Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación

Lecturas en Ciencias de la Computación
ISSN 1316-6239

rr_porfiado
un generador de números seudo aleatorios

Rafael Pastoriza

RT 2009-05

Centro CIOMMA

Caracas, Marzo, 2009

RT-2009-05
Lecturas en Ciencias de la Computación
ISSN 1316-6239

rr_porfiado
un generador de números seudo aleatorios

R. A. Pastoriza*
rpastoriza@cantv.net

marzo 2009

Abstract

We introduce *rr_porfiado* a recursive pseudo random generator based in a real nonlinear transformation which also is not invertible. A C code is included. The empirical test show it has excellent statistical properties.

Resumen

Se introduce *rr_porfiado* un generador recursivo de números reales seudo aleatorios basado en transformaciones no lineales, y no invertible. Se incluye su implantación en C. Las pruebas empíricas realizadas con *Tuftest* permiten concluir que tiene excelentes propiedades estadísticas.

Palabras clave: pseudo random generator, random

*CIOMMA. .Centro de Investigación de Operaciones y Modelos Matemáticos Aplicados, Escuela de Computación, Facultad de Ciencias, Universidad central de Venezuela, Los Chaguaramos, Apartado 47002, Caracas, 1041-A, Venezuela.

Resumen

1.- Introducción

En un trabajo previo el autor, [3] Pastoriza, presentó un análisis de diferentes generadores de números seudo aleatorios que pueden ser utilizados en criptografía. Ahí aparece descripto un generador de números seudo aleatorios, GNSA, criptográficamente resistente *porfiadoUL_CR*.

En este trabajo se reporta otro generador, *rr_porfiado*, que es computacionalmente más liviano que el descripto anteriormente – *porfiadoUL_CR* - y que incorpora un mecanismo distinto de inicialización. Esto último en atención a tratar de conseguir que las sucesiones de números generados sean sustancialmente distintas cuando se inicializa el generador con números cercanos. No se incluyen aquí la reinicialización aleatoria ni el uso de una tabla de mezcla, como tampoco la decimación de la sucesión; esto es todas aquellas previsiones que conducen a configurar un GNSA criptográficamente resistente .

Ambos generadores son extensiones de un generador representante de una clase muy amplia de generadores seudo aleatorios basados en transformaciones en números reales introducida y estudiada por [1] Lavieri en 1984.

2.- Descripción de *rr_porfiado*

Se inicializa el algoritmo con un número real *u* de entrada. Se emplean las funciones trigonométricas *seno* y *coseno*, con sus argumentos amplificados, con el fin de separar los valores resultantes ante dos preimágenes cercanas. Luego se calculan las mantisas de las expresiones cuadráticas resultantes para devolver, por primera vez como valor generado la mantisa de la suma calculada.

La generación en si se apoya en la no linealidad de las expresiones recursivas:

$$x_{n+1} = 151.4751/(x_n+G_{n+1})$$
$$y_{n+1} = 173.4789/(y_n+F_{n+1})$$

calculadas sobre x_n y y_n , y las mantisas de 100 veces sus valores anteriores.

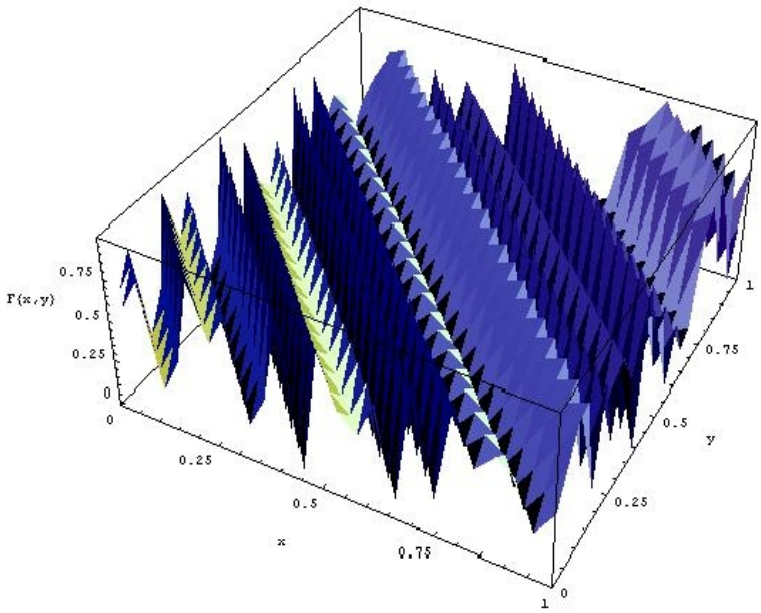
Las constantes 151.4751 y 173.4789 han sido sugeridas por las pruebas realizadas.

La naturaleza no lineal, y no invertible, de la función empleada en este algoritmo se puede visualizar en los gráficos siguientes.

En términos funcionales la recursión puede ser vista como la función:

f(x,y) = (mantisa[mantisa[mantisa[17347.89/(y+mantisa[x])]] + mantisa[mantisa[15147.51/(x +mantisa[y])]]])

cuyo gráfico es:



La sola observación del mismo nos indica que es imposible la inversión, o sea dado un cierto valor de *f(x,y)* conseguir cuales pares *(x,y)* tienen ese valor como imagen. Aspecto este último de marcada importancia en criptografía.

El algoritmo *rr_porfiado* es:

Algoritmo rr_porfiado

entradas:
u

1. Inicialización

$x_1 = (1.0 + sin(1000*u))$
$y_1 = (1.0 + cos(1000*u))$
$F_1 = mantisa(100*3.0*x_1^2 + 1.0)$
$G_1 = mantisa(100*5.0*y_1^2 + 5.0)$
$z_1 = mantisa(F_1 + G_1)$

devolver z_1

2. *Generar recursivamente:*

$F_{n+1} = mantisa(100*x_n)$
$G_{n+1} = mantisa(100*y_n)$
$x_{n+1} = 151.4751/(x_n + G_{n+1})$
$y_{n+1} = 173.4789/(y_n + F_{n+1})$
$z_{n+1} = mantisa(F_{n+1} + G_{n+1})$

devolver z_{n+1}

salidas:
z_i

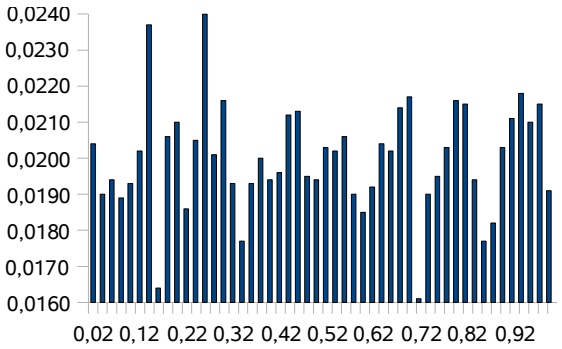
donde *mantisa(u)*indica la parte decimal del argumento real *u*.

El código en C aparece en el apéndice 1.

3.- Resultados

Los gráficos siguientes permiten tener una idea del comportamiento seudo aleatorio de las sucesiones numéricas [0,1] generadas por el algoritmo *rr_porfiado*.

Primero se puede observar un histograma con las frecuencias relativas de los números u_i. de una sucesión de 10000 números, inicializados al azar



En segundo término se puede observar un gráfico en dos dimensiones de los pares (u_i,u_{i+1}) de la misma sucesión de 10000 números.

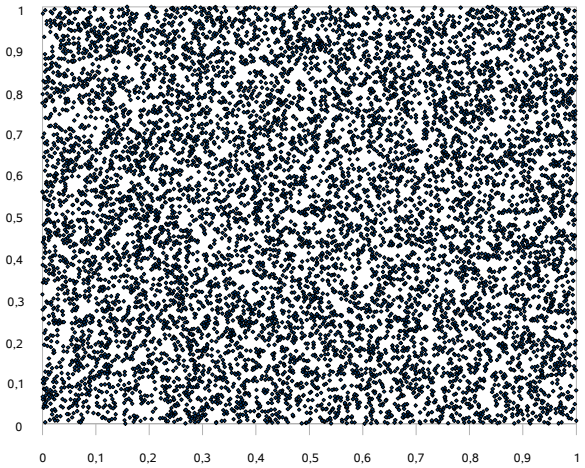
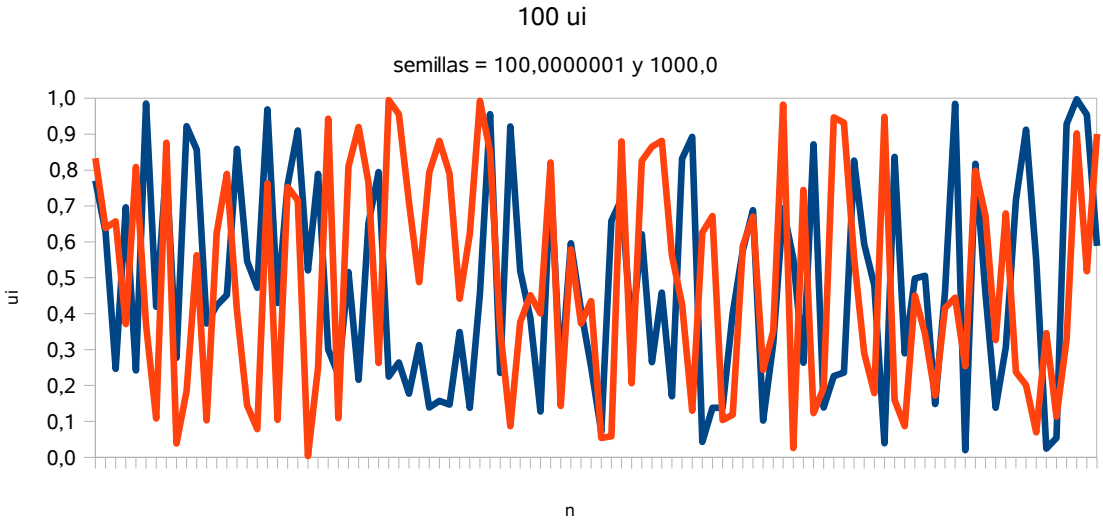


Imagen en la cual no se visualizan patrones reconocibles.

En el gráfico siguiente se puede observar la conducta de dos sucesiones con inicialización cercanas.

Se recolectaron 100 valores consecutivos generados por *rr_porfiado* con las inicializaciones cercanas 100.0 y 100.0000001.



Como se puede observar las dos sucesiones difieren significativamente en los valores generados.

4.- Pruebas empíricas de aleatoriedad

La prueba de bondad de ajuste de Kolmorov-Smirnov, a dos lados, para la sucesión de 10000 números seudo aleatorios en el intervalo [0,1] confirma la uniformidad de la sucesión estudiada.

El estadístico $T_1 = \sup(\text{abs}(S_N(x) - F_X(x)))$, a dos colas, es de

0,005308

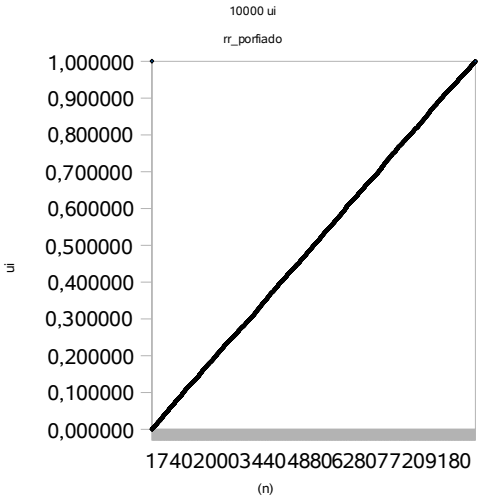
bajo la hipótesis nula de que la supuesta función de distribución $F_X(x)$ es uniforme [0,1].

El valor crítico de tamaño $\alpha = 0.01$ es

0,016300

En base a estos valores no se puede rechazar la hipótesis nula de que la muestra proviene de una población que se distribuye uniformemente en[0,1].

La ilustración siguiente proporciona una idea gráfica de la conducta uniforme de la sucesión.



Se hicieron 101 pruebas aleatorias e independientes de rr*_porfiado* con la batería de pruebas *Tuftest* de [2] G. Marsaglia usando inicializaciones al azar e independientes del generador, con valores iniciales en los rangos indicados:

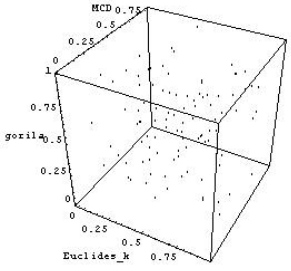
$v \in [-100.0,100.0]$.

Los valores decimales se llevaron a enteros largos sin signo UL, y es sobre estos que se realiza la prueba ya que *Tuftest* requiere un GNSA que arroje este tipo de números enteros largos .

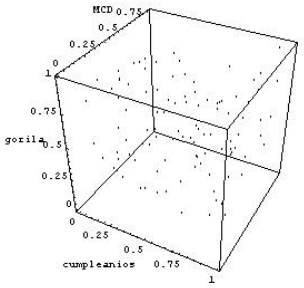
Los valores resultantes se pueden consultar en el apéndice 2.

Los gráficos siguientes proporcionan una visión de los valores estadísticos obtenidos.

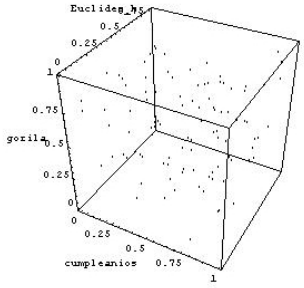
Dado que en cada prueba de *Tuftest* se obtienen 4 estadísticos, a saber *cumpleaños*, *Euclides k*, *MCD*, y *gorila* se elaboraron 4 vistas 3D proyectando en cada caso una variable de la cuaterna.



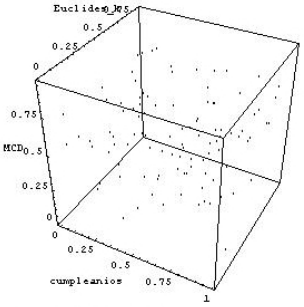
rr_porfiado 102 pruebas independientes
proyeccion cumpleanios --> (, Euclides_k, MCD, gorila)



rr_porfiado 102 pruebas independientes
proyeccion Euclides_k --> (cumpleanios, ,MCD, gorila)



rr_porfiado 102 pruebas independientes
proyeccio MCD --> (cumpleanios, Euclides_k, ,gorila)



rr_porfiado 102 pruebas independientes
proyeccion gorila --> (cumpleanios, Euclides_k, MCD,)

No aparecen patrones significativos en los gráficos anteriores.

Hay 2 resultados fuera del rango [0.01,0.99] y ambos corresponden a la prueba del cumpleaños.

Para hacer mas interpretables los resultados estos valores estadísticos se categorizaron según la descripción que sigue:

-1 cuando se van del rango permitido; 1 a 5 con valores permitidos.

El rango entero total es {-1, [4,20]} privilegiando los valores centrales según la siguiente función *rango*:

valor = *rango*(***p***) es:

Si ***p*** es un estadístico de *Tuftest* entonces

Si ***p*** ≤ *0.01 o* ***p*** ≥ *0.99* entonces *valor = -1*
en caso contrario
Si ***p*** ≤ *valor = [10*****p****] +1*
caso contrario *valor = 10-[10*****p****]*

[x] es la parte entera de *x*.

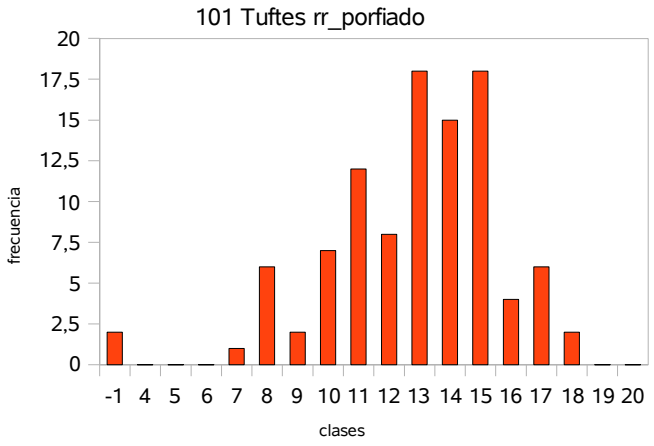
Esto categorización se hizo en atención a que valores menores a 0.01 o superiores a 0.99 indican que la sucesión examinada, y por consiguiente el GNSA que las ha generado, es inaceptable.

Los resultados obtenidos y la distribución de frecuencias absolutas se pueden apreciar en la tabla siguiente.

frec	2	0	0	0	1	6	2	7	12	8	18	15	18	4	6	2	0	0
clases	-1	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Como se puede observar sólo hay 2 casos en donde se obtuvo un resultado cuestionable. Su moda es 13 y 15.

En el gráfico siguiente se presenta el correspondiente histograma.



Estos resultados permiten asegurar que el generador *rr_porfiado* tiene buenas propiedades estadísticas.

4.- Conclusiones

Se introduce *rr_porfiado* un nuevo generador de números seudo aleatorios basado en transformaciones en números reales ,no lineales y recursivas.

Se da una descripción algorítmica del algoritmo y su implantación en C.

Se exploraron las propiedades de *rr_porfiado* sin encontrase anomalías.

Se comprobaron mediante *Tuftest* las buenas propiedades estadísticas del generador descripto.

En conclusión se dispone de un nuevo generador de números seudo aleatorios adecuado para su empleo en Simulación.

Dado que no es posible su reversión también se puede considerar para su utilización en Criptografía, en aquellas aplicaciones que no requieran máxima seguridad

Referencias

[1] R. A. Lavieri (1984), “Exploración de nuevas ideas para generación de números pseudo-aleatorios” Trabajo de ascenso. Universidad Central de Venezuela, Facultad de Ciencias, Escuela de Computación. .

[2] G . Marsaglia, and Wai Wan Tsang (2006), “Some difficult-to-pass tests of randomness”.

[3] R.A. Pastoriza (2008), “Generadores de números seudoa aleatorios y criptografía”. RT-2008-07. Universidad Central de Venezuela. Facultad de Ciencias. Escuela de Computación. CIOMMA.

Apéndice 1 .- código en C de l algoritmo rr_porfiado

El código que aparece a continuación es la versión en entero largo

```
/********************* rr_porfiado ********************************/
/* Este archivo contiene una simple función para implantar rr_porfiado un
   algoritmo de generacion de números seudo aaleatorios basado en la
   versión en doble precision del algoritmo porfiado bidimensional de
   Lavieri,R..

   rr_porfiado implanta una version ligeramente diferente de porfiado.

   Primero se inicializa con un solo número real. Se utiliza funciones
   trigonometricas y cuadraticas para separar entradas con valores
   cercanos.
   Se usa una transformación no lineal similar a la sugerida por Lavieri.

   Algoritmo rr_porfiado es el siguiente:

entradas:
                    u: número real en doble precisión

1. Inicialización

                    x1 = (1.0 + sin(1000*u))
                    y1 = (1.0 + cos(1000*u))
                    F1 =  mantisa(100*3.0*x12+ 1.0)
                    G1 =  mantisa(100* 5.0*y12 + 5.0)
                    z1  =  mantisa(F1 + G1)

                    devolver  z1: número real en doble precisión

 2.  Generar recursivamente:

                    Fn+1 = mantisa(100*xn)
                    Gn+1 = mantisa(100*yn)
                    xn+1 = 151.4751/(xn+Gn+1)
                    yn+1 = 173.4789/(yn+Fn+1)
                    zn+1 =  mantisa(Fn+1 + Gn+1)

                    devolver  zn+1: número real en doble precisión

   donde mantisa(u)indica la parte decimal del argumento u.


   programador R.A. Pastoriza       dic/2008
*/
```

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

/*  valor inicial cualquier numero real expresado en doble precision  */

/*              double v = 0.1;       */

/*********************************************************/

        unsigned long rr_porfiado(double u)
        { /*    versión entero largo sin signo UL    */
/*  constantes del metodo  */
                double A=  151.4751;
                double B=  173.4789;
                static int vez = 1;
                static double x, y;
                double Fx, Gy;
                double z;

            if(vez==1)
                { /*  inicializacion  */
/*  separar entradas cercanas   */
                x = (1.0 + sin(1000*u));
                y = (1.0 + cos(1000*u));
                Fx = (100*3.0*x*x + 1.0) - (int) (100*3.0*x*x + 1.0);
                Gy = (100* 5.0*y*y + 5.0) - (int) (100* 5.0*y*y + 5.0);
                z = (Fx + Gy) - (int)(Fx + Gy); /*  mantisa  */
                vez = 2;
                return 4294967295UL*z;
                }
             else
                { /*  generacion  */
/*  mantisas desde tercera posicion   */
                Fx = (100*x) - (int)(100*x);
                Gy = (100*y) - (int)(100*y);
/*  funcion nolineal  */
                x = A/(x+Gy);
                y = B/(y+Fx);
                z = (Fx + Gy) - (int)(Fx + Gy);
                return 4294967295UL*z;
                }
        }

/********************** fin rr_porfiado ******************************/
```

Apéndice 2

En esta sección se incluyen los resultados de las 101 pruebas a *rr_porfiado,* inicializadas con semillas independientes, con *Tuftest*.

tuftest_rr_porfiado2

	cumpleaños	gcd	Euclides	gorila
1	0.808	0.569731	0.464774	0.853
2	0.800	0.837424	0.825171	0.110
3	0.626	0.798528	0.757199	0.046
4	0.294	0.087524	0.647947	0.022
5	0.297	0.479061	0.524188	0.681
6	0.190	0.308156	0.534432	0.800
7	0.565	0.154719	0.646669	0.753
8	0.274	0.350589	0.788249	0.788
9	0.579	0.498588	0.510512	0.118
10	0.441	0.044949	0.776148	0.501
11	0.208	0.305789	0.556732	0.715
12	0.759	0.708084	0.988038	0.780
13	0.900	0.621102	0.521789	0.201
14	0.122	0.660365	0.622877	0.680
15	0.266	0.085940	0.286621	0.012
16	0.895	0.081447	0.540280	0.540
17	0.098	0.281161	0.138774	0.904
18	0.011	0.573330	0.542171	0.111
19	0.550	0.109457	0.802003	0.896
20	0.243	0.162435	0.529651	0.480
21	0.864	0.895466	0.351102	0.228
22	0.653	0.307659	0.338271	0.510
23	0.722	0.571873	0.351223	0.591
24	0.759	0.097882	0.116858	0.159
25	0.685	0.618434	0.163861	0.202
26	0.406	0.658387	0.686514	0.497
27	0.455	0.623769	0.180214	0.072
28	0.149	0.151065	0.707466	0.942
29	0.427	0.697905	0.442909	0.733
30	0.662	0.321588	0.296072	0.796
31	0.620	0.220878	0.613426	0.041
32	0.442	0.095994	0.700623	0.189
33	0.185	0.675647	0.859358	0.841
34	0.396	0.937887	0.439878	0.124
35	0.186	0.289012	0.554631	0.033
36	0.354	0.542004	0.368104	0.403
37	0.650	0.230260	0.404554	0.760
38	0.760	0.928155	0.283398	0.280
39	0.742	0.750621	0.630124	0.688
40	0.019	0.612775	0.935231	0.426
41	0.632	0.820524	0.795802	0.097
42	0.850	0.688530	0.693351	0.787
43	0.442	0.674825	0.361729	0.839
44	0.097	0.892507	0.680565	0.740
45	0.569	0.654016	0.735647	0.101
46	0.379	0.613658	0.198163	0.025
47	0.592	0.543166	0.791089	0.890
48	0.373	0.172165	0.734754	0.228
49	0.352	0.342194	0.527972	0.074
50	0.485	0.533992	0.459334	0.057
51	0.379	0.715902	0.293110	0.011
52	0.744	0.719093	0.857667	0.458
53	0.810	0.223747	0.497450	0.533
54	0.598	0.936660	0.329534	0.775
55	0.563	0.371513	0.281295	0.305
56	0.141	0.683013	0.024417	0.060
57	0.549	0.222771	0.637816	0.769
58	0.652	0.386393	0.306829	0.279
59	0.030	0.724409	0.215007	0.968
60	0.001	0.825635	0.531916	0.255
61	0.509	0.392777	0.657611	0.229
62	0.673	0.854738	0.590064	0.847
63	0.468	0.112529	0.705075	0.583
64	0.570	0.763953	0.765572	0.201
65	0.445	0.192926	0.245327	0.456
66	0.026	0.451410	0.637680	0.985
67	0.408	0.746082	0.843531	0.446
68	0.284	0.262287	0.447072	0.344
69	0.769	0.063174	0.469455	0.817
70	0.136	0.214022	0.351726	0.443
71	0.508	0.183018	0.421105	0.212
72	0.278	0.703961	0.627258	0.283
73	0.190	0.625896	0.558240	0.743
74	0.100	0.568037	0.786939	0.586
75	0.054	0.790890	0.722479	0.197
76	0.167	0.984388	0.373824	0.258
77	0.514	0.839025	0.372817	0.373
78	0.371	0.143062	0.492585	0.810
79	0.332	0.691892	0.628624	0.937
80	0.232	0.501364	0.141302	0.511
81	0.791	0.596570	0.710663	0.370
82	0.564	0.832072	0.372742	0.724
83	0.460	0.515249	0.838626	0.866
84	0.129	0.671967	0.595339	0.927
85	0.845	0.439331	0.849224	0.345
86	0.698	0.869251	0.736459	0.312
87	0.712	0.969769	0.673809	0.631
88	0.791	0.160327	0.766852	0.618
89	0.288	0.118867	0.493520	0.241
90	0.760	0.407892	0.635485	0.531
91	0.617	0.650303	0.165927	0.651
92	0.166	0.579643	0.435180	0.824
93	0.002	0.170836	0.040747	0.056
94	0.507	0.803800	0.823999	0.010
95	0.856	0.356877	0.744255	0.245
96	0.710	0.787822	0.124302	0.750
97	0.473	0.468678	0.651033	0.063
98	0.536	0.784360	0.913774	0.134
99	0.600	0.418478	0.189019	0.557
100	0.753	0.736006	0.746472	0.693
101	0.632	0.268515	0.934378	0.486

final tuftest_rr_po<===