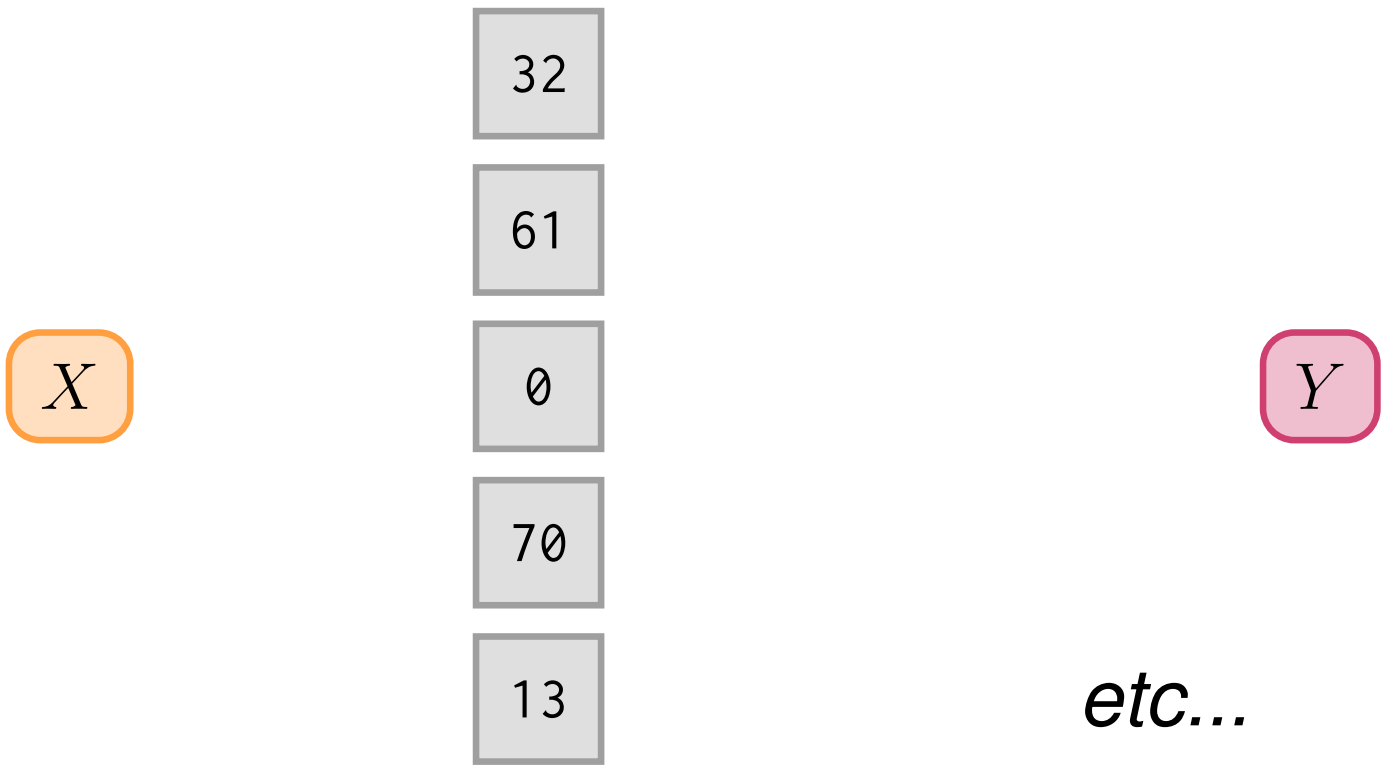
„Algorithmus M“ Generator[2, S.32][1, S.33]

Mit zwei Generatoren X und Y:



„Algorithmus M“ Generator[2, S.32][1, S.33]

Mit zwei Generatoren X und Y:



Algo. M Visualisierung I: Spektral-Methode-Test

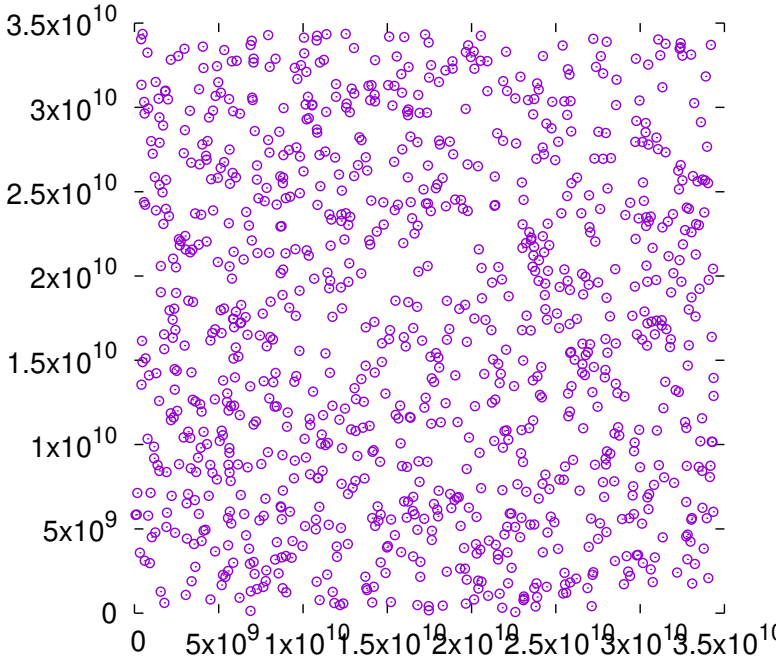


Abbildung: „Gut“

Mit

$$X_n = (3141592653 X_{n-1} + 2718281829) \mod 2^{25}$$
$$X_0 = 5772156649$$

$$Y_n = (2718281829 X_{n-1} + 3131592653) \mod 2^{35}$$
$$Y_0 = 1781072418$$

und $k = 64$.

Algo. M Visualisierung II: Monte-Carlo-Test

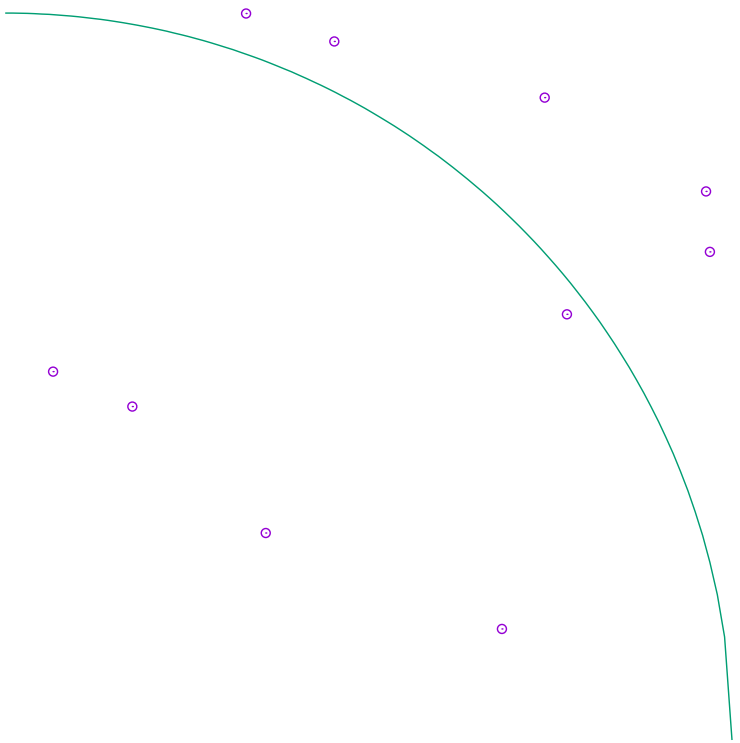


Abbildung: $n = 10$

Monte-Carlo Test zum Approximieren von π:

n	Treffer	Δ
10	5	1.14159265

Algo. M Visualisierung II: Monte-Carlo-Test

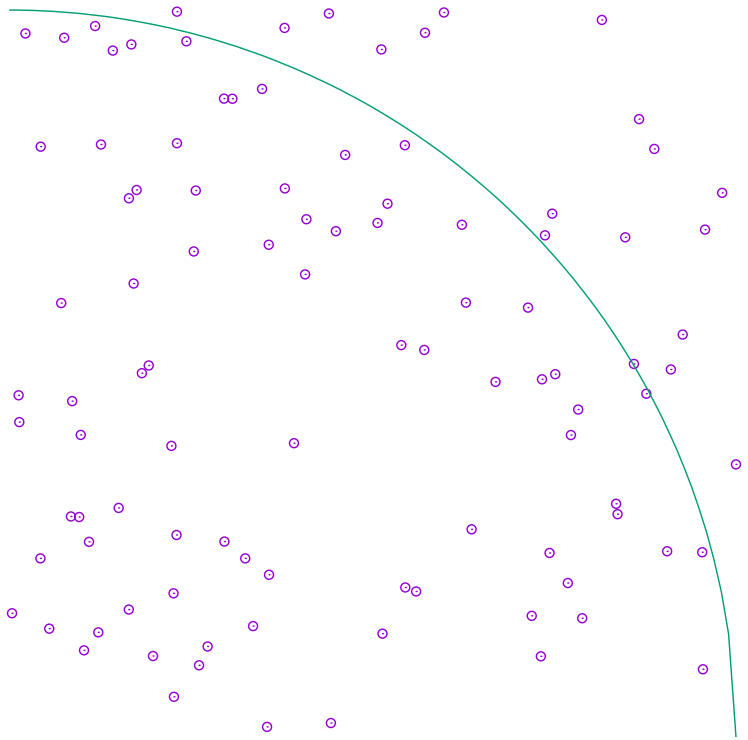


Abbildung: $n = 100$

Monte-Carlo Test zum Approximieren von π:

n	Treffer	Δ
10	5	1.14159265
100	82	0.13840734

Algo. M Visualisierung II: Monte-Carlo-Test

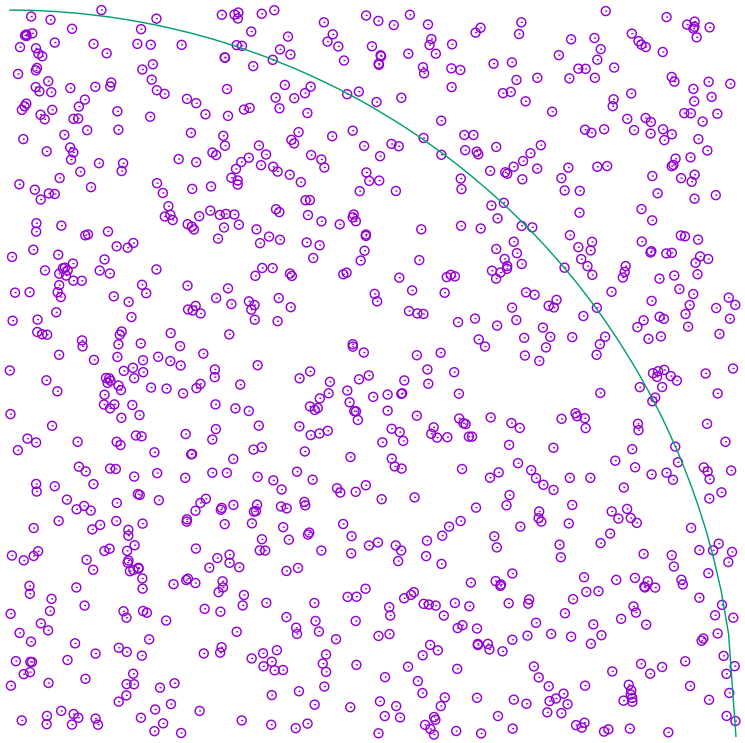


Abbildung: $n = 1000$

Monte-Carlo Test zum Approximieren von π:

n	Treffer	Δ
10	5	1.14159265
100	82	0.13840734
1000	788	0.0104073464

Algo. M Visualisierung II: Monte-Carlo-Test

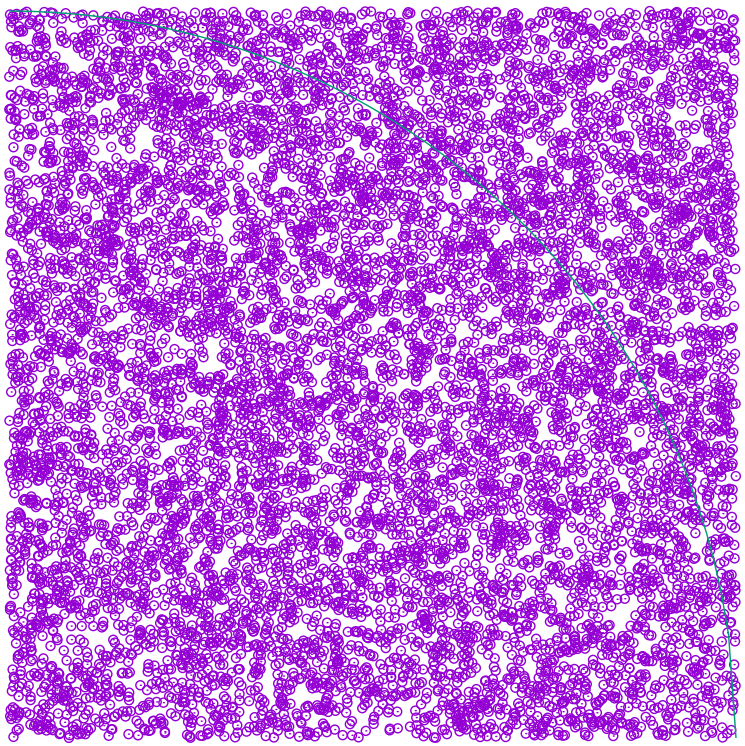


Abbildung: $n = 10000$

Monte-Carlo Test zum Approximieren von π:

n	Treffer	Δ
10	5	1.14159265
100	82	0.13840734
1000	788	0.0104073464
10000	7803	0.020392653

Algo. M Visualisierung II: Monte-Carlo-Test

Abbildung: $n = 100000$

Monte-Carlo Test zum Approximieren von π:

n	Treffer	Δ
10	5	1.14159265
100	82	0.13840734
1000	788	0.0104073464
10000	7803	0.020392653
100000	78692	0.0060873464

Algo. M Visualisierung III: Rausch-Test

RNG benutzen um zufällige Pixelwerte zu generieren:

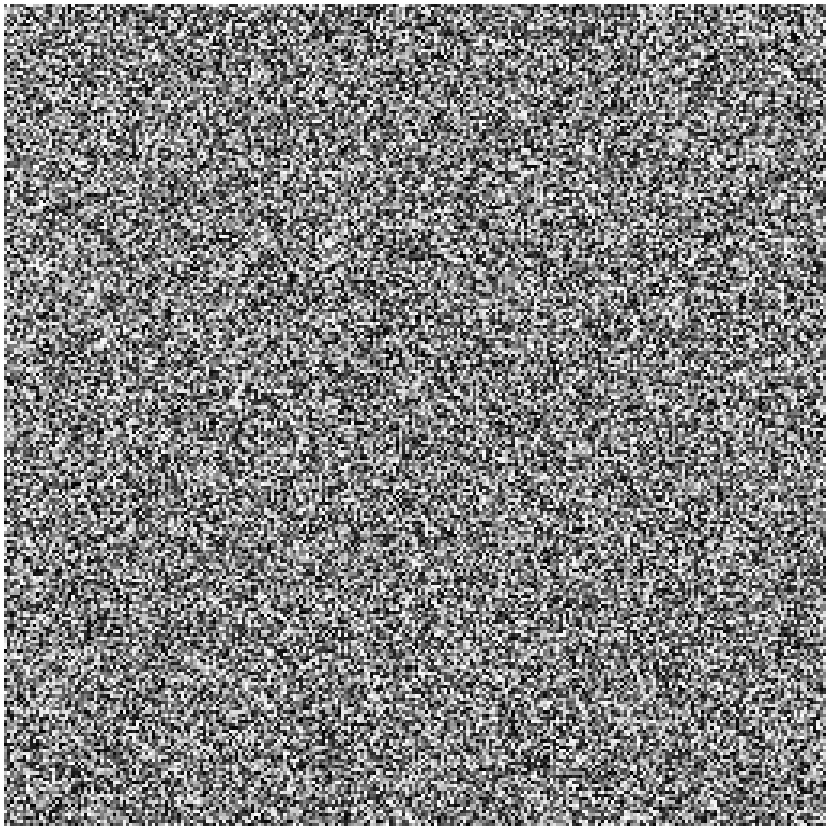


Abbildung: „Algorithmus M“ mit Parametern wie vorhin

Beobachtungen und Anmerkungen I

Beispiel für RNG „höherer Ordnung“, bzw
Randomness Enhancer

Beobachtungen und Anmerkungen II

In manchen fällen kann aber „Zufälligkeit“ abnehmen, wenn x und y zu ähnlich sind[2, S.33, Aufgabe 3][1, S.36, ibid].

Beobachtungen und Anmerkungen III

Periode wird KGV der Perioden von X und Y sein[2, S.33, Aufgabe 2][1, S.36, ibid].

Weitere Variationen auf dem bisher Gesehenem

Algorithm B.[2, S.32][1, S.34]

Analog zu *Algorithm M.*, nur mit einem RNG für das Füllen *und* auswählen.

Kombinierte RNGs[1, S.35]

$$Z_{n+1} = (X_n + Y_n) \bmod m$$

Dispensing Generator [1, S.35]

Auslassen von Werten eines anderen RNGs.

Acta Est Fabula

Almost all good computer programms contain at least one random-number generator.

D. Knuth, Seminumerical Algorithms, 1969[5, S.4]

[1] D. E. Knuth, *The Art of Computer Programming, Volume II: Seminumerical Algorithms, 3rd Edition*. Reading, Massachusetts: Addison-Wesley, 1998, ISBN: 0201896842. Adresse: http://www.worldcat.org/oclc/312898417.

[2] ——, *The Art of Computer Programming, Volume II: Seminumerical Algorithms, 2nd Edition*. Reading, Massachusetts: Addison-Wesley, 1981, ISBN: 0-201-03822-6.

[3] ——, *The Art of Computer Programming, Volume II: Seminumerical Algorithms*. Reading, Massachusetts: Addison-Wesley, 1969, ISBN: 0201038021. Adresse: http://www.worldcat.org/oclc/310551264.

[4] ——, *The Art of Computer Programming: Errata to Volume 2 (3rd Edition)*, Reading, Massachusetts, Jan. 2011. Adresse: https://cs.stanford.edu/~knuth/all2-pre.ps.gz.

[5] ——, *The Art of Computer Programming: Errata to Volume 2 (3rd Edition)*, after 2011, Reading, Massachusetts, Mai 2019. Adresse: https://cs.stanford.edu/~knuth/err2.ps.gz.

[6] I. Free Software Foundation, *Source code for* `random_r.c`, Adresse: https://sourceware.org/git/?p=glibc.git;a=blob;f=stdlib/random%5C_r.c;h=3292713641b6b19ffaa9785b52764c62678ee0c7;hb=HEAD.