Project

Gursharn Smruti Shwetab Sagnika Kaustav

Indian Statistical Institute

August 27, 2010

Random Numbers

- The word "random" means apparent absence of cause, planning, or design, "lack of method or system", or "accidental, haphazard."
- Statistically, it means inability to predict outcomes or to find any pattern in a series of outcomes.
- **A random number is a number generated by a process, whose outcome is unpredictable, and which cannot be subsequentially reliably reproduced.**

Random Numbers

- The word "random" means apparent absence of cause, planning, or design, "lack of method or system", or "accidental, haphazard."
- Statistically, it means inability to predict outcomes or to find any pattern in a series of outcomes.
- **A random number is a number generated by a process, whose outcome is unpredictable, and which cannot be subsequentially reliably reproduced.**

Random Numbers

- The word "random" means apparent absence of cause, planning, or design, "lack of method or system", or "accidental, haphazard."
- Statistically, it means inability to predict outcomes or to find any pattern in a series of outcomes.
- **A random number is a number generated by a process, whose outcome is unpredictable, and which cannot be subsequentially reliably reproduced.**

Random Numbers

- The word “random” means apparent absence of cause, planning, or design, “lack of method or system”, or “accidental, haphazard.”
- Statistically, it means inability to predict outcomes or to find any pattern in a series of outcomes.
- **A random number is a number generated by a process, whose outcome is unpredictable, and which cannot be subsequentially reliably reproduced.**

Difficulties

- Generating true random numbers involves precise, accurate and repeatable system measurements of non-deterministic processes.
- Such processes are almost unachievable in practice.
- That is why we try to seek an alternative in the form of pseudorandom numbers.

Difficulties

- Generating true random numbers involves precise, accurate and repeatable system measurements of non-deterministic processes.
- Such processes are almost unachievable in practice.
- That is why we try to seek an alternative in the form of pseudorandom numbers.

Difficulties

- Generating true random numbers involves precise, accurate and repeatable system measurements of non-deterministic processes.
- Such processes are almost unachievable in practice.
- That is why we try to seek an alternative in the form of pseudorandom numbers.

Difficulties

- Generating true random numbers involves precise, accurate and repeatable system measurements of non-deterministic processes.
- Such processes are almost unachievable in practice.
- That is why we try to seek an alternative in the form of pseudorandom numbers.

Pseudo Random number

- Psedurandom sequences are outcomes of a deterministic causal process.
- Psedurandom processes can be used repeatedly to produce exactly the same numbers.
- Pseudorandom number generators are algorithms that use mathematical formulae or simply precalculated tables to produce sequences of numbers that appear random. For example, linear congruential method.

Pseudo Random number

- Pseudrandom sequences are outcomes of a deterministic causal process.
- Pseudrandom processes can be used repeatedly to produce exactly the same numbers.
- Pseudorandom number generators are algorithms that use mathematical formulae or simply precalculated tables to produce sequences of numbers that appear random. For example, linear congruential method.

Pseudo Random number

- Psedurandom sequences are outcomes of a deterministic causal process.
- Psedurandom processes can be used repeatedly to produce exactly the same numbers.
- Pseudorandom number generators are algorithms that use mathematical formulae or simply precalculated tables to produce sequences of numbers that appear random. For example, linear congruential method.

Pseudo Random number

- Psedurandom sequences are outcomes of a deterministic causal process.
- Psedurandom processes can be used repeatedly to produce exactly the same numbers.
- Pseudorandom number generators are algorithms that use mathematical formulae or simply precalculated tables to produce sequences of numbers that appear random. For example, linear congruential method.

Characteristics of Pseudorandom numbers

- PRNGs are efficient i.e. they can produce many numbers in a short time.
- They are deterministic in the sense that a given sequence of numbers can be reproduced at a later stage if the starting point in the sequence is known.
- PRNGs are typically also periodic, which means that the sequence will eventually repeat itself.

Several computational methods for random number generation exist which lack true randomness.

Characteristics of Pseudorandom numbers

- PRNGs are efficient i.e. they can produce many numbers in a short time.
- They are deterministic in the sense that a given sequence of numbers can be reproduced at a later stage if the starting point in the sequence is known.
- PRNGs are typically also periodic, which means that the sequence will eventually repeat itself.

Several computational methods for random number generation exist which lack true randomness.

Characteristics of Pseudorandom numbers

- PRNGs are efficient i.e. they can produce many numbers in a short time.
- They are deterministic in the sense that a given sequence of numbers can be reproduced at a later stage if the starting point in the sequence is known.
- PRNGs are typically also periodic, which means that the sequence will eventually repeat itself.

Several computational methods for random number generation exist which lack true randomness.

Characteristics of Pseudorandom numbers

- PRNGs are efficient i.e. they can produce many numbers in a short time.
- They are deterministic in the sense that a given sequence of numbers can be reproduced at a later stage if the starting point in the sequence is known.
- PRNGs are typically also periodic, which means that the sequence will eventually repeat itself.

  Several computational methods for random number generation exist which lack true randomness.

History of Random Number Generators

- Rolling a die is a preliminary RNG which was being used in anciant times in the games of chance.
- The chinese were perhaps the earliest people to formalize odds and chance 3000 years ago.
- In the sixteenth century that Italian mathematicians began to formalize the odds associated with various games of chance, like Bingo.
- Brownian motion arises from the aggregated effect of the random collsions of many molecules with suspended objects. Robert Brown claimed that it is one of very few that truely can not be predicted or is truely random. But in 1905 Albert Einstein suggested that this is not really random.

History of Random Number Generators

- Rolling a die is a preliminary RNG which was being used in ancient times in the games of chance.
- The chinese were perhaps the earliest people to formalize odds and chance 3000 years ago.
- In the sixteenth century that Italian mathematicians began to formalize the odds associated with various games of chance, like Bingo.
- Brownian motion arises from the aggregated effect of the random collsions of many molecules with suspended objects. Robert Brown claimed that it is one of very few that truely can not be predicted or is truely random. But in 1905 Albert Einstein suggested that this is not really random.

History of Random Number Generators

- Rolling a die is a preliminary RNG which was being used in anciant times in the games of chance.
- The Chinese were perhaps the earliest people to formalize odds and chance 3000 years ago.
- In the sixteenth century that Italian mathematicians began to formalize the odds associated with various games of chance, like Bingo.
- Brownian motion arises from the aggregated effect of the random collsions of many molecules with suspended objects. Robert Brown claimed that it is one of very few that truely can not be predicted or is truely random. But in 1905 Albert Einstein suggested that this is not really random.

History of Random Number Generators

- Rolling a die is a preliminary RNG which was being used in anciant times in the games of chance.
- The chinese were perhaps the earliest people to formalize odds and chance 3000 years ago.
- In the sixteenth century that Italian mathematicians began to formalize the odds associated with various games of chance, like Bingo.
- Brownian motion arises from the aggregated effect of the random collsions of many molecules with suspended objects. Robert Brown claimed that it is one of very few that truely can not be predicted or is truely random. But in 1905 Albert Einstein suggested that this is not really random.

History of Random Number Generators

- Rolling a die is a preliminary RNG which was being used in anciant times in the games of chance.
- The chinese were perhaps the earliest people to formalize odds and chance 3000 years ago.
- In the sixteenth century that Italian mathematicians began to formalize the odds associated with various games of chance, like Bingo.
- Brownian motion arises from the aggregated effect of the random collsions of many molecules with suspended objects. Robert Brown claimed that it is one of very few that truely can not be predicted or is truely random. But in 1905 Albert Einstein suggested that this is really random.

Popular RNGs

- **Wichmann-Hill RNG:**
  - Invented in 1982.
  - Periodicity: 6953607871644
- **Super-Duper:**
  - Invented in 1970.
  - Periodicity 4.6×10^{18}
  - Failed in MTUPLE test.
- **Marsaglia-Multicarry:**
  - Invented round 1985.
  - Periodicity: More than 2^{60}

Popular RNGs

- **Wichmann-Hill RNG:**
  - Invented in 1982.
  - Periodicity: 6953607871644
- **Super-Duper:**
  - Invented in 1970.
  - Periodicity 4.6×10^{18}
  - Failed in MTUPLE test.
- **Marsaglia-Multicarry:**
  - Invented round 1985.
  - Periodicity: More than 2^{60}

Popular RNGs

- **Wichmann-Hill RNG:**
  - * Invented in 1982.
  - * Periodicity: 6953607871644
- **Super-Duper:**
  - * Invented in 1970.
  - * Periodicity 4.6×10^{18}
  - * Failed in MTUPLE test.
- **Marsaglia-Multicarry:**
  - * Invented round 1985.
  - * Periodicity: More than 2^{60}

Popular RNGs

- **Wichmann-Hill RNG:**
  * Invented in 1982.
  * Periodicity: 6953607871644
- **Super-Duper:**
  * Invented in 1970.
  * Periodicity 4.6×10^{18}
  * Failed in MTUPLE test.
- **Marsaglia-Multicarry:**
  * Invented round 1985.
  * Periodicity: More than 2^{60}

Popular RNGs

- **Wichmann-Hill RNG:**
  - Invented in 1982.
  - Periodicity: 6953607871644
- **Super-Duper:**
  - Invented in 1970.
  - Periodicity 4.6×10^{18}
  - Failed in MTUPLE test.
- **Marsaglia-Multicarry:**
  - Invented round 1985.
  - Periodicity: More than 2^{60}

Popular RNGs

- **Wichmann-Hill RNG:**
  - * Invented in 1982.
  - * Periodicity: 6953607871644
- **Super-Duper:**
  - * Invented in 1970.
  - * Periodicity 4.6×10^{18}
  - * Failed in MTUPLE test.
- **Marsaglia-Multicarry:**
  - * Invented round 1985.
  - * Periodicity: More than 2^{60}

Popular RNGs

- **Wichmann-Hill RNG:**
  - Invented in 1982.
  - Periodicity: 6953607871644
- **Super-Duper:**
  - Invented in 1970.
  - Periodicity 4.6×10^{18}
  - Failed in MTUPLE test.
- **Marsaglia-Multicarry:**
  - Invented round 1985.
  - Periodicity: More than 2^{60}

Popular RNGs

- **Wichmann-Hill RNG:**
  * Invented in 1982.
  * Periodicity: 6953607871644
- **Super-Duper:**
  * Invented in 1970.
  * Periodicity 4.6×10^{18}
  * Failed in MTUPLE test.
- **Marsaglia-Multicarry:**
  * Invented round 1985.
  * Periodicity: More than 2^{60}

Popular RNGs

- **Wichmann-Hill RNG:**
  - Invented in 1982.
  - Periodicity: 6953607871644
- **Super-Duper:**
  - Invented in 1970.
  - Periodicity 4.6×10^{18}
  - Failed in MTUPLE test.
- **Marsaglia-Multicarry:**
  - Invented round 1985.
  - Periodicity: More than 2^{60}

Popular RNGs

- **Wichmann-Hill RNG:**
  - * Invented in 1982.
  - * Periodicity: 6953607871644
- **Super-Duper:**
  - * Invented in 1970.
  - * Periodicity 4.6×10^{18}
  - * Failed in MTUPLE test.
- **Marsaglia-Multicarry:**
  - * Invented round 1985.
  - * Periodicity: More than 2^{60}

Popular RNGs

- **Mersenne Twisler RNG:**
  - * Invented by Matsumoto and Nishimura(1998).
  - * Periodicity: $2^{19937} - 1$.
- **Knuth TAOCP-2002:** A 32 bits integer GFSR using lagged Fibonacci sequence with substractiuon.

$$X[j] = (X[j-100] - X[j-37])\,\mathbf{mod}\,2^{32}$$

  - * Periodicity: 2^{129}

Popular RNGs

- **Mersenne Twisler RNG:**
  - * Invented by Matsumoto and Nishimura(1998).
  - * Periodicity: $2^{19937} - 1$.
- **Knuth TAOCP-2002:** A 32 bits integer GFSR using lagged Fibonacci sequence with substractiuon.

$$X[j] = (X[j-100] - X[j-37])\mathbf{mod} 2^{32}$$

  - * Periodicity: 2^{129}

Popular RNGs

- **Mersenne Twisler RNG:**
  - * Invented by Matsumoto and Nishimura(1998).
  - * Periodicity: $2^{19937} - 1$.
- **Knuth TAOCP-2002:** A 32 bits integer GFSR using lagged Fibonacci sequence with substractiuon.

$$X[j] = (X[j-100] - X[j-37])\mathbf{mod} 2^{32}$$

  - * Periodicity: 2^{129}

Popular RNGs

- **Mersenne Twisler RNG:**
  - * Invented by Matsumoto and Nishimura(1998).
  - * Periodicity: $2^{19937} - 1$.
- **Knuth TAOCP-2002:** A 32 bits integer GFSR using lagged Fibonacci sequence with substractiuon.

$$X[j] = (X[j-100] - X[j-37]) \mathbf{mod} 2^{32}$$

  - * Periodicity: 2^{129}

Popular RNGs

- **Mersenne Twisler RNG:**
  - * Invented by Matsumoto and Nishimura(1998).
  - * Periodicity: $2^{19937} - 1$.
- **Knuth TAOCP-2002:** A 32 bits integer GFSR using lagged Fibonacci sequence with substractiuon.

$$X[j] = (X[j-100] - X[j-37]) \mathbf{mod} 2^{32}$$

  - * Periodicity: 2^{129}

Popular RNGs

- **Mersenne Twisler RNG:**
  - Invented by Matsumoto and Nishimura(1998).
  - Periodicity: $2^{19937} - 1$.
- **Knuth TAOCP-2002:** A 32 bits integer GFSR using lagged Fibonacci sequence with substractiuon.

$$X[j] = (X[j-100] - X[j-37])\mathbf{mod} 2^{32}$$

  - Periodicity: 2^{129}

“Anyone who considers arithmetical methods of producing random digits is, ofcourse, in a state of sin.”

– Von Neumann

Von Neumann method

- John Von Neumann suggested the middle square method in about 1946.
- It is a computer based PRNG.
- Iterating Von Neumann's procedure produces a series of numbers generated by a deterministic process intended merely to imitate a random sequence.
- The procedure is:
  1. Take any integer number of n digits.
  2. Square it.
  3. Take the middle n digits of the resulting number as the random number.

Von Neumann method

- John Von Neumann suggested the middle square method in about 1946.
- It is a computer based PRNG.
- Iterating Von Neumann's procedure produces a series of numbers generated by a deterministic process intended merely to imitate a random sequence.
- The procedure is:
  1. Take any integer number of n digits.
  2. Square it.
  3. Take the middle n digits of the resulting number as the random number.

Von Neumann method

- John Von Neumann suggested the middle square method in about 1946.
- It is a computer based PRNG.
- Iterating Von Neumann's procedure produces a series of numbers generated by a deterministic process intended merely to imitate a random sequence.
- The procedure is:
  1. Take any integer number of n digits.
  2. Square it.
  3. Take the middle n digits of the resulting number as the random number.

Von Neumann method

- John Von Neumann suggested the middle square method in about 1946.
- It is a computer based PRNG.
- Iterating Von Neumann's procedure produces a series of numbers generated by a deterministic process intended merely to imitate a random sequence.
- The procedure is:
  1. Take any integer number of n digits.
  2. Square it.
  3. Take the middle n digits of the resulting number as the random number.

Von Neumann method

- John Von Neumann suggested the middle square method in about 1946.
- It is a computer based PRNG.
- Iterating Von Neumann's procedure produces a series of numbers generated by a deterministic process intended merely to imitate a random sequence.
- The procedure is:
  1. Take any integer number of n digits.
  2. Square it.
  3. Take the middle n digits of the resulting number as the random number.

Von Neumann method

- John Von Neumann suggested the middle square method in about 1946.
- It is a computer based PRNG.
- Iterating Von Neumann's procedure produces a series of numbers generated by a deterministic process intended merely to imitate a random sequence.
- The procedure is:
  1. Take any integer number of n digits.
  2. Square it.
  3. Take the middle n digits of the resulting number as the random number.

Von Neumann method

- John Von Neumann suggested the middle square method in about 1946.
- It is a computer based PRNG.
- Iterating Von Neumann's procedure produces a series of numbers generated by a deterministic process intended merely to imitate a random sequence.
- The procedure is:
  1. Take any integer number of n digits.
  2. Square it.
  3. Take the middle n digits of the resulting number as the random number.

Von Neumann method

- Since there are at most only 10^n different numbers, eventually the whole sequence repeats in the same order.
- The method often reaches a fixed point 0. Once it reaches 0, it stays there for ever.

Von Neumann method

- Since there are at most only 10^n different numbers, eventually the whole sequence repeats in the same order.
- The method often reaches a fixed point 0. Once it reaches 0, it stays there for ever.

Code for Von Neumann Algorithm

- Von Neumann Algorithm is carried out in two ways, viz, direct method and using bitwise shift operator.
- Direct method is implemented both in R and in C.
- Needless to say that Bitwise shift method is implemented using C.
- We try to get some idea of the randomness of the generated random numbers using graphs.

Code for Von Neumann Algorithm

- Von Neumann Algorithm is carried out in two ways, viz, direct method and using bitwise shift operator.
- Direct method is implemented both in R and in C.
- Needless to say that Bitwise shift method is implemented using C.
- We try to get some idea of the randomness of the generated random numbers using graphs.

Code for Von Neumann Algorithm

- Von Neumann Algorithm is carried out in two ways, viz, direct method and using bitwise shift operator.
- Direct method is implemented both in R and in C.
- Needless to say that Bitwise shift method is implemented using C.
- We try to get some idea of the randomness of the generated random numbers using graphs.

Code for Von Neumann Algorithm

- Von Neumann Algorithm is carried out in two ways, viz, direct method and using bitwise shift operator.
- Direct method is implemented both in R and in C.
- Needless to say that Bitwise shift method is implemented using C.
- We try to get some idea of the randomness of the generated random numbers using graphs.

CODES

(in HTML)

Preliminary Analysis based on Graphs

- The occurence of the digits in the generated random numbers should be random.
- If it follows any pattern then we can say that the appearance of the digit is non-random.
- If the appearance of the digit falls in the particular sequence of the duration of successive occurences then the same conclusion holds.
- If the last time a particular digit appears is too small then we conclude that, that digit never comes later on and hence there exists a non-random pattern in its occurence.

Preliminary Analysis based on Graphs

- The occurence of the digits in the generated random numbers should be random.
- If it follows any pattern then we can say that the appearance of the digit is non-random.
- If the appearance of the digit falls in the particular sequence of the duration of successive occurences then the same conclusion holds.
- If the last time a particular digit appears is too small then we conclude that, that digit never comes later on and hence there exists a non-random pattern in its occurence.

Preliminary Analysis based on Graphs

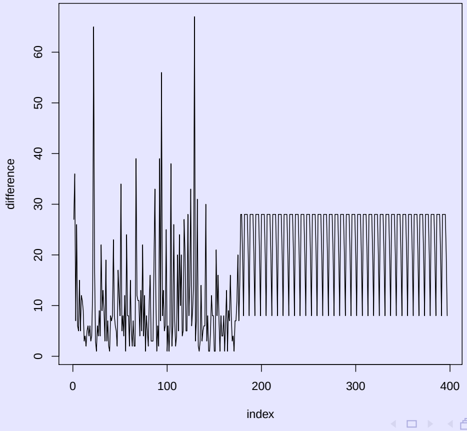
- The occurence of the digits in the generated random numbers should be random.
- If it follows any pattern then we can say that the appearance of the digit is non-random.
- If the appearance of the digit falls in the particular sequence of the duration of successive occurences then the same conclusion holds.
- If the last time a particular digit appears is too small then we conclude that, that digit never comes later on and hence there exists a non-random pattern in its occurence.

Preliminary Analysis based on Graphs

- The occurence of the digits in the generated random numbers should be random.
- If it follows any pattern then we can say that the appearance of the digit is non-random.
- If the appearance of the digit falls in the particular sequence of the duration of successive occurences then the same conclusion holds.
- If the last time a particular digit appears is too small then we conclude that, that digit never comes later on and hence there exists a non-random pattern in its occurence.

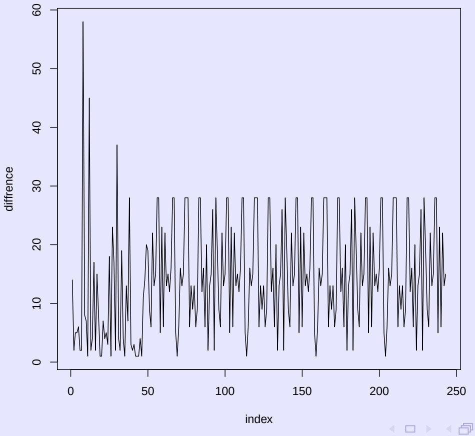
Duration of successive appearances of 4

- Graph 1



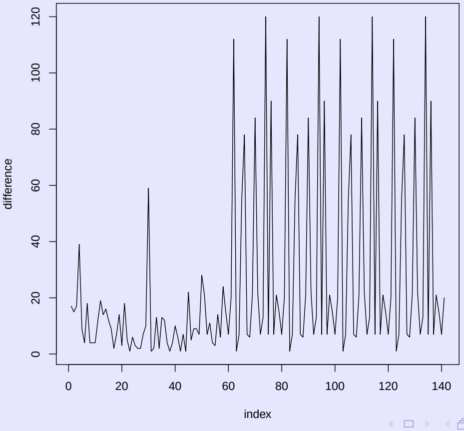
Duration of successive appearances of 2

- Graph 2



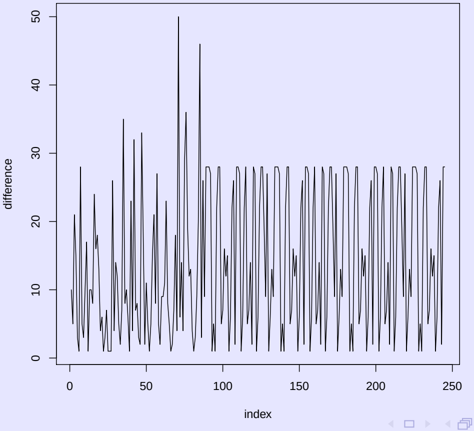
Duration of successive appearances of 3

- Graph 3



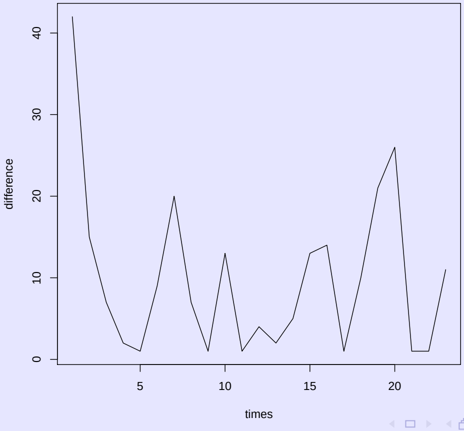
Duration of successive appearances of 4

- Graph 4



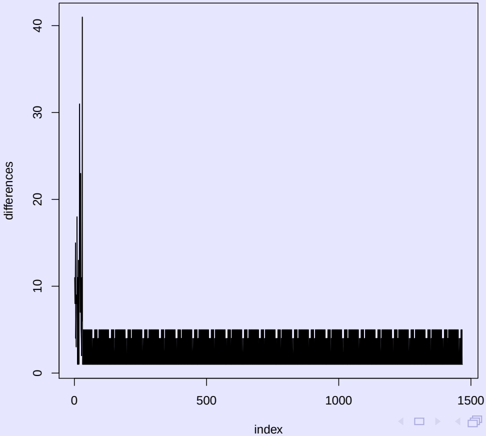
Duration of successive appearances of 4

- Graph 5



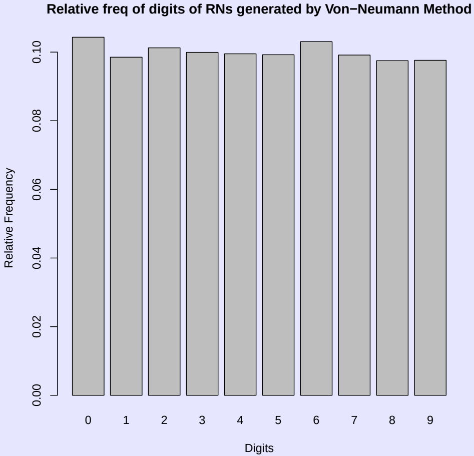
Duration of successive appearances of 0

- Graph 6



Relative Frequency of digits

- Graph 7



Observations

- In the first four graphs we see that the sequence of durations of successive occurances falls in a loop.
- In the graph 5 we see that the digit 4 has appeared very few times.
- Infact 4 appears at 249 th position for the last time, but there are total $3500(500 \times 7)$ random digits.
- In the graph 6 we see that the digit 0 is appearing very frequently though the sequence is not degenerated at 0.
- The last graph shows that in long run the digits are coming almost with equal probability.

Observations

- In the first four graphs we see that the sequence of durations of successive occurances falls in a loop.
- In the graph 5 we see that the digit 4 has appeared very few times.
- Infact 4 appears at 249 th position for the last time, but there are total $3500(500 \times 7)$ random digits.
- In the graph 6 we see that the digit 0 is appearing very frequently though the sequence is not degenerated at 0.
- The last graph shows that in long run the digits are coming almost with equal probability.

Observations

- In the first four graphs we see that the sequence of durations of successive occurances falls in a loop.
- In the graph 5 we see that the digit 4 has appeared very few times.
- Infact 4 appears at 249 th position for the last time, but there are total $3500(500 \times 7)$ random digits.
- In the graph 6 we see that the digit 0 is appearing very frequently though the sequence is not degenerated at 0.
- The last graph shows that in long run the digits are coming almost with equal probability.

Observations

- In the first four graphs we see that the sequence of durations of successive occurances falls in a loop.
- In the graph 5 we see that the digit 4 has appeared very few times.
- Infact 4 appears at 249 th position for the last time, but there are total $3500(500 \times 7)$ random digits.
- In the graph 6 we see that the digit 0 is appearing very frequently though the sequence is not degenerated at 0.
- The last graph shows that in long run the digits are coming almost with equal probability.

Observations

- In the first four graphs we see that the sequence of durations of successive occurances falls in a loop.
- In the graph 5 we see that the digit 4 has appeared very few times.
- Infact 4 appears at 249 th position for the last time, but there are total $3500(500 \times 7)$ random digits.
- In the graph 6 we see that the digit 0 is appearing very frequently though the sequence is not degenerated at 0.
- The last graph shows that in long run the digits are coming almost with equal probability.

Bitwise Programming

- In Bitwise programming a random number between 0 and $2^n - 1$ is chosen.
- Then it is squared.
- Next, appropriate number of left and right bitwise shifts give the middle digits.
- Finally the output is given as a decimal number.

Bitwise Programming

- In Bitwise programming a random number between 0 and $2^n - 1$ is chosen.
- Then it is squared.
- Next, appropriate number of left and right bitwise shifts give the middle digits.
- Finally the output is given as a decimal number.

Bitwise Programming

- In Bitwise programming a random number between 0 and $2^n - 1$ is chosen.
- Then it is squared.
- Next, appropriate number of left and right bitwise shifts give the middle digits.
- Finally the output is given as a decimal number.

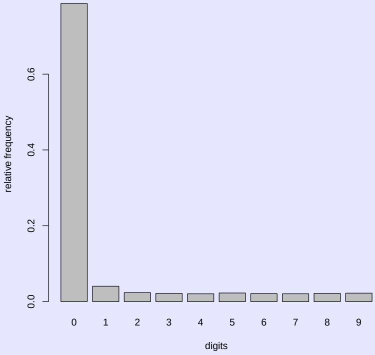
Bitwise Programming

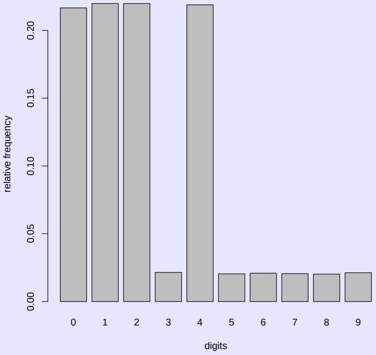
- In Bitwise programming a random number between 0 and $2^n - 1$ is chosen.
- Then it is squared.
- Next, appropriate number of left and right bitwise shifts give the middle digits.
- Finally the output is given as a decimal number.

CODES

(in HTML)

Graphical analysis





Analysis of Graphs

- The graph on the left side shows that the sequence of digits eventually converges to zero.
- Graph on the right side shows that the sequence falls in a loop.
- The loop consists of four digits $0, 1, 2, 4$
- We have observed that the loop is actually $24, 10, 24, 10, \ldots$(Initially we have taken the number of digits in the binary number is 22)

Analysis of Graphs

- The graph on the left side shows that the sequence of digits eventually converges to zero.
- Graph on the right side shows that the sequence falls in a loop.
- The loop consists of four digits $0, 1, 2, 4$
- We have observed that the loop is actually $24, 10, 24, 10, ...$(Initially we have taken the number of digits in the binary number is 22)

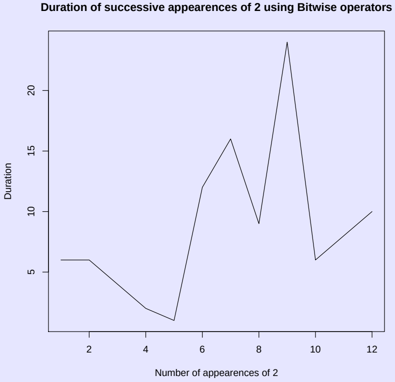
Analysis of Graphs

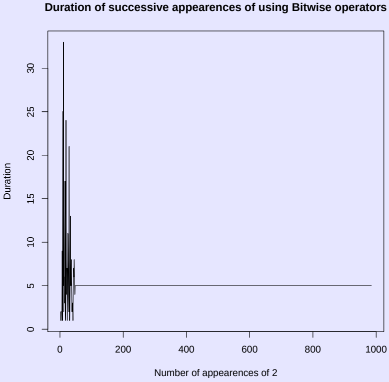
- The graph on the left side shows that the sequence of digits eventually converges to zero.
- Graph on the right side shows that the sequence falls in a loop.
- The loop consists of four digits $0, 1, 2, 4$
- We have observed that the loop is actually $24, 10, 24, 10, ...$(Initially we have taken the number of digits in the binary number is 22)

Analysis of Graphs

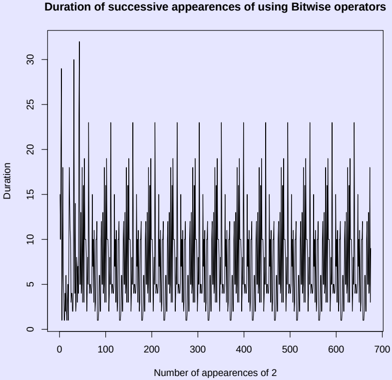
- The graph on the left side shows that the sequence of digits eventually converges to zero.
- Graph on the right side shows that the sequence falls in a loop.
- The loop consists of four digits $0, 1, 2, 4$
- We have observed that the loop is actually $24, 10, 24, 10, ...$(Initially we have taken the number of digits in the binary number is 22)

Graphical analysis





Graphical analysis



Conclusions based on the Graphs

- The proportion of occurences of digit 2 is approximately $\frac{15}{10000}$ which is much less than $\frac{1}{10}$.
- It is observed from the graph 2 that eventually the duration of successive appearence of digit 2 becomes constant while graph 3 shows that it falls in a loop.

Conclusions based on the Graphs

- The proportion of occurences of digit 2 is approximately $\frac{15}{10000}$ which is much less than $\frac{1}{10}$.
- It is observed from the graph 2 that eventually the duration of successive appearence of digit 2 becomes constant while graph 3 shows that it falls in a loop.

Comparison between Bitwise and Direct method

- Barplot of relative frequencies of digits in the sequence of random number generated is one way to compare these two ways of implementation of Von Neumann method.
- In case of direct method,the relative frequencies of all the digits are approximately same but in case of Bitwise method, in general, the digit 0 occur more often than other digits.
- Another way to do so is to plot the duration between the successive appearance of any digit.
- In case of Bitwise method the sequence of durations falls in a loop much before than that in case of Direct method.
- In Bitwise method the sequence of durations eventually becomes constant in several cases.

Comparison between Bitwise and Direct method

- Barplot of relative frequencies of digits in the sequence of random number generated is one way to compare these two ways of implementation of Von Neumann method.
- In case of direct method,the relative frequencies of all the digits are approximately same but in case of Bitwise method, in general, the digit 0 occur more often than other digits.
- Another way to do so is to plot the duration between the successive appearance of any digit.
- In case of Bitwise method the sequence of durations falls in a loop much before than that in case of Direct method.
- In Bitwise method the sequence of durations eventually becomes constant in several cases.

Comparison between Bitwise and Direct method

- Barplot of relative frequencies of digits in the sequence of random number generated is one way to compare these two ways of implementation of Von Neumann method.
- In case of direct method,the relative frequencies of all the digits are approximately same but in case of Bitwise method, in general, the digit 0 occur more often than other digits.
- Another way to do so is to plot the duration between the successive appearance of any digit.
- In case of Bitwise method the sequence of durations falls in a loop much before than that in case of Direct method.
- In Bitwise method the sequence of durations eventually becomes constant in several cases.

Comparison between Bitwise and Direct method

- Barplot of relative frequencies of digits in the sequence of random number generated is one way to compare these two ways of implementation of Von Neumann method.
- In case of direct method,the relative frequencies of all the digits are approximately same but in case of Bitwise method, in general, the digit 0 occur more often than other digits.
- Another way to do so is to plot the duration between the successive appearance of any digit.
- In case of Bitwise method the sequence of durations falls in a loop much before than that in case of Direct method.
- In Bitwise method the sequence of durations eventually becomes constant in several cases.

Comparison between Bitwise and Direct method

- Barplot of relative frequencies of digits in the sequence of random number generated is one way to compare these two ways of implementation of Von Neumann method.
- In case of direct method,the relative frequencies of all the digits are approximately same but in case of Bitwise method, in general, the digit 0 occur more often than other digits.
- Another way to do so is to plot the duration between the successive appearance of any digit.
- In case of Bitwise method the sequence of durations falls in a loop much before than that in case of Direct method.
- In Bitwise method the sequence of durations eventually becomes constant in several cases.

K-algorithm

Given a 10-digit decimal number X, this algorithm may be used to change X to the number that should come next in a supposedly random sequence. The algorithm is as follows:

K1 [Choose Number of Iterations] Set $Y \leftarrow \lfloor X/10^9 \rfloor$, the most significant digit of X (We will execute steps K2 through K13 exactly $Y + 1$ times)

K2 [Choose Random Step] Set $Z \leftarrow \lfloor X/10^8 \rfloor \mathbf{mod} 10$, the second most significant digit of X. Go to step $K(3 + Z)$

K3 [Ensure $\geq 5 \times 10^9$] If $X < 5000000000$, set $X \leftarrow X + 5000000000$

K4 [Middle Square] Replace X by $\lfloor X^2/10^5 \rfloor \mathbf{mod} 10^{10}$

K-algorithm

Given a 10-digit decimal number X, this algorithm may be used to change X to the number that should come next in a supposedly random sequence. The algorithm is as follows:

K1 [Choose Number of Iterations] Set $Y \leftarrow \lfloor X/10^9 \rfloor$, the most significant digit of X (We will execute steps K2 through K13 exactly $Y+1$ times)

K2 [ ] Set $Z \leftarrow \lfloor X/10^8 \rfloor \mathbf{mod} 10$, the second most significant digit of X. Go to step $K(3+Z)$

K3 [ ] If $X < 5000000000$, set $X \leftarrow X + 5000000000$

K4 [Middle Square] Replace X by $\lfloor X^2/10^5 \rfloor \mathbf{mod} 10^{10}$

K-algorithm

Given a 10-digit decimal number X, this algorithm may be used to change X to the number that should come next in a supposedly random sequence. The algorithm is as follows:

- K1 [Choose Number of Iterations] Set $Y \leftarrow \lfloor X/10^9 \rfloor$, the most significant digit of X (We will execute steps K2 through K13 exactly $Y + 1$ times)
- K2 [Choose Random Step] Set $Z \leftarrow \lfloor X/10^8 \rfloor \mathbf{mod} 10$, the second most significant digit of X. Go to step $K(3+Z)$
- K3 [ ] If $X < 5000000000$, set $X \leftarrow X + 5000000000$
- K4 [ ] Replace X by $\lfloor X^2/10^5 \rfloor \mathbf{mod} 10^{10}$

K-algorithm

Given a 10-digit decimal number X, this algorithm may be used to change X to the number that should come next in a supposedly random sequence. The algorithm is as follows:

- K1 [Choose Number of Iterations] Set $Y \leftarrow \lfloor X/10^9 \rfloor$, the most significant digit of X (We will execute steps K2 through K13 exactly $Y+1$ times)
- K2 [Choose Random Step] Set $Z \leftarrow \lfloor X/10^8 \rfloor \mathbf{mod} 10$, the second most significant digit of X. Go to step $K(3+Z)$
- K3 [Ensure $\geq 5 \times 10^9$] If $X < 5000000000$, set $X \leftarrow X + 5000000000$
- K4 [ ] Replace X by $\lfloor X^2/10^5 \rfloor \mathbf{mod} 10^{10}$

K-algorithm

Given a 10-digit decimal number X, this algorithm may be used to change X to the number that should come next in a supposedly random sequence. The algorithm is as follows:

- K1 [Choose Number of Iterations] Set $Y \leftarrow \lfloor X/10^9 \rfloor$, the most significant digit of X (We will execute steps K2 through K13 exactly $Y+1$ times)
- K2 [Choose Random Step] Set $Z \leftarrow \lfloor X/10^8 \rfloor \mathbf{mod} 10$, the second most significant digit of X. Go to step $K(3+Z)$
- K3 [Ensure $\geq 5 \times 10^9$] If $X < 5000000000$, set $X \leftarrow X + 5000000000$
- K4 [Middle Square] Replace X by $\lfloor X^2/10^5 \rfloor \mathbf{mod} 10^{10}$

K-Algorithm

- K5 [Multiply] Replace X by $(1001001001X)\mathbf{mod}10^{10}$
- K6 [ ] If $X < 100000000$, then set $X \leftarrow X + 9814055677$; otherwise set $X \leftarrow 10^{10} - X$
- K7 [ ] Interchange the higher 5 digits of X with the higher-order five digits; that is, set $X \leftarrow 10^5(X\mathbf{mod}10^5) + \lfloor X/10^5 \rfloor$
- K8 [Multiply]Same as K5
- K9 [Decrease Digits]Decrease each nonzero digits of the decimal representation of X by one.

K-Algorithm

- K5 [Multiply] Replace X by $(1001001001X) \mathbf{mod} 10^{10}$
- K6 [Pseudo-complement] If $X < 100000000$, then set $X \leftarrow X + 9814055677$; otherwise set $X \leftarrow 10^{10} - X$
- K7 [ ] Interchange the higher 5 digits of X with the higher-order five digits; that is, set $X \leftarrow 10^5(X \mathbf{mod} 10^5) + \lfloor X/10^5 \rfloor$
- K8 [ ]Same as K5
- K9 [Decrease Digits]Decrease each nonzero digits of the decimal representation of X by one.

K-Algorithm

K5 [Multiply] Replace X by $(1001001001X)\mathbf{mod}10^{10}$

K6 [Pseudo-complement] If $X < 100000000$, then set $X \leftarrow X + 9814055677$; otherwise set $X \leftarrow 10^{10} - X$

K7 [Interchange halves] Interchange the higher 5 digits of X with the higher-order five digits; that is, set $X \leftarrow 10^5(X\mathbf{mod}10^5) + \lfloor X/10^5 \rfloor$

K8 [ ]Same as K5

K9 [ ]Decrease each nonzero digits of the decimal representation of X by one.

K-Algorithm

- K5 [Multiply] Replace X by $(1001001001X)\mathbf{mod}10^{10}$
- K6 [Pseudo-complement] If $X < 100000000$, then set $X \leftarrow X + 9814055677$; otherwise set $X \leftarrow 10^{10} - X$
- K7 [Interchange halves] Interchange the higher 5 digits of X with the higher-order five digits; that is, set $X \leftarrow 10^5(X\mathbf{mod}10^5) + \lfloor X/10^5 \rfloor$
- K8 [Multiply]Same as K5
- K9 [ ]Decrease each nonzero digits of the decimal representation of X by one.

K-Algorithm

K5 [Multiply] Replace X by $(1001001001X)\mathbf{mod}10^{10}$

K6 [Pseudo-complement] If $X < 100000000$, then set $X \leftarrow X + 9814055677$; otherwise set $X \leftarrow 10^{10} - X$

K7 [Interchange halves] Interchange the higher 5 digits of X with the higher-order five digits; that is, set $X \leftarrow 10^5(X\mathbf{mod}10^5) + \lfloor X/10^5 \rfloor$

K8 [Multiply]Same as K5

K9 [Decrease Digits]Decrease each nonzero digits of the decimal representation of X by one.

K-Algorithm

K10 [99999 Modify] If $X < 10^5$, set $X \leftarrow X^2 + 99999$; otherwise set $X \leftarrow X - 99999$

K11 [ ] (At this point X can not be zero.) If $X < 10^9$, set $X \leftarrow 10X$ and repeat this step.

K12 [ ]Replace X by $\lfloor X(X-1)/10^5 \rfloor \mathbf{mod} 10^{10}$, that is middle 10 digits of $X(X-1)$

K13 [Repeat?] If $Y > 0$, decrease Y by one and return to $K2$. If $Y = 0$, the algorithm terminates with X as the desired random value.

K-Algorithm

K10 [99999 Modify] If $X < 10^5$, set $X \leftarrow X^2 + 99999$; otherwise set $X \leftarrow X - 99999$

K11 [Normalize] (At this point X can not be zero.) If $X < 10^9$, set $X \leftarrow 10X$ and repeat this step.

K12 [ ]Replace X by $\lfloor X(X-1)/10^5 \rfloor \mathbf{mod} 10^{10}$, that is middle 10 digits of $X(X-1)$

K13 [ ] If $Y > 0$, decrease Y by one and return to $K2$. If $Y = 0$, the algorithm terminates with X as the desired random value.

K-Algorithm

K10 [99999 Modify] If $X < 10^5$, set $X \leftarrow X^2 + 99999$; otherwise set $X \leftarrow X - 99999$

K11 [Normalize] (At this point X can not be zero.) If $X < 10^9$, set $X \leftarrow 10X$ and repeat this step.

K12 [Modified Middle Square]Replace X by $\lfloor X(X-1)/10^5 \rfloor \mathbf{mod} 10^{10}$, that is middle 10 digits of $X(X-1)$

K13 [ ] If $Y > 0$, decrease Y by one and return to $K2$. If $Y = 0$, the algorithm terminates with X as the desired random value.

K-Algorithm

K10 [99999 Modify] If $X < 10^5$, set $X \leftarrow X^2 + 99999$; otherwise set $X \leftarrow X - 99999$

K11 [Normalize] (At this point X can not be zero.) If $X < 10^9$, set $X \leftarrow 10X$ and repeat this step.

K12 [Modified Middle Square]Replace X by $\lfloor X(X-1)/10^5 \rfloor \mathbf{mod} 10^{10}$, that is middle 10 digits of $X(X-1)$

K13 [Repeat?] If $Y > 0$, decrease Y by one and return to $K2$. If $Y = 0$, the algorithm terminates with X as the desired random value.

Programming for K-Algorithm

- To solve this problem we used $\mathbf{C}$ language.
- This can also be solved in $\mathbf{R}$.
- In order to minimize the complexity involved in implementing the algorithm in $\mathbf{R}$ we prefered $\mathbf{C}$.
- We have made the generated random numbers readily available, for further testing, in $\mathbf{R}$ by building a shared object.

Programming for K-Algorithm

- To solve this problem we used $\mathbf{C}$ language.
- This can also be solved in $\mathbf{R}$.
- In order to minimize the complexity involved in implementing the algorithm in $\mathbf{R}$ we prefered $\mathbf{C}$.
- We have made the generated random numbers readily available, for further testing, in $\mathbf{R}$ by building a shared object.

Programming for K-Algorithm

- To solve this problem we used $\mathbf{C}$ language.
- This can also be solved in $\mathbf{R}$.
- In order to minimize the complexity involved in implementing the algorithm in $\mathbf{R}$ we prefered $\mathbf{C}$.
- We have made the generated random numbers readily available, for further testing, in $\mathbf{R}$ by building a shared object.

Programming for K-Algorithm

- To solve this problem we used $\mathbf{C}$ language.
- This can also be solved in $\mathbf{R}$.
- In order to minimize the complexity involved in implementing the algorithm in $\mathbf{R}$ we prefered $\mathbf{C}$.
- We have made the generated random numbers readily available, for further testing, in $\mathbf{R}$ by building a shared object.

Difficulties Encountered

- Memory Allocation Problem
  - In **C**, the maximum positive integer value that can be saved is 18446744073709551615.
  - Since we are dealing with a 10-digit number, in several steps of the program, we have to handle numbers exceeding this bound.
- While making shared library for **R**, the numbers are transferred as NUMERIC(not as INTEGER) as there may be some memory allocation problem.

Difficulties Encountered

- Memory Allocation Problem
  - In $\mathbf{C}$, the maximum positive integer value that can be saved is 18446744073709551615.
  - Since we are dealing with a 10-digit number, in several steps of the program, we have to handle numbers exceeding this bound.
- While making shared library for $\mathbb{R}$, the numbers are transferred as NUMERIC(not as INTEGER) as there may be some memory allocation problem.

Difficulties Encountered

- Memory Allocation Problem
  - In **C**, the maximum positive integer value that can be saved is 18446744073709551615.
  - Since we are dealing with a 10-digit number, in several steps of the program, we have to handle numbers exceeding this bound.
- While making shared library for **R**, the numbers are transferred as NUMERIC(not as INTEGER) as there may be some memory allocation problem.

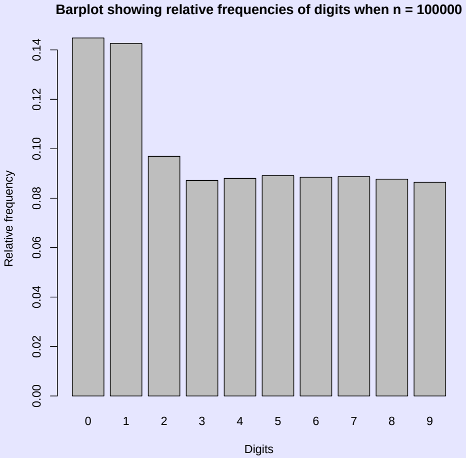
Difficulties Encountered

- Memory Allocation Problem
    - * In $\mathbf{C}$, the maximum positive integer value that can be saved is 18446744073709551615.
    - * Since we are dealing with a 10-digit number, in several steps of the program, we have to handle numbers exceeding this bound.
- While making shared library for $\mathbf{R}$, the numbers are transferred as NUMERIC(not as INTEGER) as there may be some memory allocation problem.

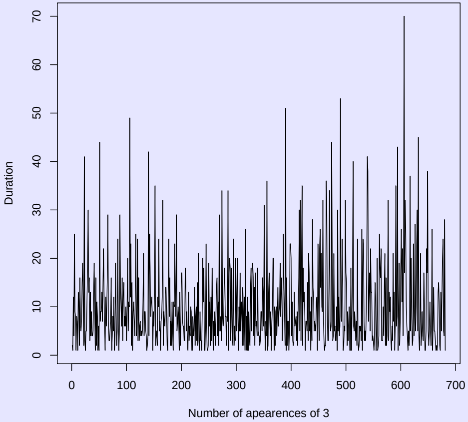
CODES

(in HTML)

Graphical analysis



Graphical analysis



Observations and Conclusion

- From the barplot it is clear that the probability of occurance of 0 and 1 is higher than the rest.
- This is obviously not expected.
- In the second graph the time lag between two successive occurance of 3 is observed.
- It comes out to be random enough.
- So we can conclude that though the numbers are appearing at random time point, the probability of their appearance is not equal.

Observations and Conclusion

- From the barplot it is clear that the probability of occurance of 0 and 1 is higher than the rest.
- This is obviously not expected.
- In the second graph the time lag between two successive occurance of 3 is observed.
- It comes out to be random enough.
- So we can conclude that though the numbers are appearing at random time point, the probability of their appearance is not equal.

Observations and Conclusion

- From the barplot it is clear that the probability of occurance of 0 and 1 is higher than the rest.
- This is obviously not expected.
- In the second graph the time lag between two successive occurance of 3 is observed.
- It comes out to be random enough.
- So we can conclude that though the numbers are appearing at random time point, the probability of their appearance is not equal.

Observations and Conclusion

- From the barplot it is clear that the probability of occurance of 0 and 1 is higher than the rest.
- This is obviously not expected.
- In the second graph the time lag between two successive occurance of 3 is observed.
- It comes out to be random enough.
- So we can conclude that though the numbers are appearing at random time point, the probability of their appearance is not equal.

Observations and Conclusion

- From the barplot it is clear that the probability of occurance of 0 and 1 is higher than the rest.
- This is obviously not expected.
- In the second graph the time lag between two successive occurance of 3 is observed.
- It comes out to be random enough.
- So we can conclude that though the numbers are appearing at random time point, the probability of their appearance is not equal.

Linear Congruential Generators

- Introduced by D.H.Lehmer around 1949.
- The generator is defined by the

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

Where,

1. $\{X_n\}$ = Sequence of pseudorandom values
2. m =The modulus $\quad m > 0$
3. a =The multiplier $\quad 0 < a < m$
4. c =The increment $\quad 0 \leq c < m$
5. X_0 =The seed or start value. $\quad 0 \leq X_0 < m$

- This method gives random numbers over $[0, m]$

Linear Congruential Generators

- Introduced by D.H.Lehmer around 1949.
- The generator is defined by the

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

Where,

1. $\{X_n\}$ = Sequence of pseudorandom values
2. m =The modulus $m > 0$
3. a =The multiplier $0 < a < m$
4. c =The increment $0 \leq c < m$
5. X_0 =The seed or start value. $0 \leq X_0 < m$

- This method gives random numbers over $[0, m]$

Linear Congruential Generators

- Introduced by D.H.Lehmer around 1949.
- The generator is defined by the

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

Where,

1. $\{X_n\}$ = Sequence of pseudorandom values
2. m =The modulus $\quad m > 0$
3. a =The multiplier $\quad 0 < a < m$
4. c =The increment $\quad 0 \leq c < m$
5. X_0 =The seed or start value. $\quad 0 \leq X_0 < m$

- This method gives random numbers over $[0, m]$

Linear Congruential Generators

- Introduced by D.H.Lehmer around 1949.
- The generator is defined by the

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

Where,

1. $\{X_n\} =$ Sequence of pseudorandom values
2. $m =$The modulus $\quad m > 0$
3. $a =$The multiplier $\quad 0 < a < m$
4. $c =$The increment $\quad 0 \leq c < m$
5. $X_0 =$The seed or start value. $\quad 0 \leq X_0 < m$

- This method gives random numbers over $[0, m]$

Linear Congruential Generators

- Introduced by D.H.Lehmer around 1949.
- The generator is defined by the

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

Where,

1. $\{X_n\} =$ Sequence of pseudorandom values
2. $m =$The modulus $\quad m > 0$
3. $a =$The multiplier $\quad 0 < a < m$
4. $c =$The increment $\quad 0 \leq c < m$
5. $X_0 =$The seed or start value. $\quad 0 \leq X_0 < m$

- This method gives random numbers over $[0, m]$

Linear Congruential Generators

- Introduced by D.H.Lehmer around 1949.
- The generator is defined by the

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

Where,

1. $\{X_n\} =$ Sequence of pseudorandom values
2. $m =$The modulus $\quad m > 0$
3. $a =$The multiplier $\quad 0 < a < m$
4. $c =$The increment $\quad 0 \leq c < m$
5. $X_0 =$The seed or start value. $\quad 0 \leq X_0 < m$

- This method gives random numbers over $[0, m]$

Linear Congruential Generators

- Introduced by D.H.Lehmer around 1949.
- The generator is defined by the

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

Where,

1. $\{X_n\} =$ Sequence of pseudorandom values
2. $m =$The modulus $\quad m > 0$
3. $a =$The multiplier $\quad 0 < a < m$
4. $c =$The increment $\quad 0 \leq c < m$
5. $X_0 =$The seed or start value. $\quad 0 \leq X_0 < m$

- This method gives random numbers over $[0, m]$

Linear Congruential Generators

- Introduced by D.H.Lehmer around 1949.
- The generator is defined by the

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

Where,

1. $\{X_n\} =$ Sequence of pseudorandom values
2. $m =$The modulus $\quad m > 0$
3. $a =$The multiplier $\quad 0 < a < m$
4. $c =$The increment $\quad 0 \leq c < m$
5. $X_0 =$The seed or start value. $\quad 0 \leq X_0 < m$

- This method gives random numbers over $[0, m]$

Linear Congruential Generators

- The sequence is not random for all choices of the parameters.
- We have to choose them carefully to get a long chain of random numbers.
- Eventually the process will fall in a loop.
- But we have to be careful that the length of the loop is large enough.
- Here is an example where bad choices of the parameters give a sequence which is perfectly non-random.

Linear Congruential Generators

- The sequence is not random for all choices of the parameters.
- We have to choose them carefully to get a long chain of random numbers.
- Eventually the process will fall in a loop.
- But we have to be careful that the length of the loop is large enough.
- Here is an example where bad choices of the parameters give a sequence which is perfectly non-random.

Linear Congruential Generators

- The sequence is not random for all choices of the parameters.
- We have to choose them carefully to get a long chain of random numbers.
- Eventually the process will fall in a loop.
- But we have to be careful that the length of the loop is large enough.
- Here is an example where bad choices of the parameters give a sequence which is perfectly non-random.

Linear Congruential Generators

- The sequence is not random for all choices of the parameters.
- We have to choose them carefully to get a long chain of random numbers.
- Eventually the process will fall in a loop.
- But we have to be careful that the length of the loop is large enough.
- Here is an example where bad choices of the parameters give a sequence which is perfectly non-random.

Linear Congruential Generators

- The sequence is not random for all choices of the parameters.
- We have to choose them carefully to get a long chain of random numbers.
- Eventually the process will fall in a loop.
- But we have to be careful that the length of the loop is large enough.
- Here is an example where bad choices of the parameters give a sequence which is perfectly non-random.

Linear Congruential Generators

- Take $m = 10$ and $X_0 = a = c = 7$. Then the sequence obtained is:

$$7, 6, 9, 0, 7, 6, 9, 0, \ldots$$

- This shows that the sequence is not random for all choices of the parameters.
- Now we will concentrate on choosing the combination of parameters which will give a good result.

Linear Congruential Generators

- Take $m = 10$ and $X_0 = a = c = 7$. Then the sequence obtained is:

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

- This shows that the sequence is not random for all choices of the parameters.
- Now we will concentrate on choosing the combination of parameters which will give a good result.

Linear Congruential Generators

- Take $m = 10$ and $X_0 = a = c = 7$. Then the sequence obtained is:
$$7, 6, 9, 0, 7, 6, 9, 0, ...$$
- This shows that the sequence is not random for all choices of the parameters.
- Now we will concentrate on choosing the combination of parameters which will give a good result.

Linear Congruential Generators

Choice of Multiplier ‘a’

- When $a = 0$ $X_n = c \textbf{ mod } m$ which is obviously not random.
- When $a = 1$, $X_n = (X_0 + nc)\textbf{mod } m$ amd the sequence will not behave like a random sequence.
- So in general we take $a \geq 2$

Choice of Modulus ‘m’

- First we should not take $m = 2$, then the sequence would at best have a form $0, 1, 0, 1, 0, 1, \ldots$
- If w is the word size of the computer a choice of m may be $w \pm 1$.
- Another choice may be the largest prime number $< w$.

Linear Congruential Generators

Choice of Multiplier 'a'

- When $a = 0$ $X_n = c \textbf{ mod } m$ which is obviously not random.
- When $a = 1$, $X_n = (X_0 + nc)\textbf{mod } m$ amd the sequence will not behave like a random sequence.
- So in general we take $a \geq 2$

Choice of Modulus 'm'

- First we should not take $m = 2$, then the sequence would at best have a form $0, 1, 0, 1, 0, 1, \ldots$
- If w is the word size of the computer a choice of m may be $w \pm 1$.
- Another choice may be the largest prime number $< w$.

Linear Congruential Generators

Choice of Multiplier ‘a’

- When $a = 0$ $X_n = c \textbf{ mod } m$ which is obviously not random.
- When $a = 1$, $X_n = (X_0 + nc)\textbf{mod } m$ amd the sequence will not behave like a random sequence.
- So in general we take $a \geq 2$

Choice of Modulus ‘m’

- First we should not take $m = 2$, then the sequence would at best have a form $0, 1, 0, 1, 0, 1, \ldots$
- If w is the word size of the computer a choice of m may be $w \pm 1$.
- Another choice may be the largest prime number $< w$.

Linear Congruential Generators

Choice of Multiplier 'a'

- When $a = 0$ $X_n = c \bmod m$ which is obviously not random.
- When $a = 1$, $X_n = (X_0 + nc)\mathbf{mod}\ m$ amd the sequence will not behave like a random sequence.
- So in general we take $a \geq 2$

Choice of Modulus 'm'

- First we should not take $m = 2$, then the sequence would at best have a form $0, 1, 0, 1, 0, 1, \ldots$
- If w is the word size of the computer a choice of m may be $w \pm 1$.
- Another choice may be the largest prime number $< w$.

Linear Congruential Generators

Choice of Multiplier 'a'

- When $a = 0$ $X_n = c \textbf{ mod } m$ which is obviously not random.
- When $a = 1$, $X_n = (X_0 + nc)\textbf{mod } m$ amd the sequence will not behave like a random sequence.
- So in general we take $a \geq 2$

Choice of Modulus 'm'

- First we should not take $m = 2$, then the sequence would at best have a form $0, 1, 0, 1, 0, 1, \ldots$
- If w is the word size of the computer a choice of m may be $w \pm 1$.
- Another choice may be the largest prime number $< w$.

Linear Congruential Generators

Choice of Multiplier ‘a’

- When $a = 0$ $X_n = c \textbf{ mod } m$ which is obviously not random.
- When $a = 1$, $X_n = (X_0 + nc)\textbf{mod } m$ amd the sequence will not behave like a random sequence.
- So in general we take $a \geq 2$

Choice of Modulus ‘m’

- First we should not take $m = 2$, then the sequence would at best have a form $0, 1, 0, 1, 0, 1, ...$
- If w is the word size of the computer a choice of m may be $w \pm 1$.
- Another choice may be the largest prime number $< w$.

Linear Congruential Generators(periodicity)

- Since only m different values are possible, the period surely can not be larger than m.
- It can be proved that a linear congruential method has period length m iff
  - c is relatively prime to m,
  - $b = a - 1$ is a multiple of p, for every prime p dividing m.
  - b is a multiple of 4, if m is a multiple of 4, $b = a - 1$.

Linear Congruential Generators(periodicity)

- Since only m different values are possible, the period surely can not be larger than m.
- It can be proved that a linear congruential method has period length m iff
  1. c is relatively prime to m,
  2. $b = a - 1$ is a multiple of p, for every prime p dividing m.
  3. b is a multiple of 4, if m is a multiple of 4, $b = a - 1$.

Linear Congruential Generators(periodicity)

- Since only m different values are possible, the period surely can not be larger than m.
- It can be proved that a linear congruential method has period length m iff
  1. c is relatively prime to m,
  2. $b = a - 1$ is a multiple of p, for every prime p dividing m.
  3. b is a multiple of 4, if m is a multiple of 4, $b = a - 1$.

Linear Congruential Generators(periodicity)

- Since only m different values are possible, the period surely can not be larger than m.
- It can be proved that a linear congruential method has period length m iff
  1. c is relatively prime to m,
  2. $b = a - 1$ is a multiple of p, for every prime p dividing m.
  3. b is a multiple of 4, if m is a multiple of 4, $b = a - 1$.

Linear Congruential Generators(periodicity)

- Since only m different values are possible, the period surely can not be larger than m.
- It can be proved that a linear congruential method has period length m iff
  1. c is relatively prime to m,
  2. $b = a - 1$ is a multiple of p, for every prime p dividing m.
  3. b is a multiple of 4, if m is a multiple of 4, $b = a - 1$.

Merits and Demerits

- Fast and Requires minimal memory.
- Proper choice of parameters gives 'good' sequence of PRNs.
- It is not, in general, cryptographically secure.
- If the parameters are not properly chosen it may have very short period.

Merits and Demerits

- Fast and Requires minimal memory.
- Proper choice of parameters gives ‘good’ sequence of PRNs.
- It is not, in general, cryptographically secure.
- If the parameters are not properly chosen it may have very short period.

Merits and Demerits

- Fast and Requires minimal memory.
- Proper choice of parameters gives ‘good’ sequence of PRNs.
- It is not, in general, cryptographically secure.
- If the parameters are not properly chosen it may have very short period.

Merits and Demerits

- Fast and Requires minimal memory.
- Proper choice of parameters gives ‘good’ sequence of PRNs.
- It is not, in general, cryptographically secure.
- If the parameters are not properly chosen it may have very short period.

Acknowledgement

- The Art of Computer Programming, Vol-2, Knuth
- Dr. Deepayan Sarkar
- Arijit Dutta, (M.Stat 2nd year)