Random Function Test Report

凉凉

May 13, 2024

Abstract

In this report, a bunch of test are done on different pseudo random generator.

Contents

1 Introduce

In this report, there will be a series tests for pseudo random number generator functions:

- `(random 1.0)` for Common Lisp SBCL 2.4.3 built-in function `cl:random` (MT19937) [1]

  This will be used as the default random generator. [2] The following test will first be applied on default method first.

- Linear Congruential Generator Algorithms

  $$s_{n+1} = a \times s_n + b \bmod m$$

  The following (a, b, m) will be tested:

Table 1: Parameters a, b, m for Linear Congruential Generator with init seed. The name with `class` prefixed was get from class slides, and the name with `lcd` prefix was get from Wikipedia.

Name	a	b	m	Notes
`class-dependent`	1229	1	2048	Fail with independent test at section 3.1
`class-independent`	1597	51749	244944	
`lcg-zx81`	75	74	65537	m = $2^{16} + 1$
`lcg-ranqd1`	1664525	1013904223	4294967296	m = 2^{32}

- Middle-Square Method [3]

  $$\begin{aligned} s_{n+1} &= \left\lfloor \frac{s_n^2}{10^{\mathrm{numShift}(s_n^2)}} \right\rfloor \bmod 10^{\mathrm{size}} \\ \mathrm{numShift}(x) &= \left\lceil \frac{\mathrm{numSize}(x) - \mathrm{size}}{2} \right\rceil \\ \mathrm{numSize}(x) &= \lceil \lg x \rceil \end{aligned}$$

  This is just for fun, so only test for `seed = 675248`.

- True Random Numbers [4]

- Homebrew Hardware Random Generator

  Using RP2040 to read floating ADC0 Pin for noise input p_{adc0}, using the last bit $p_{\mathrm{adc0}} \bmod 2$ to generate random bin bits. Then using the random bits to make $u = \sum_i^n (\frac{1}{2})^{b_i \times i}$ for $u \in [0, 1)$.

  However, the result maynot be so good as shown in Figure 8, 16.

  I think the reason is due to the bit → float algorithm, which should be replaced with better ones. Also, a buffer for the random number could be added to improve the reading speed.

[1] The SBCL `random` function is based on MT19937 Algorithm. The source code for implementation could be found on Github.

[2] see the code at 11

[3] Middle-Square method was invented by John von Neumann, which could be considered as the first pseudo random number generator. Although this method may not be so good for random number generator, it has historically meanings and could act as a counterexample.

[4] The true number was request from random.org, which provides a true random numbers from cosmic ray.

2 Uniform

2.1 Uniform Test

Generate n numbers distributed between $[0, 1)$, and plot the histogram of them, calculate the standard deviation σ_{RNG}.

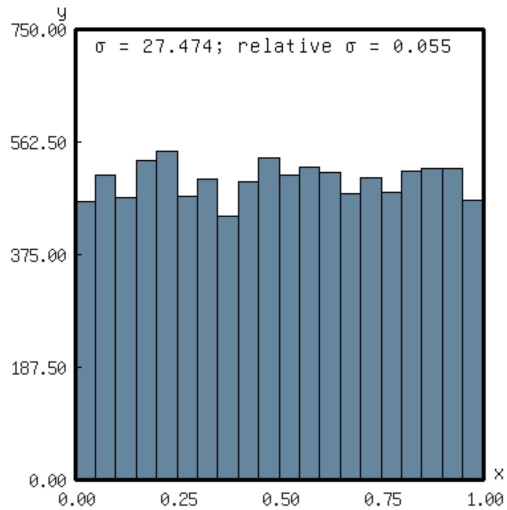

Figure 1: 10000 samples for `(random 1.0)`

2.2 Uniform Test on Other RNG

For other Random Number Generator:

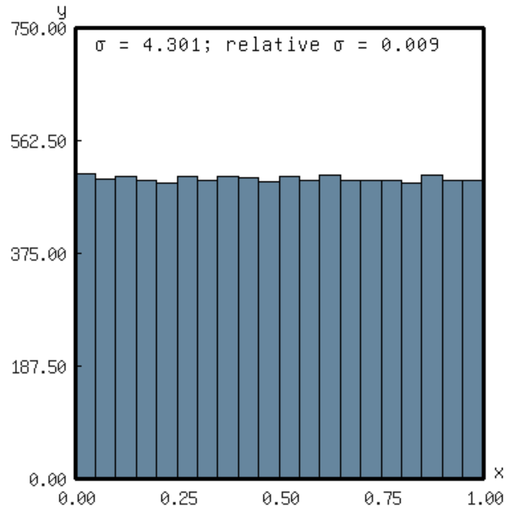

Figure 2: 10000 samples for `class-dependent`

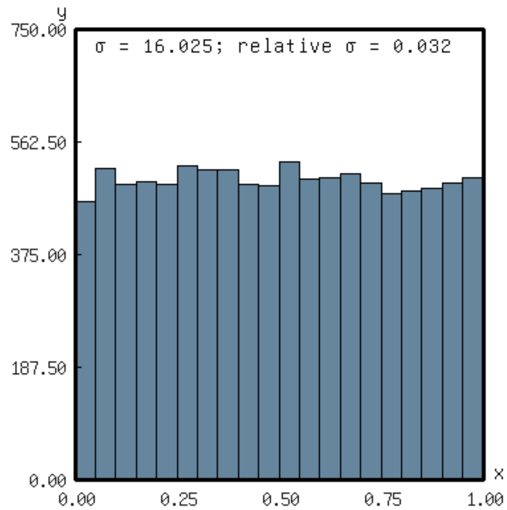

Figure 3: 10000 samples for `class-independent`

For LCG Methods:

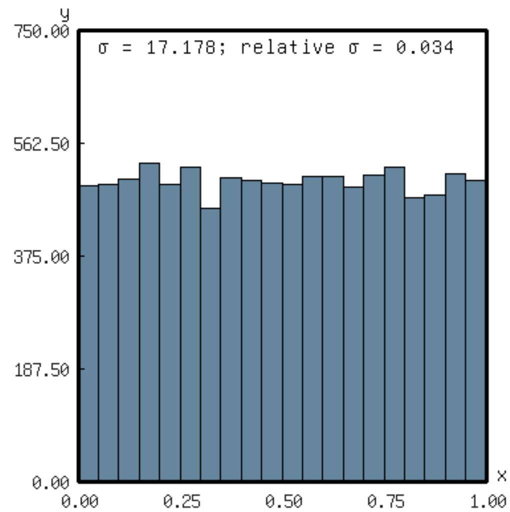

Figure 4: 10000 samples for `lcg-zx81`

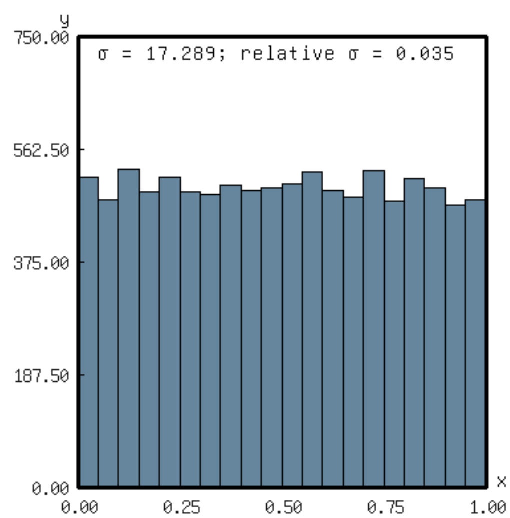

Figure 5: 10000 samples for `lcg-ranqd1`

For Middle Square Method

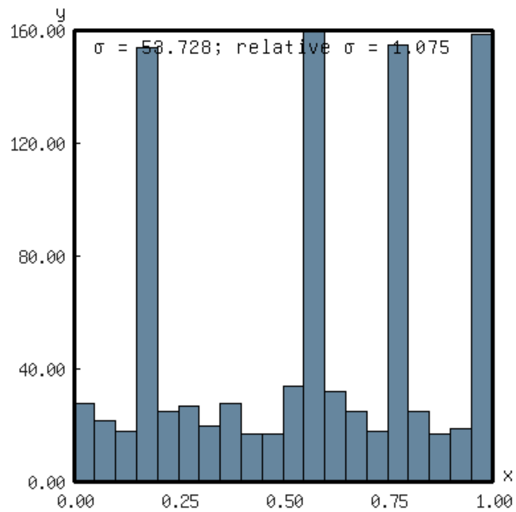

Figure 6: 1000 samples for `middle-square-675248`

It could be seen that the Middle-Square Method is not well evenly distributed between $[0, 1)$.

Also, since Middle-Square Method is likely to stuck into loop, which would produce same results, so the test sample `n` could not be too large.

For True Random from random.org:

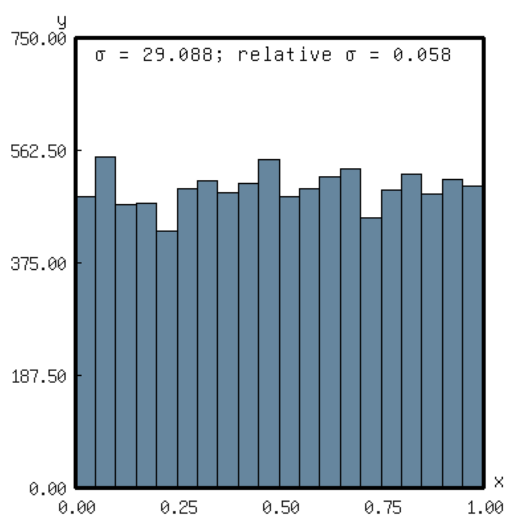

Figure 7: 10000 samples for `random-org`

For Homebrew Hardware Random Number Generator:

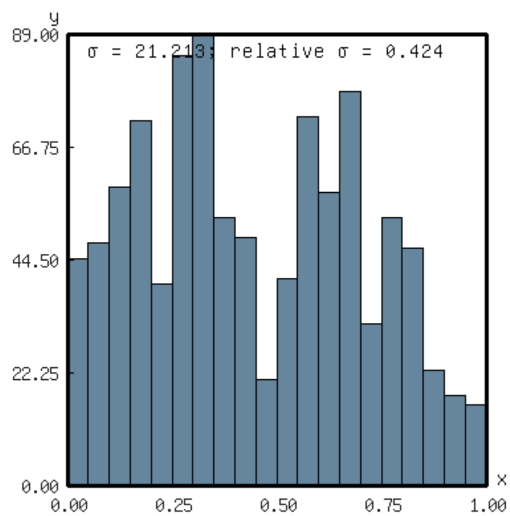

Figure 8: 10000 samples for `hrng`

3 Independent

3.1 Independent Test

Generate random numbers distributed between $[0, 1)$, and make them into point pair $(x, y) \in [0, 1)$, and see the plane.

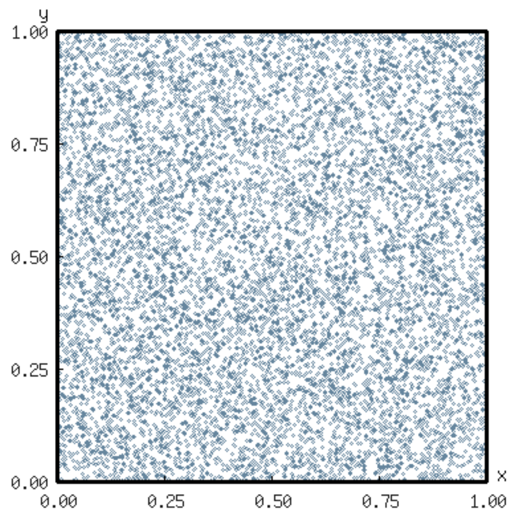

Figure 9: Independent Test for `(random 1.0)`

3.2 Independent Test for other RNG

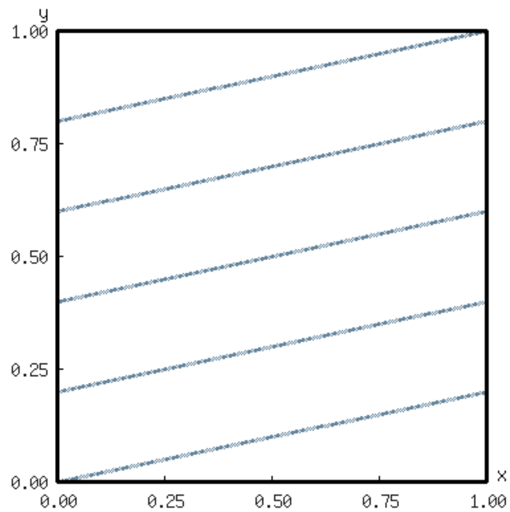

Figure 10: Independent Test for `class-dependent`

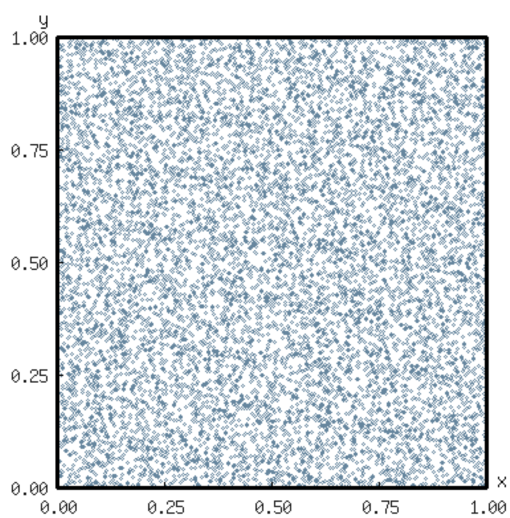

Figure 11: Independent Test for `class-independent`

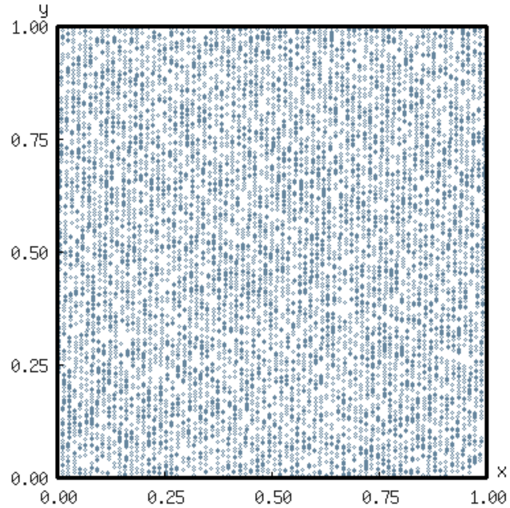

Figure 12: Independent Test for `lcg-zx81`

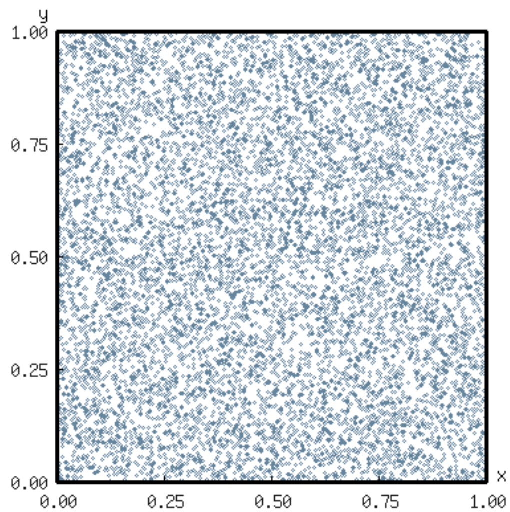

Figure 13: Independent Test for `lcg-ranqd1`

For The Middle Square method:

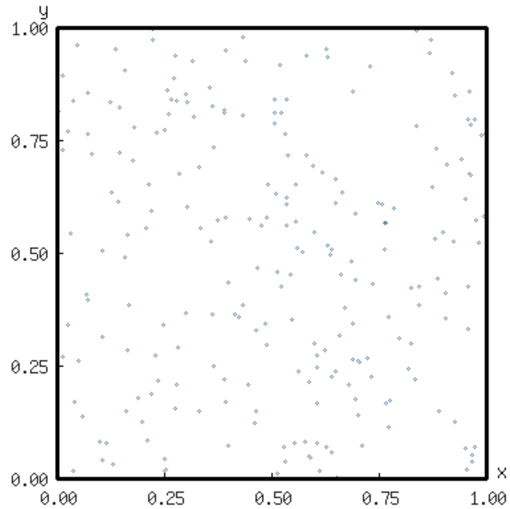

Figure 14: Independent Test for `middle-square-675248`

Middle Square method will not evenly fill the sampling space.
For True Random Numbers: [5]

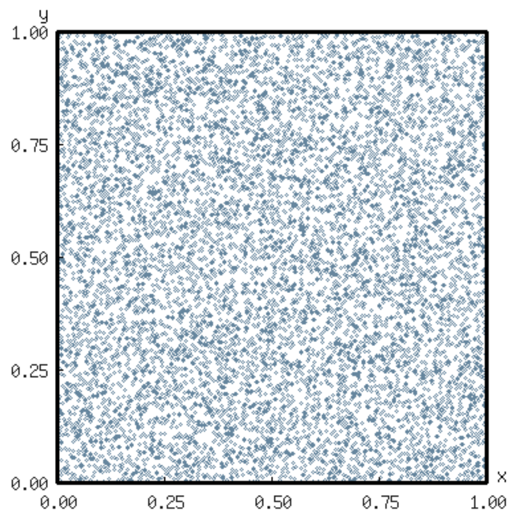

Figure 15: Independent Test for `random-org-iter`

For homebrew hardware random number generator:

[5] There's request limits for free access, so you may consider pay for api usage or...

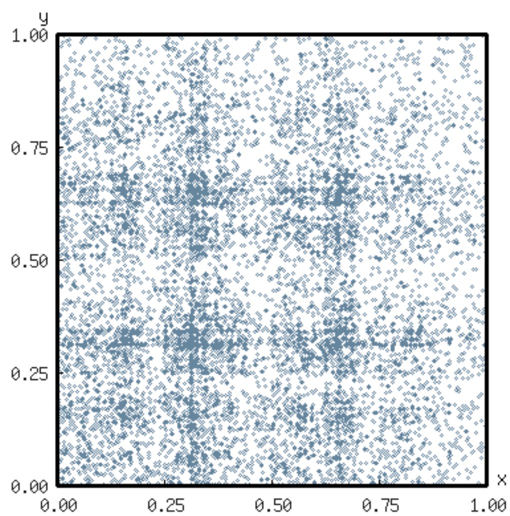

Figure 16: Independent Test for `hrng`

4 Periodicity

4.1 Periodicity Test

The periodicity test process for $p^{\text{PER}}(N, p)$ works like below:

1. generate a $n \times n$ matrix with random elements;
2. threshold the matrix with probability p, mark as $\boldsymbol{M}$
3. start from left $(0, i)$ and top $(i, 0)$ edge and perform depth first search, the depth first search routine is described below:[6]

   (a) start from $i_{\text{init}}, j_{\text{init}}$, with route $= \{(i_{\text{init}}, j_{\text{init}})\}$.

   (b) for current position i, j, find all next position next $= \{i_{\text{next}}, j_{\text{next}}\}$

   (c) choose $i_{\text{next}}, j_{\text{next}} \in$ next, if:

   - $\boldsymbol{M}_{i_{\text{next}}, j_{\text{next}}} = 1$ for next $i_{\text{next}}, j_{\text{next}}$ move able;
   - and $(i_{\text{next}}, j_{\text{next}}) \notin$ route for not moved before

   is

   - true
     then if:
     - passing top edge and $j_{\text{next}} = n$
     - passing bottom edge and $j_{\text{next}} = 0$
     - passing left edge and $i_{\text{next}} = n$
     - passing right edge and $i_{\text{next}} = 0$

     is
     - true means reach to the end, the route: $\{(i_{\text{next}}, j_{\text{next}})\}$ + route is the route crossing the $\boldsymbol{M}$
     - false repeat from depth first search step 2, with $i, j = i_{\text{next}}, j_{\text{next}}$, and route $= \{(i_{\text{next}} + j_{\text{next}})\}$ + route.
   - false
     change $i_{\text{next}}, j_{\text{next}} \in$ next until next is empty, if next is empty, then current i, j is unacceptable, set $\boldsymbol{M}_{i,j} = -1$ as a dead end.
     Note: it could be seen that when doing depth first search on $\boldsymbol{M}$ for possible cross route, it is possible for searching program came across a same point multiple times. so adding a dead end mark will accelerate the searching process.

   repeat for each $0 \leq i < n$, if for any i, there's a route crossing the matrix, the matrix is periodicity, otherwise, it is not periodicity.
4. repeat for `sampling` times from step 1, count all the periodicity times as `count`, and $p^{\text{PER}}(N, p)$ is $\frac{\text{count}}{\text{sampling}}$.

[6] There is a naive mistake in the previous algorithm, although it should be easy to pick out: the $\boldsymbol{M}$ should be restore after the horizontal and vertical search is ended. Because there's a case that you may not cross, for example, from top to bottom, but your searching process found that the left edge and right edge happened to cross from left to right, but if you do not restore the $\boldsymbol{M}$, this possible route will be mistakenly marked as dead end. A tricky way to solve this is make a copy of $\boldsymbol{M}$ and use orgin for horizontal, and the copy for vertical seperately (see raw code 11). This would make the searching time a little bit longer, though it would be more accerate.

Here's an example for the above algorithms (Figure 17):

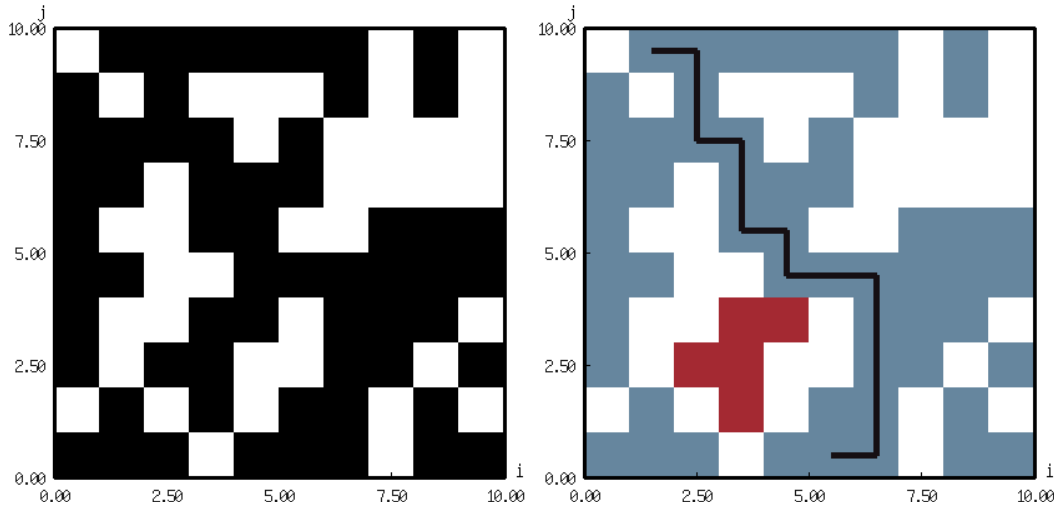

Figure 17: the **periodicity** 10×10 matrix with $p = 0.4$

Here the left side is the original matrix, with black grid for 1 and white grid for 0, thresholded by p; the right side is the searching matrix, with red grid for -1, aka. the dead end to accelerate the searching process.

This should be more clear with the following example (the none periodicity matrix, Figure 18):

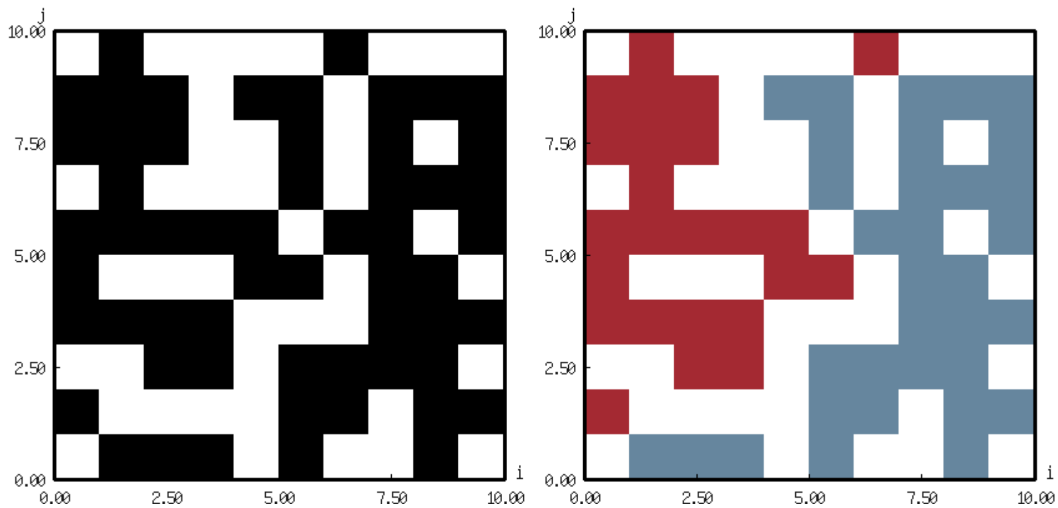

Figure 18: the **none periodicity** 10×10 matrix with $p = 0.4$

As you may see that in the non periodicity matrix, all the left and top edge is painted with red color for the dead end.

This method could also be applied to larger systems, for example (Figure 19):

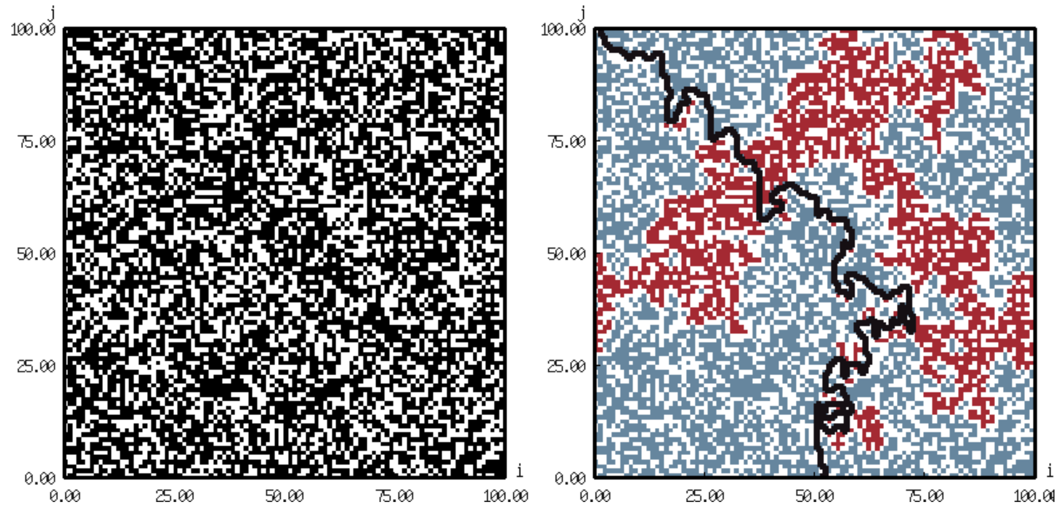

Figure 19: the **periodicity** 100×100 matrix with $p = 0.4$

And none periodicity 100×100 matrix as (Figure 20):

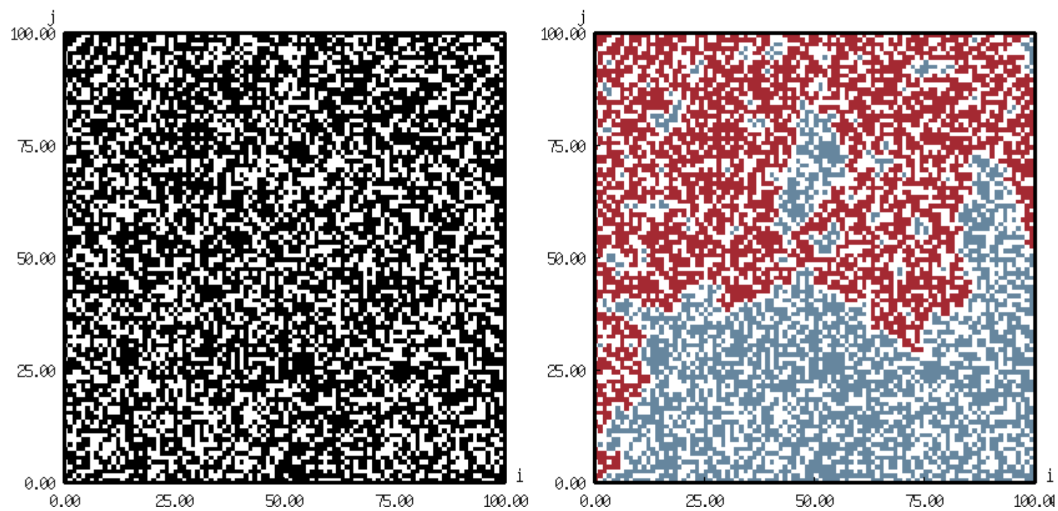

Figure 20: the **none periodicity** 100×100 matrix with $p = 0.4$

4.2 Relationship for $p^{\mathrm{PER}}(N, p)$ with N and p

Sampling for N from 10 to 400 with step 10, crosspondingly, p from 0 to 1 with step 0.1, each $p^{\mathrm{PER}}(N, p)$ for 1000 sampling times, which could be rendered into a 2d color plot as shown below:

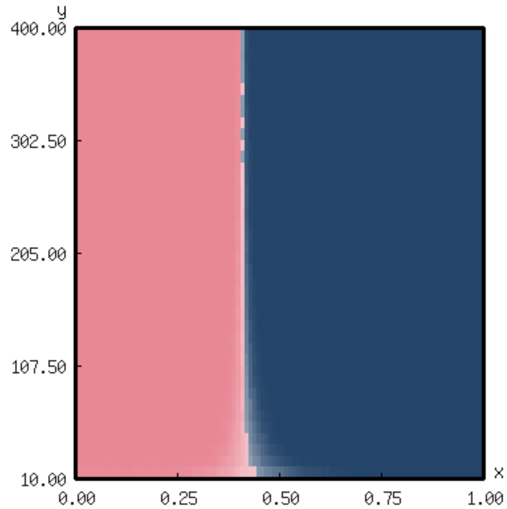

Figure 21: $p^{\mathrm{PER}}(N \in [10, 400], p \in [0, 1])$, with pink for 1, blue for 0

So the $p_{\mathrm{c}}(N)$ would be plotted as below:

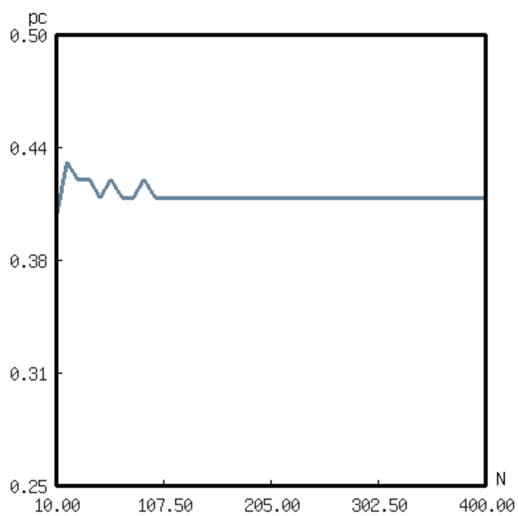

Figure 22: plot of $p_{\mathrm{c}}(N)$, with $p_{\mathrm{c}} \to 0.41$.

4.3 Periodicity on other random generator

For other RNG, sampling with N from 10 to 100 by 10, p from 0 to 1 by 0.1, each p^{PER} with 2000 sampling points.

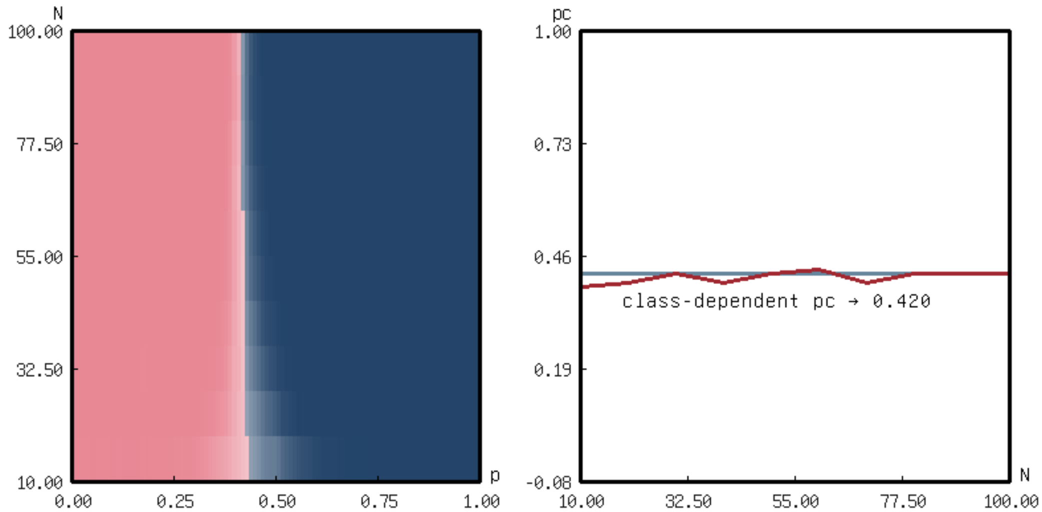

Figure 23: plot of $p_{\mathrm{c}}(N)$, with $p_{\mathrm{c}} \to 0.42$ for `class-dependent`

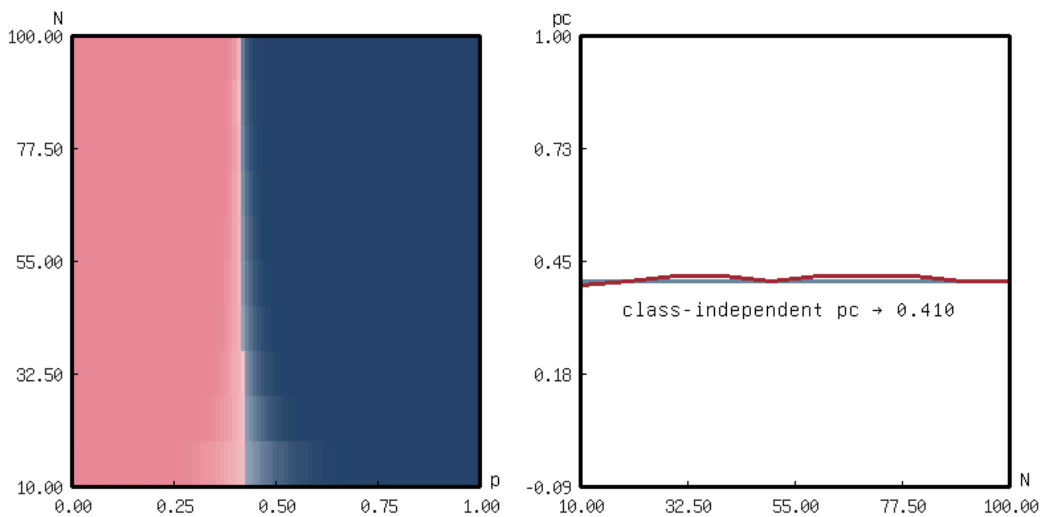

Figure 24: plot of $p_{\mathrm{c}}(N)$, with $p_{\mathrm{c}} \to 0.42$ for `class-independent`

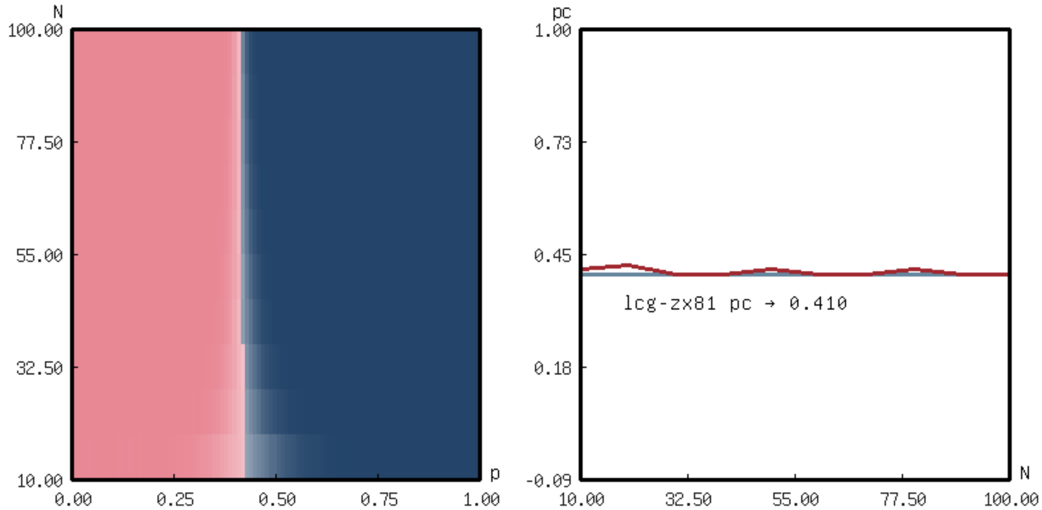

Figure 25: plot of $p_c(N)$, with $p_c \to 0.42$ for `lcg-zx81`

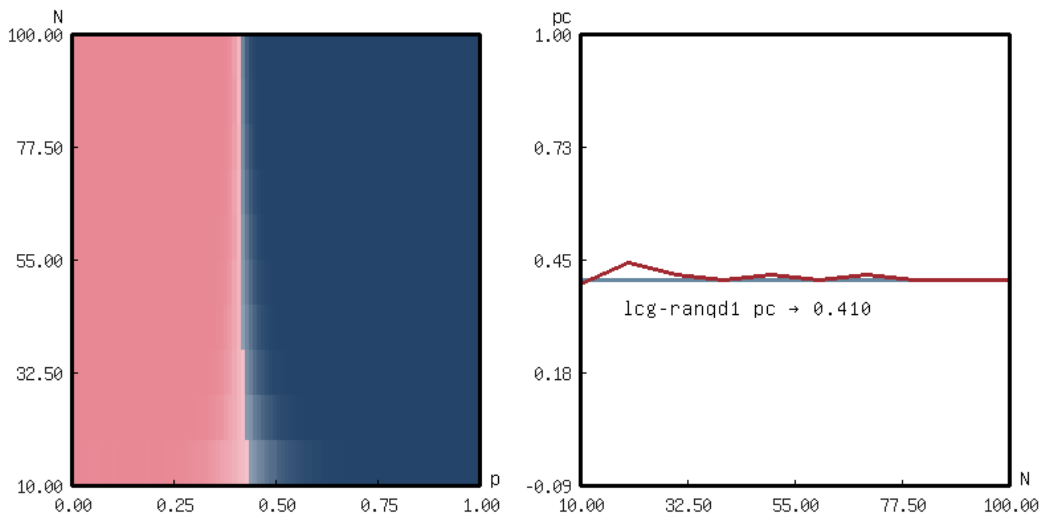

Figure 26: plot of $p_c(N)$, with $p_c \to 0.42$ for `lcg-ranqd1`

5 Cluster

5.1 Cluster Test

To calculate all the cluster in the $n \times n$ random matrix, a simple algorithm using DFS will be described as below: [fn: code implemented at B.4]

1. with $n \times n$ random matrix, start from $(i, j), i, j \in [0, n)$
2. using Depth First Searching method to find all the connected grid
   (a) start with cluster $= \{\}$, from $(i_{\text{init}}, j_{\text{init}})$, where $\boldsymbol{M}_{i_{\text{init}}, j_{\text{init}}} = 1$, set $\boldsymbol{M}_{i_{\text{init}}, j_{\text{init}}} =$ visited, [7] then let $(i, j) = (i_{\text{init}}, j_{\text{init}})$ and start iteration
   (b) for current position (i, j), find all its neighbour $(i_{\text{next}}, j_{\text{next}}) \in \text{nexts}(i, j)$, where $\boldsymbol{M}_{i_{\text{next}}, j_{\text{next}}} = 1$ (moveable), set $\boldsymbol{M}_{i_{\text{next}}, j_{\text{next}}} =$ visited
   (c) let $(i, j) = (i_{\text{next}}, j_{\text{next}})$, and repeat step b until no $(i_{\text{next}}, j_{\text{next}}) \in \text{nexts}(i, j)$ that $\boldsymbol{M}_{i_{\text{next}}, j_{\text{next}}} = 1$, add (i, j) to cluster, and reset (i, j) to previous position, iter from step b, until no previous (i, j) could be iterated
   (d) for each $(i, j) \in$ cluster, set $\boldsymbol{M}_{i,j} = \text{size(cluster)}$

[7] here the visited is `-1` in B.4

Here is some example for the cluster of the matrix:

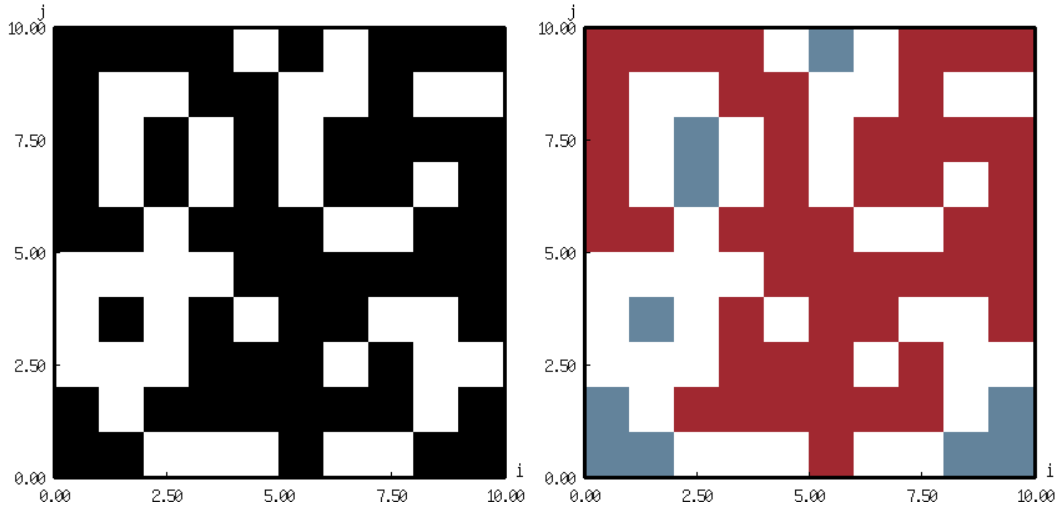

Figure 27: 10×10 matrix with threshold $p = 0.4$ cluster

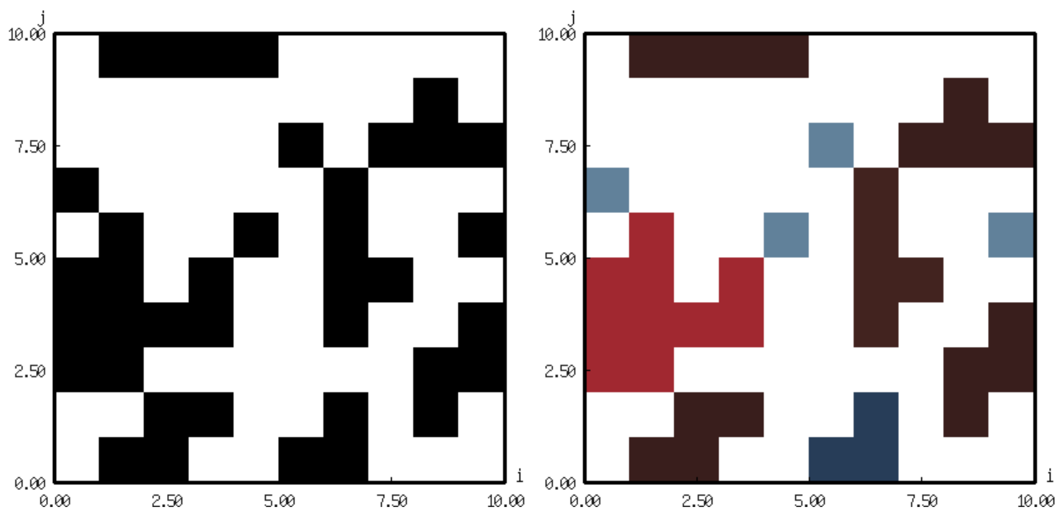

Figure 28: 10×10 matrix with threshold $p = 0.6$ cluster

For larger system:

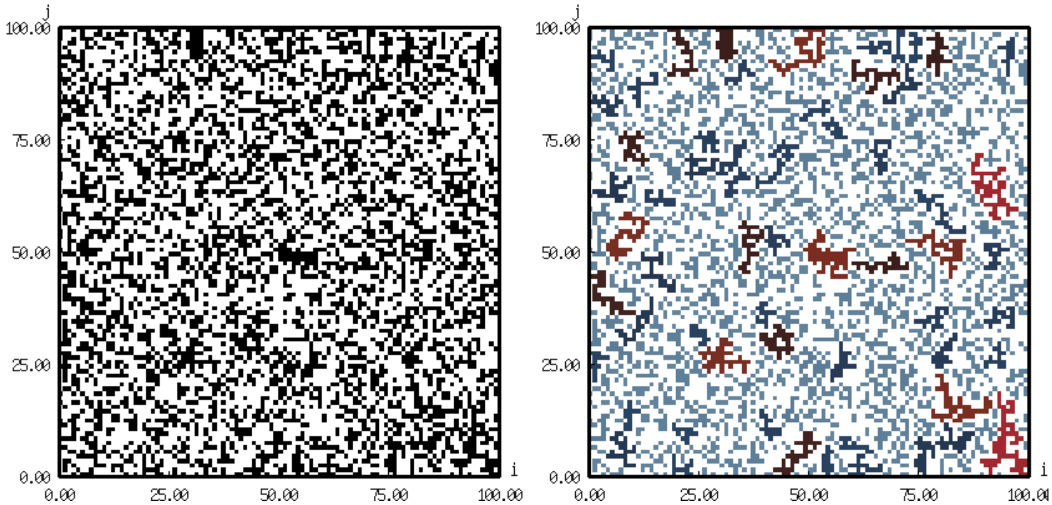

Figure 29: 100×100 matrix with threshold $p = 0.6$ cluster

Since we already know that $p_c \approx 0.41$ (Figure 22), with $p < p_c$ we'd expected to see less but lager clusters.

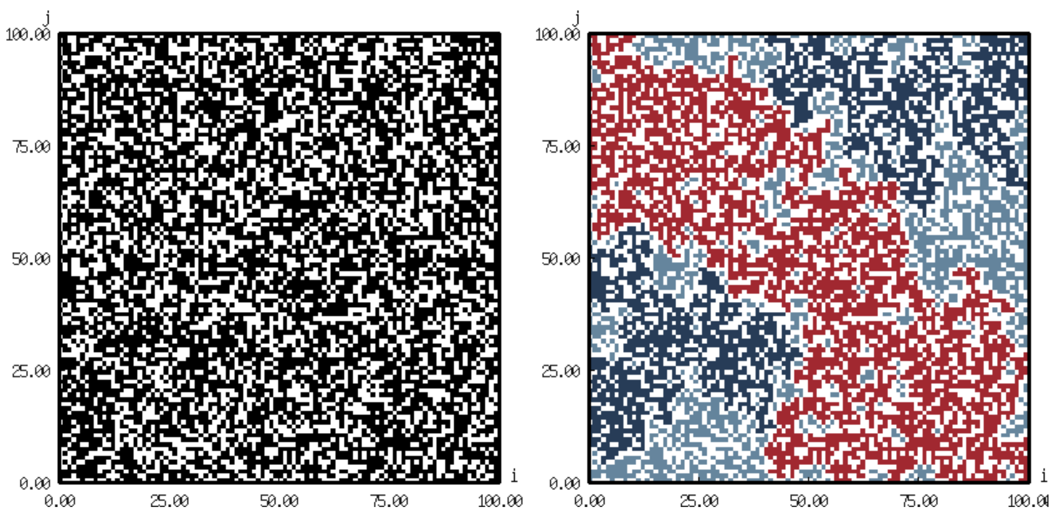

Figure 30: 100×100 matrix with threshold $p = 0.4$ cluster

As could be seen on the plot, with colored cluster, it is easy to spot out the percolation matrix.

5.2 Cluster Size Histogram

If counting each $n \times n$ matrix cluster size and plot them into histogram, will get the following results:

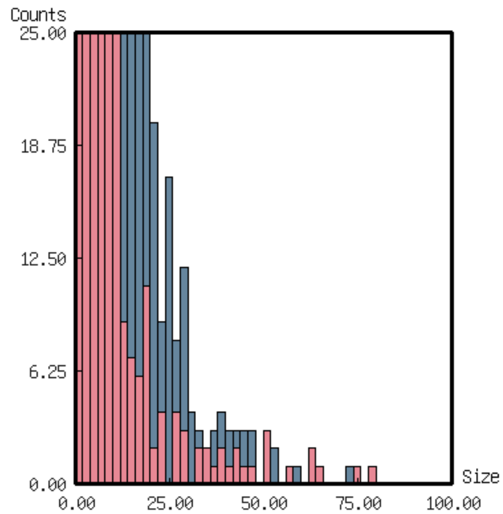

Figure 31: 200×200 matrix with $p = 0.6$ (blue), $p = 0.4$ (pink) cluster size histogram

Appendix

A Random Generators

A.1 Congruential Generators

```
(defun make-linear-congruential-generator (seed a b m)
  "Make a linear congrunential generator with `seed'. "
  (let ((seed seed))
    (lambda ()
      (setf seed (mod (+ (* a seed) b) m))
      (float (/ seed m)))))
```

Listing 1: General process to define a LCG random number generator

The Linear Congruential Generator Algorithms defined at Table 1 will be defined as:

```
(loop for (name a b m) in table
      for lcg-name = (intern (string-upcase
                              (subseq name 1 (1- (length name)))))
      do (eval `(defparameter ,lcg-name
                  (make-linear-congruential-generator ,seed ,a ,b ,m)))
      collect lcg-name)
```

Listing 2: Defines the LCG random number generator. The `seed` will be the unix time integer.

A.2 Middle-Square Method

```
(defun make-middle-square-random (seed
                                  &optional (repeat 1)
                                    (size (ceiling (log seed 10))))
  "Make a Uniform Middle Square random iterator. "
  (let ((seed seed)
        (max (expt 10 size)))
    (lambda ()
      (loop for i below repeat
            for square = (* seed seed)
            for len = (ceiling (log square 10))
            for shift = (ceiling (- len size) 2)
            do (setf seed (mod (floor square (expt 10 shift)) max))
            finally (return (/ seed max))))))
```

```
(defparameter middle-square-675248
  (make-middle-square-random 675248))
```

A.3 True Random Generator

The True random could be got via random.org:

```
(defparameter random-org-request-template
  "https://www.random.org/decimal-fractions/?num=~d&dec=~d&col=1&format=plain&rnd=new"
  "Template URL for `random-org' request. ")

(defun random-org (n &optional (precise 10))
  "Get `n' (< 10000) random numbers between [0, 1]."
  (declare ((integer 0 10000) n))
  (let ((request-url (format nil random-org-request-template n precise)))
    (with-input-from-string (stream (dex:get request-url))
```

```
      (loop for line = (read-line stream nil nil)
            while line
            collect (with-input-from-string (float line)
                      (read float))))))

(defun make-random-org (&optional (buffer-size 1000))
  "Make a `random-org' iterator. "
  (let ((buffer (random-org buffer-size)))
    (lambda ()
      (when (endp buffer)
        (setf buffer (random-org buffer-size)))
      (pop buffer))))
```

Since this depends heavily on the network to request for true random numbers, it's much slower to calculate the results.

```
(defparameter random-org-iter
  (make-random-org 5000))
```

A.4 Hardware Random Generator

The core part is like below: [8]

```
(defun random-float ()
  (let ((float 0.0))
    (dotimes (i random-bit-size)
      (setf float (+ (mod (analogread adc-pin) 2) (* 0.5 float))))
    (* 0.5 float)))
```

Listing 3: Read random float from ADC 0 on RP2040

Then it should be readed via Serial port to lisp. [9]

A.4.1 Hardware Random Number uLisp

```
;; ADC Pin Number
(defvar adc-pin 26)
(defvar random-bit-size 32)

(defun random-float ()
  (let ((float 0.0))
    (dotimes (i random-bit-size)
      (setf float (+ (mod (analogread adc-pin) 2) (* 0.5 float))))
    (* 0.5 float)))

(defun hardware-random-loop ()
  (loop
    (when (read)
      (digitalwrite :led-builtin t)
      (print (random-float))
      (digitalwrite :led-builtin nil))))

(hardware-random-loop)
```

Listing 4: The Hardware random number generation on RP2040

[8]the code was coded into RP2040 via uLisp project

[9]the code is omitted.

A.4.2 Serial Code C

```c
#include <sys/fcntl.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <termios.h>

int init_serial(char * dev_name, int baud_rate) {
  int fd = open(dev_name, O_RDWR | O_NONBLOCK);
  if (fd < 0) return 0; // error

  // init termio
  struct termios tio;
  memset(&tio, 0, sizeof(tio));
  tio.c_cflag = CS8 | CLOCAL | CREAD;
  tio.c_cc[VTIME] = 100;

  // Disable echo
  tio.c_lflag &= ~(ECHO);

  cfsetispeed(&tio, baud_rate);
  cfsetospeed(&tio, baud_rate);

  tcsetattr(fd, TCSANOW, &tio);

  return fd;
}

int serial_read(int fd, char* buffer, int buffer_size) {
  return read(fd, buffer, buffer_size);
}

int serial_write(int fd, char* data) {
  return write(fd, data, strlen(data));
}

int serial_close(int fd) {
  return close(fd);
}
```

Listing 5: A Minimum Serial Library to Read/Write Serial Port

Compile it as shared dynamic lib:

```
gcc -shared serial.c -o serial.dylib && realpath serial.dylib
```

which should be wrapped via CFFI in Common Lisp:

```lisp
(defpackage #:serial
  (:use :cl :cffi)
  (:export #:with-serial
           #:serial-write-string
           #:serial-read-string))
```

```lisp
(in-package :serial)
(load-foreign-library library-path)

(defcfun ("init_serial" %serial-init) :int
  (dev-name (:pointer :char))
  (baud-rate :int))

```

```
(defcfun ("serial_read" %serial-read) :int
  (serial :int)
  (buffer (:pointer :char))
  (buffer-size :int))

(defcfun ("serial_write" %serial-write) :int
  (serial :int)
  (data (:pointer :char)))

(defcfun ("serial_close" %serial-close) :int
  (serial :int))

(defmacro with-serial ((serial name &key (baud-rate 9600))
                       &body body)
  `(let* ((,serial (with-foreign-string (dev-name ,name)
                     (%serial-init dev-name ,baud-rate)))
          (res (progn ,@body)))
     (%serial-close ,serial)
     res))

(defun serial-write-string (serial string)
  (with-foreign-string (data string)
    (let ((status (%serial-write serial data)))
      (if (>= status 0) t nil))))

(defun serial-read-string (serial &optional (buffer-size 64) (empty nil))
  (with-foreign-string (buffer (make-string buffer-size))
    (let ((status (%serial-read serial buffer buffer-size)))
      (cond ((= status 0) empty)
            ((> status 0) (foreign-string-to-lisp buffer))
            (t (error "Cannot read serial... "))))))
```

A.4.3 Hardware Random Number Generator

```
;; (load "serial.lisp")
(defparameter hrng
  (let ((float-scan (cl-ppcre:create-scanner "\\d+(\\.\\d+)?(e[-+]?\\d+)?")))
    (flet ((read-float (str)
             (let ((str (cl-ppcre:scan-to-strings float-scan str)))
               (when str (with-input-from-string (stream str) (read stream))))))
      (lambda ()
        (serial:with-serial (rp "/dev/tty.usbmodem11101")
          (serial:serial-write-string rp (format nil "f~%"))
          (sleep 0.01)
          (read-float (serial:serial-read-string rp)))))))
```

Note: Currently the most slow part is the serial port data transfer process, to transfer data from RP2040 to computer need time, so it would slow down the process. There are possible way to solve this, data compressing and passing multiple random in a data frame to speed up the query process.

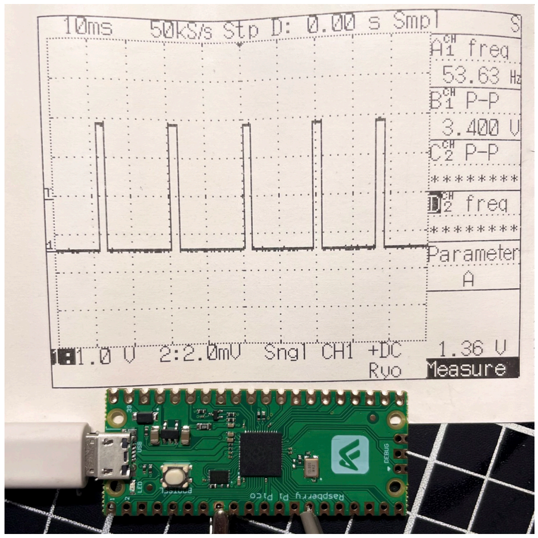

Figure 32: Photo of Homebrew Hardware Random Number Generator, the Oscillator wave shown above is the query process from the computer (HIGH for ADC process), the LOW time is computer reading and waiting for the results.

B Common Lisp Codes

B.1 Post Processing

B.1.1 Smooth

The moving average filtering smooth process could be defined as a working scanner on the given data list:

```lisp
(defun get-arity (f)
  "Get the function `f' arg size. "
  (multiple-value-bind (i ordinary)
      (sb-int:parse-lambda-list
       (sb-introspect:function-lambda-list f))
    (declare (ignore i))
    (length ordinary)))

(defun scan (fn list &optional (step 1))
  "Scan a function `fn' on `list'. "
  (loop with arg-size = (get-arity fn)
        with remain = list
        for i from 0 upto (- (length list) arg-size) by step
        collect (apply fn (subseq remain 0 arg-size))
        do (setf remain (nthcdr step remain))))
```

Listing 6: `scan` function on a given list

for example:

```lisp
(defun %make-uniform-smooth-scanner (size)
  "y_{i, smooth} = (... + y_{i - 1} + y_{i} + y_{i + 1} + ...) / size"
  (let ((args (loop for i below size
                    collect (intern (format nil "Y~d" i)))))
    (eval `(lambda ,args (/ (+ ,@args) ,size)))))

(defun %make-weighted-smooth-scanner (weights)
  "y_{i, smooth} = (... + w_{i - 1} * y_{i - 1} + w_{i} * y_{i} + ...) / sum(ws)"
```

```
  (let ((args (loop for i below (length weights)
                    collect (intern (format nil "Y~d" i)))))
    (eval `(lambda ,args
             (/ (+ ,@(mapcar (lambda (w y) `(* ,w ,y)) weights args))
                ,(reduce #'+ weights))))))

(defun make-smooth-scanner (desc)
  (etypecase desc
    (integer (%make-uniform-smooth-scanner desc))
    (list    (%make-weighted-smooth-scanner desc))))

(defun smooth (list
               &optional (scanner (make-smooth-scanner '(1 2 3 2 1))))
  "Smooth the `list' with smooth `scanner'. "
  (scan scanner list))
```

Listing 7: Moving Average Filtering Algorithm

B.1.2 Stastics

```
(defun mean (list)
  "Calculate `list' mean."
  (float (/ (reduce #'+ list) (length list))))
```

Listing 8: Mean value of a list $E(X) = \frac{\sum X_i}{n}$

```
(defun standard-deviation (list)
  "Calculate the `list' standard deviation."
  (flet ((square (x) (* x x)))
    (sqrt (- (mean (mapcar #'square list))
             (square (mean list))))))
```

Listing 9: Standard Deviation $\sigma = \sqrt{E(X^2) - E(X)}$

B.2 Uniform Test Code

```
(defun n-random-list (n &optional (rand-f (lambda () (random 1.0))))
  "Collect `n' elements with `rand-f' random. "
  (loop for i below n collect (funcall rand-f)))
```

Listing 10: Generate n random numbers distributed between $[0, 1)$ for Uniform Test (section 2.1)

B.3 Periodicity Test Code

B.3.1 Algorithm

See section 4.1 step 1 and 2, generate a random matrix with threshold.

```
(defun make-random-matrix-threshold (n thres
                                     &optional (rand-f (lambda () (random 1.0))))
  "Make a random matrix of size `n' and threshold it with `thres'.
Retun a 2D array matrix with element for 0 and 1. "
  (let ((matrix-1 (make-array (list n n) :initial-element 0))
        (matrix-2 (make-array (list n n) :initial-element 0)))
    (loop for j below n do
      (loop for i below n
            if (> (funcall rand-f) thres)
```

```
               do (setf (aref matrix-1 j i) 1
                        (aref matrix-2 j i) 1)))
    (values matrix-1 matrix-2)))
```

Listing 11: Generate a $n \times n$ matrix and its copy with random 0 and 1 thresholded by `thres`.

for the step 3 depth first search:

```
(defun search-for-route (matrix init-i init-j)
  (destructuring-bind (m n)
      (array-dimensions matrix)
    (let ((m-1 (1- m))
          (n-1 (1- n)))
      (labels ((nexts (i j)
                 (let ((next ()))
                   (when (> i 0)   (push (list (1- i) j) next))
                   (when (> j 0)   (push (list i (1- j)) next))
                   (when (< i n-1) (push (list (1+ i) j) next))
                   (when (< j m-1) (push (list i (1+ j)) next))))
               (cross? (i j)
                 (or (and (= 0 init-i) (= i n-1))
                     (and (= 0 init-j) (= j m-1))
                     (and (= n-1 init-i) (= i 0))
                     (and (= m-1 init-j) (= j 0))))
               (router (i j prev-route)
                 (loop for (next-i next-j) in (nexts i j)
                       for point = (list next-i next-j)
                       for route
                         = (when (and (= (aref matrix next-j next-i) 1)
                                      (not (find point prev-route :test #'equal)))
                             (if (cross? next-i next-j)
                                 (cons point prev-route)
                                 (router next-i next-j (cons point prev-route))))
                       if route
                         return route
                       finally (progn (setf (aref matrix j i) -1)
                                      (return nil)))))
        (when (= (aref matrix init-j init-i) 1)
          (router init-i init-j (list (list init-i init-j))))))))
```

```
(defun search-for-route (matrix n m init-i init-j)
  "Search from point at 'i' and 'j' for a route crossing the 'matrix'.
Return 'nil' if could not find the route,
otherwise for the list of the route."
  (labels ((nexts (i j) ;; find next point near i, j
             (let ((next ()))
               (when (> i 0) (push (list (1- i) j) next))
               (when (> j 0) (push (list i (1- j)) next))
               (when (< i n) (push (list (1+ i) j) next))
               (when (< j m) (push (list i (1+ j)) next))
               next))
           (iter (i j prev-route
                  left-edge? right-edge?
                  top-edge? bottom-edge?) ;; depth first search
             (loop
               for (next-i next-j) in (nexts i j)
               for route
                 = (when (and
                           ;; next i, j is move able
                           (= (aref matrix next-j next-i) 1)
                           ;; and not moved before
                           (not (find (list next-i next-j)
```

```
                                        prev-route :test #'equal)))
                          (if (or
                                  ;; if prev-path passing upper/bottom/right/left edge
                                  (and left-edge?   (= i n))
                                  (and right-edge?  (= i 0))
                                  (and bottom-edge? (= j 0))
                                  (and top-edge?    (= j m)))
                                 ;; if reaches the end
                                 (cons (list next-i next-j) prev-route)
                                 ;; continue to search
                                 (iter next-i next-j
                                       (cons (list next-i next-j) prev-route)
                                       (or (= next-i 0) left-edge?)
                                       (or (= next-i n) right-edge?)
                                       (or (= next-j 0) top-edge?)
                                       (or (= next-j m) bottom-edge?))))
                    if route
                      return route
                    finally (progn (setf (aref matrix j i) -1)
                                   (return nil)))))
    (when (= (aref matrix init-j init-i) 1)
      (iter init-i init-j (list (list init-i init-j))
            ;; left        right          top            bottom
            (= init-i 0) (= init-i n) (= init-j 0) (= init-j m)))))
```

Listing 12: Depth First Search with Dead End Mark acceleration

so if a matrix having a route that could cross the matrix, then this matrix for p and N would be percolating:

```
(defun percolating? (n p &optional (rand-f (lambda () (random 1.0))))
  "Test if square matrix is percolation for `n' and `p'. "
  (multiple-value-bind (matrix-1 matrix-2)
      (make-random-matrix-threshold n p rand-f)
    (loop with size = (1- n)
          for i from 0 below n
          if (or (search-for-route matrix-1 size size i 0)
                 (search-for-route matrix-2 size size 0 i))
            return t)))
```

Listing 13: Test if Square Matrix is percolation. Here the horizontal and vertical test is using copied matrix, this is to avoid missing match patterns.

for step 4, it should do sampling times (default for `100` times):

```
(defun percolating-posibility (n p
                               &optional (sampling 1000)
                                 (rand-f (lambda () (random 1.0))))
  "Return the probability of percolation for given `p' and `N'.

This is the parallel version of `percolation-posibility' function,
if you could not load `lparallel', just replace `lparallel:pdotimes'
to `dotimes' for non parallel version code. "
  (let ((count 0))
    (lparallel:pdotimes (i sampling)
      (declare (ignorable i))
      (when (percolating? n p rand-f)
        (incf count)))
    (float (/ count sampling))))
```

Listing 14: Calculate pobability for percolation on given `n` and `p` with `sampling` samples.

B.3.2 Data

1. $p^{\mathrm{PER}}(N, p)$ for default SBCL `random`

```lisp
(time
 (defparameter percolation-vs-n-p
   (loop for n from 400 downto 10 by 10
         do (print n)
         collect (time
                  (loop for p from 0.0 upto 1.0 by 0.01
                        do (print p)
                        collect (percolating-posibility n p))))
   "Percolation posibility pPER(N, p) with sampling = 1000. "))
```

Listing 15: Generate $p^{\mathrm{PER}}(N, p)$

it would cost around `5000` seconds (around 1.5 hour) on my computer (i7-13700) [10] to calculate the above code snippet:

```
Evaluation took:
  4721.348 seconds of real time
  64117.591650 seconds of total run time (63761.549186 user, 356.042464 system)
  [ Real times consist of 253.890 seconds GC time, and 4467.458 seconds non-GC time. ]
  [ Run times consist of 253.631 seconds GC time, and 63863.961 seconds non-GC time. ]
  1358.04% CPU
  9,971,450,936,347 processor cycles
  4,791,920,449,072 bytes consed
```

the results will be saved in `percolation-vs-n-p.lisp`:

```lisp
(with-open-file (stream out-path :direction :output
                                 :if-does-not-exist :create
                                 :if-exists :supersede)
  (print percolation-vs-n-p stream))
```

Listing 16: Save `percolation-vs-n-p` to local file

for demo usage, just load the pre-calculated file:

```lisp
(with-open-file (stream out-path)
  (defparameter percolation-vs-n-p
    (read stream)))
```

Listing 17: Load `percolation-vs-n-p` from saved file

As you could see for the result at Figure 21, the $p^{\mathrm{PER}}(N, p)$ is trembling less, so just cut off the calculate time with less N would be acceptable.

So the following `percolation-vs-n-p` calculation would be calculated only for N from 100 to 10 by step 10.

2. $p^{\mathrm{PER}}(N, p)$ for `class-dependent`

```lisp
(time
 (defparameter percolation-vs-n-p-class-dependent
   (loop for n from 100 downto 10 by 10
         do (print n)
```

[10] On macbook air m1 this would approximately about 2.5h or so.

```
          collect (time
                   (loop for p from 0.0 upto 1.0 by 0.01
                         do (print p)
                         collect (percolating-posibility
                                   n p 2000 class-dependent))))))
```

Listing 18: Generate $p^{\mathrm{PER}}(N, p)$ for `class-dependent`

3. $p^{\mathrm{PER}}(N, p)$ for `class-independent`

```
(time
 (defparameter percolation-vs-n-p-class-independent
   (loop for n from 100 downto 10 by 10
         do (print n)
         collect (time
                   (loop for p from 0.0 upto 1.0 by 0.01
                         do (print p)
                         collect (percolating-posibility
                                   n p 2000 class-independent))))))
```

Listing 19: Generate $p^{\mathrm{PER}}(N, p)$ for `class-independent`

4. $p^{\mathrm{PER}}(N, p)$ for `lcg-zx81`

```
(time
 (defparameter percolation-vs-n-p-lcg-zx81
   (loop for n from 100 downto 10 by 10
         do (print n)
         collect (time
                   (loop for p from 0.0 upto 1.0 by 0.01
                         do (print p)
                         collect (percolating-posibility
                                   n p 2000 lcg-zx81))))))
```

Listing 20: Generate $p^{\mathrm{PER}}(N, p)$ for `class-independent`

5. $p^{\mathrm{PER}}(N, p)$ for `lcg-ranqd1`

```
(time
 (defparameter percolation-vs-n-p-lcg-ranqd1
   (loop for n from 100 downto 10 by 10
         do (print n)
         collect (time
                   (loop for p from 0.0 upto 1.0 by 0.01
                         do (print p)
                         collect (percolating-posibility
                                   n p 2000 lcg-ranqd1))))))
```

Listing 21: Generate $p^{\mathrm{PER}}(N, p)$ for `class-independent`

B.4 Cluster Test Code

Find a cluster at position (Algorithm described in 5.1:

```
(defun search-for-a-cluster (matrix n m init-i init-j)
  "Find a cluster for `matrix' at `init-i', `init-j' with boundary `n', `m'. "
  (let ((cluster ()))
    (labels ((nexts (i j) ;; find next point near i, j
               (let ((next ()))
                 (when (> i 0) (push (list (1- i) j) next))
```

```
                 (when (> j 0) (push (list i (1- j)) next))
                 (when (< i n) (push (list (1+ i) j) next))
                 (when (< j m) (push (list i (1+ j)) next))
                 next))
             (iter (i j)
               (loop for (next-i next-j) in (nexts i j)
                     if (= (aref matrix next-j next-i) 1)
                       do (setf (aref matrix next-j next-i) -1)
                       and do (iter next-i next-j)
                     finally (push (list i j) cluster))))
      (when (= (aref matrix init-j init-i) 1)
        (setf (aref matrix init-j init-i) -1)
        (iter init-i init-j)
        (loop with size = (length cluster)
              for (i j) in cluster
              do (setf (aref matrix j i) size))
        (values (length cluster) cluster)))))
```

```
(defun search-for-cluster (matrix)
  "Find a cluster on `matrix'. "
  (destructuring-bind (m n)
      (array-dimensions matrix)
    (loop with stastics = ()
          for j below m
          do (loop for i below n
                   for cluster = (search-for-a-cluster matrix (1- n) (1- m) i j)
                   ;; count the cluster sizes
                   if cluster
                     do (push cluster stastics))
          finally (return (values matrix stastics)))))
```

B.5 Plot

B.5.1 Uniform Test Plot

```
(define-presentation label (base-presentation)
  ((%text :initarg :text :initform ""))
  (:draw (%text) (draw-text self 0 0 %text :char-spacing 1.2)))
```

Plot for uniform test:

```
(let ((out-path (format nil "n-~d-uniform-by-~a.png"
                        n (or method :default))))
  (with-present-to-file
      (plot plot :margin 10
                 :x-min 0 :x-max 1
                 :y-min 0 :y-max (* (/ 1.5 bins) n))
      (out-path)
    (let* ((data (let ((method (if (stringp method)
                                   (intern (string-upcase method))
                                   method)))
                   (if method
                       (eval `(n-random-list ,n ,method))
                       (n-random-list n))))
           (sigma nil))
      (add-plot-pane plot 'uniform-hist
                     (with-present (hist histogram-pane
                                         :plot-data data
                                         :bins bins
                                         :color +草白+)
```

```
                      (let* ((plot-data (slot-value hist
                                             'gurafu/plot::%plot-data))
                             (hist-data (mapcar #'second plot-data))
                             (s (standard-deviation hist-data)))
                        (setf sigma (format nil " = ~,3f; relative  = ~,3f"
                                            s (/ s (mean hist-data)))))))
      (add-component plot 'stastics
                     (with-present (label label :text sigma)) 0.05 0.02)))
  out-path)
```

B.5.2 Independent Test Plot

Plot for the independent test:

```
(let ((out-path (format nil "independent-test-~a.png" method))
      (method (let ((method (if (stringp method)
                                (intern (string-upcase method))
                                method)))
                (if method
                    (eval `(lambda ()
                             (list (funcall ,method) (funcall ,method))))
                    (lambda () (list (random 1.0) (random 1.0)))))))
  (with-present-to-file
      (plot plot :margin 10
                 :x-min 0 :x-max 1
                 :y-min 0 :y-max 1)
      (out-path)
    (add-plot-data plot
        (scatter-pane test :color +草白+)
      (loop for i below n
            collect (funcall method))))
  out-path)
```

B.5.3 Search Route Process Plot

Plot the searching route process:

```
(with-present-to-file
    (plots horizontal-layout-presentation :width 1000 :height 400)
    (out-path :width 850 :height 400)
  (multiple-value-bind (matrix-1 matrix-2)
      (make-random-matrix-threshold n p)
    (add-component plots 'matrix
                   (with-present (matrix plot :x-min 0 :x-max n
                                              :y-min 0 :y-max n
                                              :x-label "i" :y-label "j")
                     (add-plot-data matrix
                         (2d-grid-pane matrix :z-min 0)
                       (loop for j below n
                             collect (loop for i below n
                                           collect (aref matrix-1 j i)))))
                   0.5)
    (let ((the-route (loop with n = (1- n)
                           for i from 0 upto n
                           for route
                             = (or (search-for-route matrix-1 n n i 0)
                                   (search-for-route matrix-2 n n 0 i))
                           if route return route)))
      (add-component
       plots 'route
```

```
        (with-present (route plot :x-min 0 :x-max n
                                  :y-min 0 :y-max n
                                  :x-label "i" :y-label "j")
          (add-plot-data route
              (2d-grid-pane matrix :z-min -1
                                   :color (gurafu/core:linear-color-map
                                           +大红+ +white+ +white+ +草白+))
            (loop for j below n
                  collect (loop for i below n
                                for mat-1 = (aref matrix-1 j i)
                                for mat-2 = (aref matrix-2 j i)
                                collect (if (or (= mat-1 -1) (= mat-2 -1))
                                            -1 mat-1))))
          (when the-route
            (add-plot-data route
                (line-plot-pane route :y-min 0 :y-max n
                                     :color +紫+
                                     :line-width 5)
              (loop for (i j) in the-route
                    collect (list (+ i 0.5) (- n j 0.5))))))
        0.5))))

out-path
```

B.5.4 $p^{\mathrm{PER}}(N, p)$ Plot

Plot the $p^{\mathrm{PER}}(N, p)$:

```
(let ((out-path (format nil "plot-percolation-vs-n-p-~a.png" method)))
  (with-present-to-file
      (plot plot :margin 10 :x-min 0 :x-max 1 :y-min 10 :y-max 400)
      (out-path)
    (add-plot-data plot
        (2d-grid-pane percolation-p
                      :z-min 0 :y-min 10
                      :color (gurafu/core:linear-color-map
                             +翠蓝+ +white+ +水红+))
      (if method
          (eval (intern (string-upcase
                        (format nil "percolation-vs-n-p-~a" method))))
          percolation-vs-n-p)))
  out-path)
```

B.5.5 $p_{\mathrm{c}}(N)$ Plot

Calculate $p_{\mathrm{c}}(N)$ and plot it:

```
(let ((out-path (format nil "plot-pc-vs-n-~a.png" method))
      (data (if method
                (eval (intern (string-upcase
                              (format nil "percolation-vs-n-p-~a" method))))
                percolation-vs-n-p)))
  (with-present-to-file
      (plot plot :margin 10 :x-min 10 :y-min 0.25 :y-max 0.5
                 :x-label "N" :y-label "pc")
      (out-path)
    (let ((p 0.00))
      (flet ((dp (yp-1 yp yp+1)
               (incf p 0.01)
               (list p (/ (- yp+1 yp-1) (* 2 0.01)))))
```

```
        (add-plot-data plot
            (line-plot-pane pc :color +草白+)
          (loop for p-per in (reverse data)
                for n from 10 by 10
                collect (list n (caar (sort (scan #'dp p-per)
                                            #'< :key #'second)))
                do (setf p 0.00))))))
  out-path)
```

B.5.6 $p^{\mathrm{PER}}(N,p)$ and $p_{\mathrm{c}}(N)$ Plot Together

```
(let* ((out-path (format nil "plot-p-per-pc-~a.png" method))
       (data (eval (intern (string-upcase
                            (format nil "percolation-vs-n-p-~a" method)))))
       (pc-data (loop for p-per in (reverse data)
                      for n from 10 by 10
                      collect (let ((p 0.0))
                                (flet ((dp (yp-1 yp yp+1)
                                         (declare (ignore yp))
                                         (incf p 0.01)
                                         (list p (/ (- yp+1 yp-1) (* 2 0.01)))))
                                  (list n (caar (sort (scan #'dp p-per)
                                                      #'< :key #'second)))))))
       (pc-to (second (first (last pc-data))))
       (pc-label (format nil "~a pc → ~,3f" method pc-to)))
  (with-present-to-file
      (plots horizontal-layout-presentation :width 800 :height 400)
      (out-path :width 800 :height 400)
    (add-component plots 'p-per
                   (with-present
                       (plot plot :margin 10
                                  :x-label "p" :y-label "N"
                                  :x-min 0 :x-max 1
                                  :y-min 10 :y-max 100)
                     (add-plot-data plot
                         (2d-grid-pane percolation-p
                                       :z-min 0 :y-min 10
                                       :color (gurafu/core:linear-color-map
                                               +翠蓝+ +white+ +水红+))
                       data))
                   0.5)
    (add-component plots 'pc
                   (with-present
                       (plot plot :margin 10
                                  :x-min 10
                                  :x-label "N" :y-label "pc")
                     (add-plot-data plot
                         (line-plot-pane pc-to :color +草白+
                                         :y-max 0.5 :y-min 0.25)
                       `((10 ,pc-to) (100 ,pc-to)))
                     (add-plot-data plot
                         (line-plot-pane pc :color +莲红+)
                       pc-data)
                     (add-component plot 'label
                                    (make-instance 'label :text pc-label)
                                    0.1 (- 1.0 pc-to)))
                   0.5))
  out-path)
```

B.5.7 Matrix Cluster Plot

Plot a cluster on matrix:

```lisp
(defun make-colorful-color-map ()
  (let ((linear (gurafu/core:linear-color-map
                  +草白+ +天蓝+ +葡萄青+
                  +木红+ +茶褐+ +莲红+)))
    (lambda (v)
      (if (zerop v) +white+ (funcall linear v)))))
```

```lisp
(with-present-to-file
    (plots horizontal-layout-presentation :width 1000 :height 400)
    (out-path :width 850 :height 400)
  (multiple-value-bind (matrix-1 matrix-2)
      (make-random-matrix-threshold n p)
    (flet ((->list (mat)
             (loop for j below n
                   collect (loop for i below n
                                 collect (aref mat j i))))
           (z-max (mat)
             (loop with max = 1
                   for j below n
                   do (loop for i below n
                            if (> (aref mat j i) max)
                              do (setf max (aref mat j i)))
                   finally (return max))))
      (add-component plots 'matrix
                     (with-present (matrix plot :x-min 0 :x-max n
                                                :y-min 0 :y-max n
                                                :x-label "i" :y-label "j")
                       (add-plot-data matrix
                           (2d-grid-pane matrix :z-min 0)
                         (->list matrix-1)))
                     0.5)
      (search-for-cluster matrix-2)
      (add-component plots 'cluster
                       (with-present (route plot :x-min 0 :x-max n
                                                 :y-min 0 :y-max n
                                                 :x-label "i" :y-label "j")
                         (add-plot-data route
                             (2d-grid-pane matrix
                                           :z-min 0
                                           :z-max (z-max matrix-2)
                                           :color (make-colorful-color-map))
                           (->list matrix-2)))
                       0.5))))

out-path
```

B.5.8 Cluster Histogram Plot

Plot for cluster histogram of $n \times n$ matrix at different p:

```lisp
(let ((out-path (format nil "plot-cluster-histogram-n-~d.png" n)))
  (with-present-to-file
      (plot plot :margin 10 :x-min 0 :y-max 50
            :y-label "Counts" :x-label "Size")
      (out-path :width 400)
    (flet ((cluster (p color)
             (let ((matrix (make-random-matrix-threshold n p)))
```

```
                  (multiple-value-bind (mat stastics)
                      (search-for-cluster matrix)
                    (declare (ignore mat))
                    (with-present (cluster-hist histogram-pane
                                                :x-min 0
                                                :x-max 500
                                                :y-max 50
                                                :bins bins
                                                :plot-data stastics
                                                :color color)
                      ;; (change-class cluster-hist 'line-plot-pane)
                      )))))
    (loop for p in ps
          for c in (list +草白+ +水红+ +天蓝+ +茶褐+ +莲红+)
          do (add-plot-pane plot (gensym) (cluster p c)))
    (set-xy-bounding-box plot 0 xmax 0 ymax)))
  out-path)
```