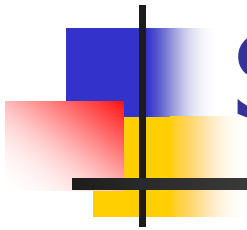


SASの一様乱数の話

本日の話のきっかけ

```
data for_t_test ;
  retain seed1 100;
  retain seed2 100;
  do i=1 to 50;
    x =
   rannor(seed1) ;
    y =
   rannor(seed2) ;
    output ;
  end ;
run ;
```

- 例えば2群の平均の比較を行うシミュレーションデータを作ることを考える
- ある分布に従う乱数を多数生成する際，
「各群の乱数のシードを同じにしているのに，なぜ同じ乱数列が出来ないの？」
という疑問が湧いた･･･
- 本日は乱数に関する実験や調査の結果を報告する

本日のメニュー

- **seed 列の仕様（その説明）**
- ranuni と call ranuni の違い
- seed 列の仕様（その理由）
- rand 関数について
- 例

（準備）一様乱数とは？

- **一様乱数列**：0 から 1 までの間の実数が以下の 2 つの性質に従って並んでいる数列のこと
  - **等確率性**：どの実数も同じ頻度で現れる
  - **無規則性**：数列に規則性はない
    → 等確率性だけでは、等差数列も許されることになる！
- 一様乱数列の一つ一つの数を一様乱数と呼ぶ
- SAS で作ることが出来る一様乱数は等確率性と無規則性をほぼ完全に満たしている（はず）

SAS のマニュアルより

- call ranuni ルーチンは，区間(0,1)上の一様乱数を与える（$2^{31}-1$個の異なる実数から1つ）
- 乱数生成時にseed（種）を与える
- 1つのDATAステップ中に複数の ranuni 関数があっても，シード値列は1つのみ

⇒ 複数の乱数列を作ることは可能だが，それぞれの乱数は1つのシード値列から生成される

- ただし，call ranuni ステートメントを使うと，複数の乱数列を別々に初期設定することが出来る

シード列と乱数列の違い

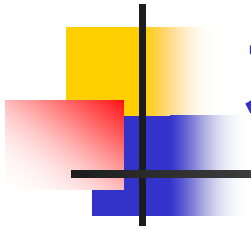
- シード列：乱数を生成するための数列

⇒ 例えば次のような数列（初期値は100 ）

100，　10億，1900万，　21億，・・・

- 乱数列：シード列を $2^{31}-1$ で割ったもの

0.496，　0.009，　0.982，・・・



本日のメニュー

- seed 列の仕様（その説明）
- **ranuni と call ranuni の違い**
- seed 列の仕様（その理由）
- rand 関数について
- 例

サンプルプログラム

```
data case ;
  retain seed1 seed2 seed3 100;
  do i=1 to 10 ;
    call ranuni(seed1, x1) ;
    call ranuni(seed2, x2) ;
    x3 = ranuni(seed3) ;
    if i=4 then do;
      seed2 = 200;
      seed3 = 200;
    end ;
    if i=6 then do;
      seed3 = 100;
    end ;
    if i=8 then do;
      seed1 = 100;
    end ;
    output ;
  end ;
run ;
```

← ① seed1 ～ seed3 に 100 を入れる

← x1 ～ x3 についてそれぞれ 10 個の乱数を生成するのだが・・・

← ② x1 と x2 は call ranuni で，x3 は単なる ranuni で乱数を生成

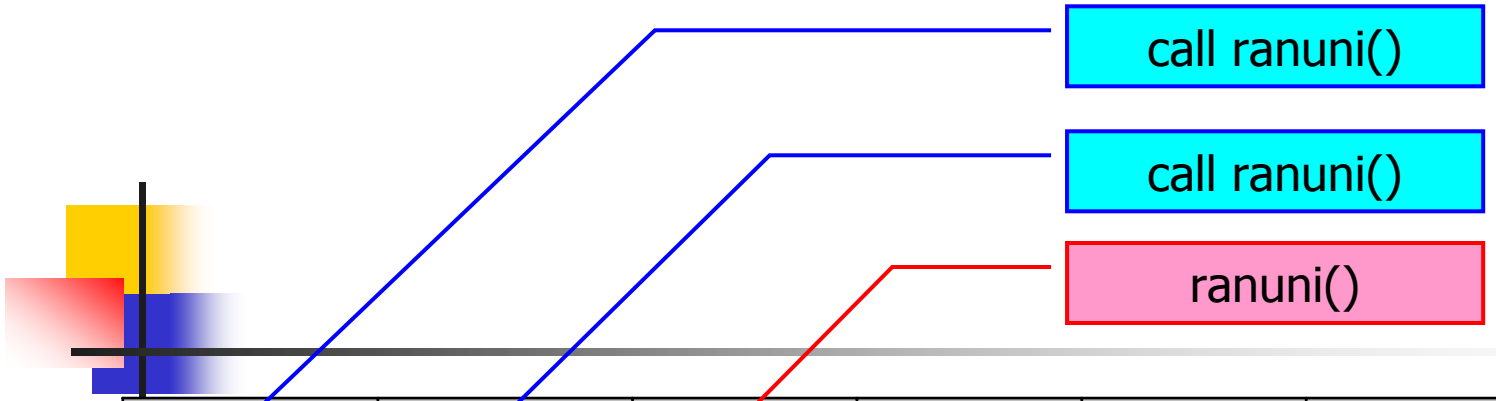
← ③ i = 4 の時，seed2 と seed3 の値を 200 に変更してみる

← ④ i = 6 の時，seed3 の値を 100 に戻してみる

← ⑤ i = 8 の時，seed1 の値を 100 に戻してみる

生成された乱数列

seed1	seed2	seed3	i	x1	x2	x3
1.07E+09	1.07E+09	100	1	0.496	0.496	0.496
19051541	19051541	100	2	0.009	0.009	0.009
2.11E+09	2.11E+09	100	3	0.982	0.982	0.982
2.02E+09	200	200	4	0.940	0.940	0.940
3.44E+08	2.13E+09	200	5	0.160	0.993	0.160
1.99E+09	38103082	100	6	0.928	0.018	0.928
6.4E+08	2.07E+09	100	7	0.298	0.965	0.298
100	1.89E+09	100	8	0.169	0.880	0.169
1.07E+09	6.88E+08	100	9	0.496	0.321	0.979
19051541	1.84E+09	100	10	0.009	0.855	0.657



seed1	seed2	seed3	i	x1	x2	x3
1.07E+09	1.07E+09	100	1	0.496	0.496	0.496
19051541	19051541	100	2	0.009	0.009	0.009
2.11E+09	2.11E+09		3	0.982	0.982	0.982
2.02E+09	20					
3.44E+08	2.13E+0					
1.99E+09	3810308					
6.4E+08	2.07E+0					
100	1.89E+0					
1.07E+09	6.88E+0					
19051541	1.84E+0					

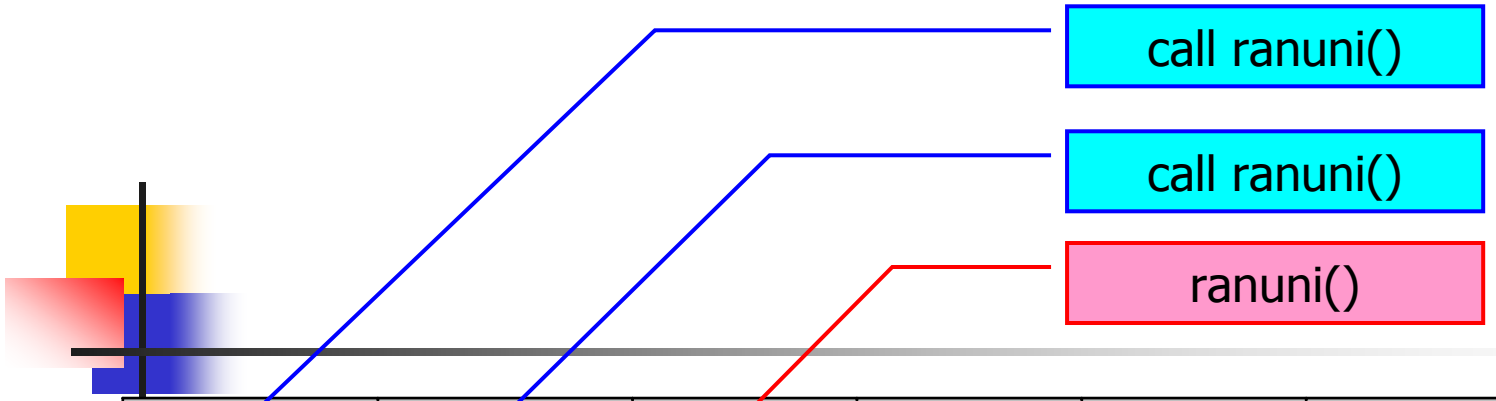
★ seed1 と seed2 は 100 ではない・・・

⇒ シード列の2個目，3個目，・・・ が表示されている

⇒ seed1 と seed2 は，それぞれ x1 と x2 の乱数の元となる値が表示されている

i≦4：x1～x3は同じ乱数

seed1	seed2	seed3	i	x1	x2	x3
1.07E+09	1.07E+09	100	1	0.496	0.496	0.496
19051541	19051541	100	2	0.009	0.009	0.009
2.11E+09	2.11E+09	100	3	0.982	0.982	0.982
2.02E+09	200	200	4	0.940	0.940	0.940
3.44E+08	2.13E+09	200	5	0.160	0.993	0.160
1.99E+09	38103082	100	6	0.928	0.018	0.928
6.4E+08	2.07E+09	100	7	0.298	0.965	0.298
100	1.89E+09	100	8	0.169	0.880	0.169
1.07E+09	6.88E+08	100	9	0.496	0.321	0.979
19051541	1.84E+09	100	10	0.009	0.855	0.657

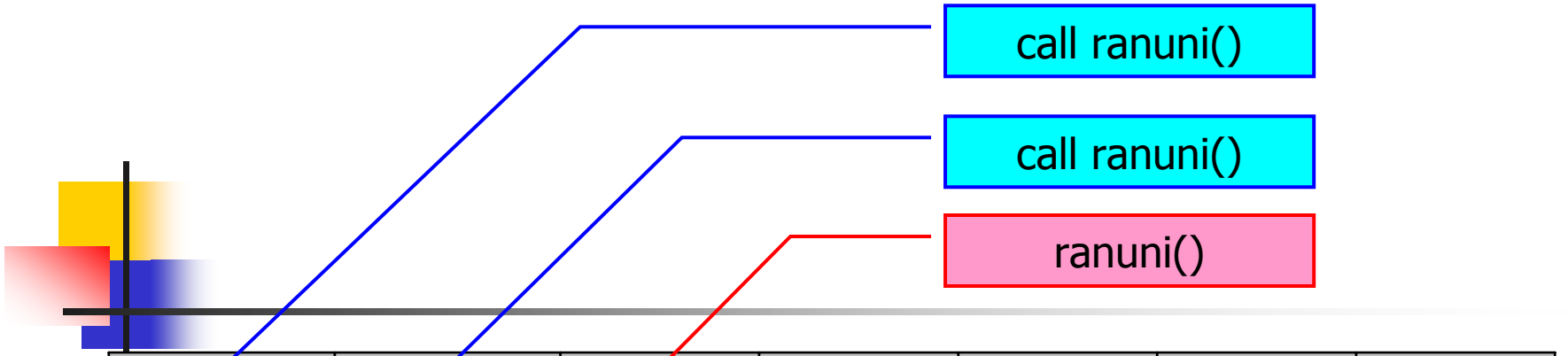


seed1	seed2	seed3	i	x1	x2	x3
1.07E+09	1.07E+09	100	1	0.496	0.496	0.496
19051541	19051541	100	2	0.009	0.009	0.009
2.11E+09	2.11E+09	100	3	0.982	0.982	0.982
2.02E+09	[illegible]200	200	4	0.940	0.940	0.940

★ seed1 と seed2 はそれぞれ，x1と x2 の
$2^{31}-1$ 倍の値が入っている
⇒ シード値列の初期値が100であるだけ
⇒ seed1 と seed2 は別々のシード値列の初期値

seed2とseed3の値を変更する

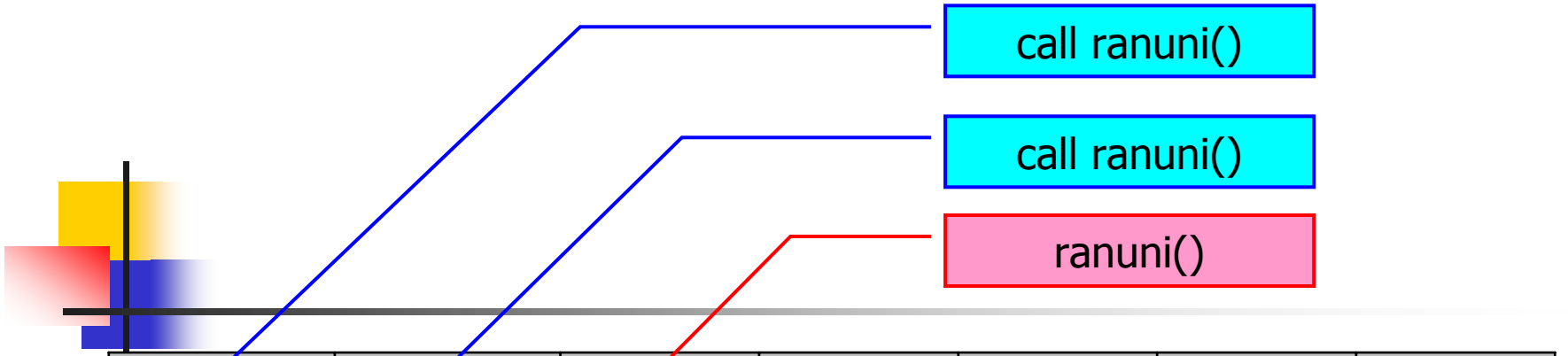
seed1	seed2	seed3	i	x1	x2	x3
1.07E+09	1.07E+09	100	1	0.496	0.496	0.496
19051541	19051541	100	2	0.009	0.009	0.009
2.11E+09	2.11E+09	100	3	0.982	0.982	0.982
2.02E+09	200	200	4	0.940	0.940	0.940
3.44E+08	2.13E+09	200	5	0.160	0.993	0.160
1.99E+09	38103082	100	6	0.928	0.018	0.928
6.4E+08	2.07E+09	100	7	0.298	0.965	0.298
100	1.89E+09	100	8	0.169	0.880	0.169
1.07E+09	6.88E+08	100	9	0.496	0.321	0.979
19051541	1.84E+09	100	10	0.009	0.855	0.657



seed1	seed2	seed3	i	x1	x2	x3
1.07E+09	1.07E+09	100	1	0.496	0.496	0.496
19051541	19051541	100	2	0.009	0.009	0.009
2.11E+09	2.11E+09	100	3	0.982	0.982	0.982
2.02E+09	200	200	4	0.940	0.940	0.940
			5	0.160	0.993	0.160
				0.928	0.018	0.928
				0.298	0.965	0.298
			8	0.169	0.880	0.169
			9	0.496	0.321	0.979
			0	0.009	0.855	0.657

★ seed2 は call ranuni で 呼び出されているので，シード列が初期化されている（初期値200）

★ seed3 は単なる ranuni で呼び出されているので，シード列はそのまま



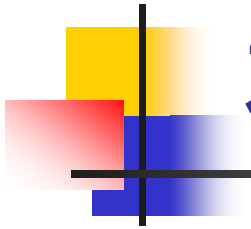
seed1	seed2	seed3	i	x1	x2	x3
1.07E+09	1.07E+09	100	1	0.496	0.496	0.496
19051541	19051541	100	2	0.009	0.009	0.009
2.11E+09	2.11E+09	100	3	0.982	0.982	0.982
2.02E+09	200	200	4	0.940	0.940	0.940
3.44E+08	2.13E+09	200	5	0.160	0.993	0.160
1.99E+09	38103082	100	6	0.928	0.018	0.928
			7	0.298	0.965	0.298
			8	0.169	0.880	0.169
			9	0.496	0.321	0.979
			10	0.009	0.855	0.657

★ その証拠に，seed3 の値を変更しても x1 と x3 の値は同じ（seed1は初期値が100のシード列）

seed1の値を100にすると・・・

seed1	seed2	seed3	i	x1	x2	x3
1.07E+09	1.07E+09	100	1	0.496	0.496	0.496
19051541	19051541	100	2	0.009	0.009	0.009
2.11E+09	2.11E+09	100	3	0.982	0.982	0.982
2.02E+09	200	200	4	0.940	0.940	0.940
3.44E+08	2.13E+					0.160
1.99E+09	38					0.928
6.4E+08	2.(					0.298
100	1.89E+					0.169
1.07E+09	6.88E+08	100	9	0.496	0.321	0.979
19051541	1.84E+09	100	10	0.009	0.855	0.657

★ seed1 は call ranuni で 呼び出されているので，シード列が初期化されている（初期値100）



本日のメニュー

- seed 列の仕様(その説明)
- ranuni と call ranuni の違い
- **seed 列の仕様(その理由)**
- rand 関数について
- 例

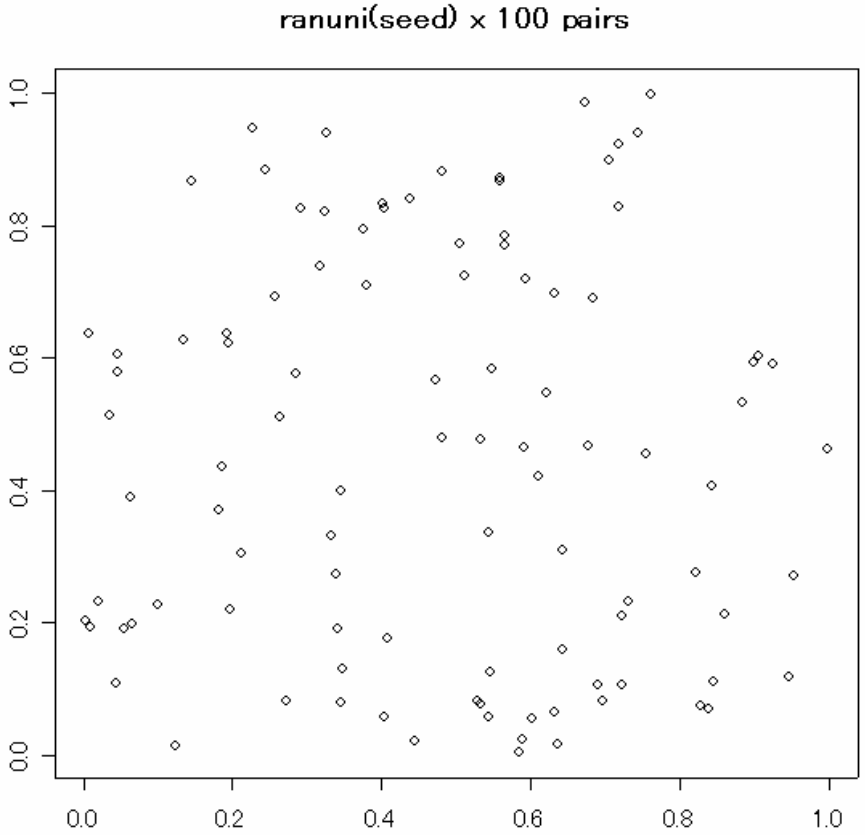
再び SAS のマニュアルより

- call ranuni ルーチンは，**乗算合同法**を使って区間(0,1)上の一様乱数を与える
- SASの乗算合同法：n ≧ 0 のとき，以下の数列からシード列を得る（X_0 はシード列の初期値）.

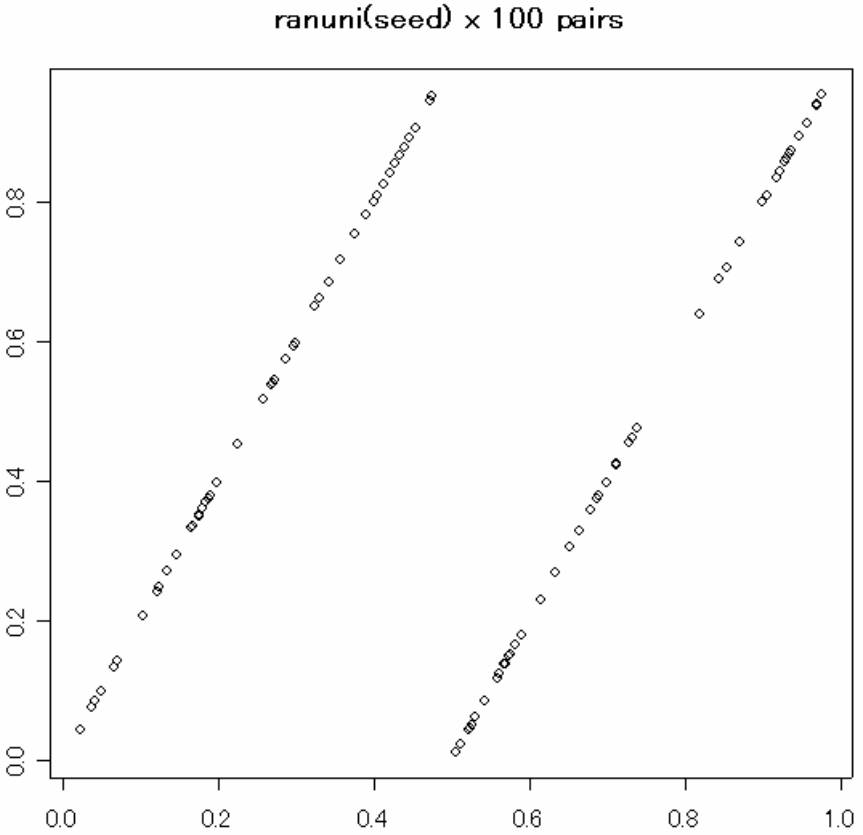
$$X_{n+1} = 397204094\, X_n \quad (\mathrm{mod}\ \ 2^{31}-1)$$

⇒ 非常に原始的な方法

- 「1つのDATAステップ中に複数の ranuni 関数があっても，**シード値列は1つのみ**」にしているのはこの理由のため??

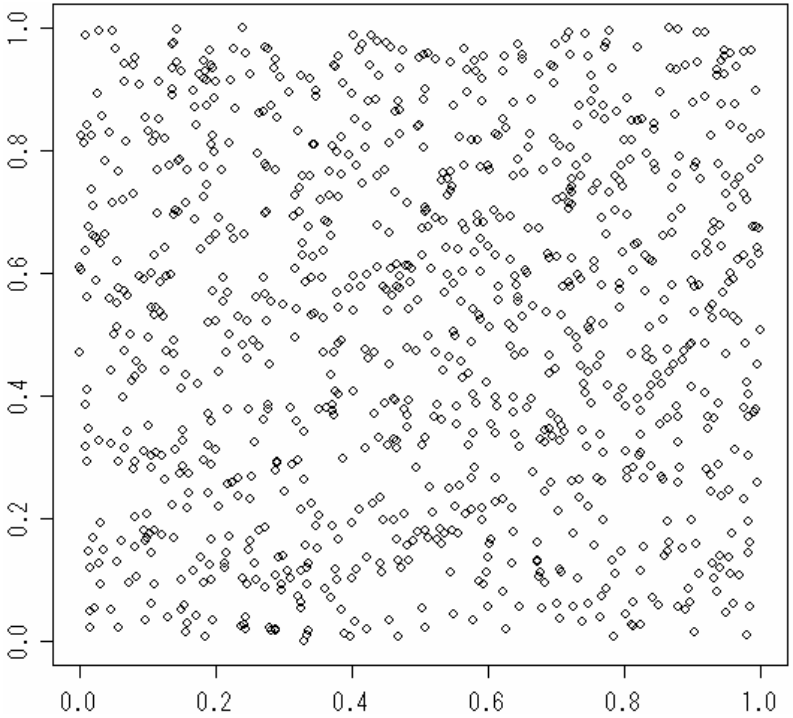


シード列(初期値100)の2個目，4個目，… をx軸データ，3個目，5個目，… をy軸データとして，データ100組をプロット

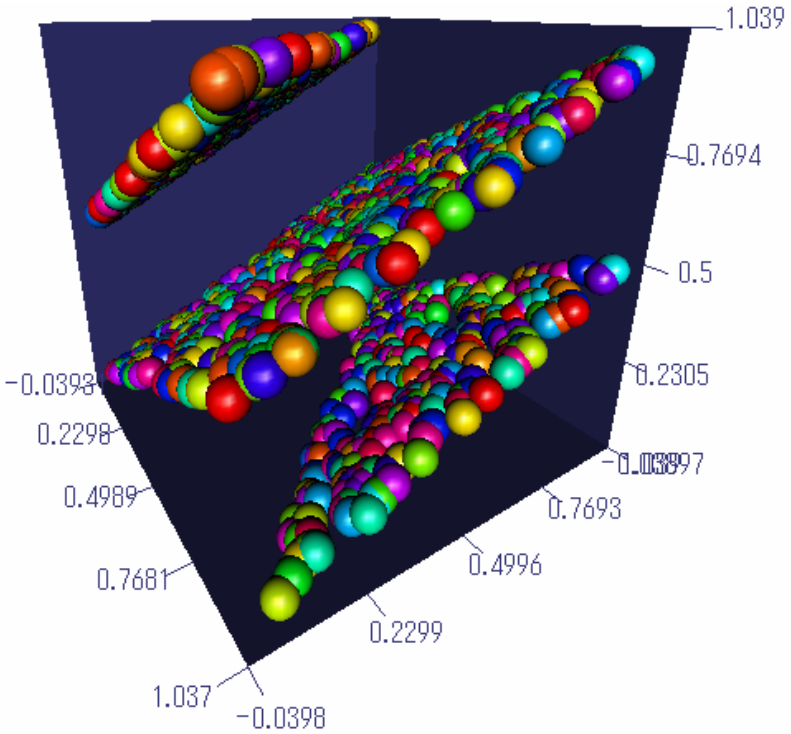


シード列(初期値100)の2個目以降をx軸データ，シード列(初期値200)の2個目以降をy軸データとして，データ100組をプロット

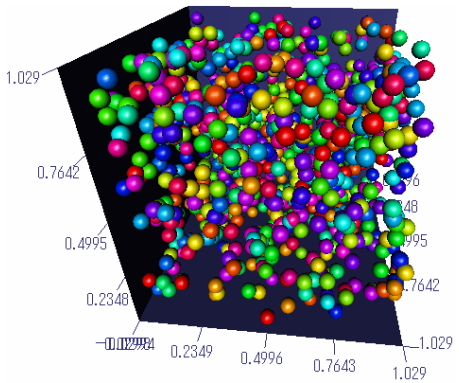
- 2次元以上の組み合わせデータを作る際，異なるシード列の乱数を組み合わせると線形性が生じる！



シード列(初期値100)の2個目以降をx軸データ，
シード列(初期値201)の2個目以降をy軸データ，
として，ranuni関数でデータ1000組を生成

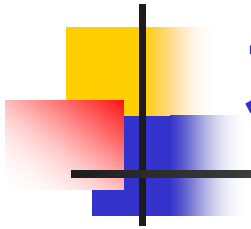


シード列(初期値100)の2個目以降をx軸データ，
シード列(初期値201)の2個目以降をy軸データ，
シード列(初期値302)の2個目以降をz軸データ
として，ranuni関数でデータ1000組を生成



シード列(初期値100)だけで
（x軸，y軸，z軸）データを
1000組生成した場合

- シードの最大公約数が小さい場合でも、乱数を組み合わせると線形性が生じる場合がある。



本日のメニュー

- seed 列の仕様(その説明)
- ranuni と call ranuni の違い
- seed 列の仕様(その理由)
- **rand 関数について**
- 例

rand関数

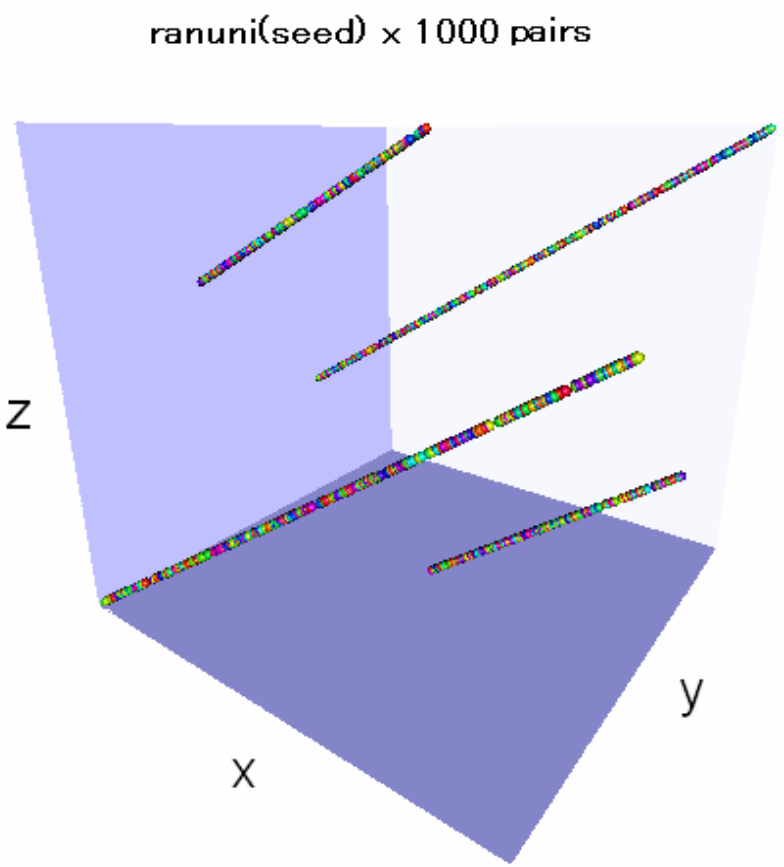
- SAS 9 で正式版となった乱数生成関数
- アルゴリズムは「メルセンヌ・ツイスター」

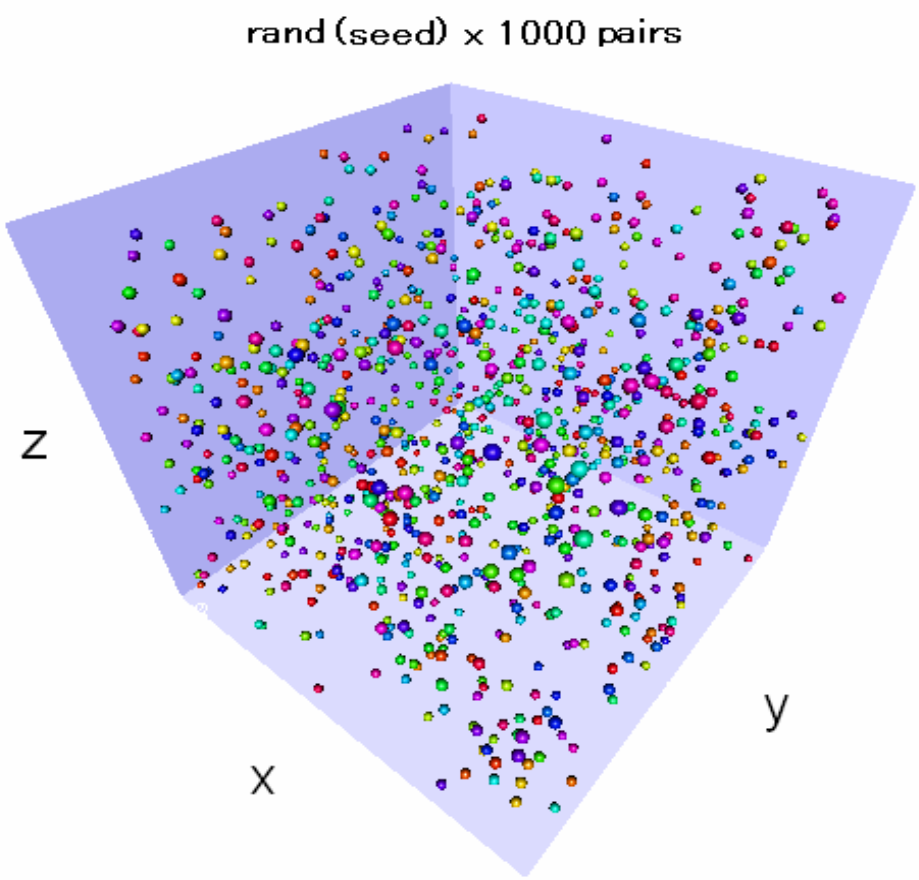
⇒ 乱数の周期は（数学的には）$2^{19937}-1$

⇒ 乱数列は623元超立方体の中で均等に分布する

⇒ その上，乱数生成速度が速い

- rand(“分布名”)で，有名な分布の乱数がすぐに得られる





シード列(初期値100)の2個目以降をx軸データ，
シード列(初期値200)の2個目以降をy軸データ，
シード列(初期値300)の2個目以降をz軸データ
として，ranuni関数でデータ1000組を生成

シード列(初期値100)の2個目以降をx軸データ，
シード列(初期値200)の2個目以降をy軸データ，
シード列(初期値300)の2個目以降をz軸データ
として，rand関数でデータ1000組を生成

- rand 関数なら線形性が生じない！！

rand()で生成可能な乱数の分布

分布名	第1引数	既存の関数
ベルヌーイ分布	Bernoulli	
ベータ分布	Beta	
二項分布	Binomial	ranbin()
コーシー分布	Cauchy	rancau()
χ2分布	Chisquare	
アーラン分布	Erlang	
指数分布	Exponential	ranexp()
F分布	F	
ガンマ分布	Gamma	rangam()
幾何分布	Geometric	
超幾何分布	Hypergeometric	
対数正規分布	Lognormal	
負の二項分布	Negbinomial	
正規分布	Normal	rannor(), normal()
ポアソン分布	Poisson	ranpoi()
t分布	T	
テーブル分布	Table	rantbl()
三角分布	Triangle	rantri()
一様分布	Uniform	ranuni(), uniform()
ワイブル分布	Weibull	

本日のメニュー

- seed 列の仕様(その説明)
- ranuni と call ranuni の違い
- **seed 列の仕様(その理由)**
- rand 関数について
- 7つの例

【例1】1種類のseedで乱数を生成

```
data data1(keep=x1) ;
  retain seed1 100;
  do i=1 to 10;
    x1 = ranuni(seed1) ;
    output ;
  end ;
run ;

data data2(keep=x2 x3) ;
  retain seed1 100;
  do i=1 to 5;
    x2 = ranuni(seed1) ;
    x3 = ranuni(seed1) ;
    output ;
  end ;
run ;
```

x1
0.496257
0.008872
0.982431
0.939865
0.160258
0.927735
0.297917
0.169172
0.979403
0.656655

x2	x3
0.496257	0.008872
0.982431	0.939865
0.160258	0.927735
0.297917	0.169172
0.979403	0.656655

call しない場合は
「シード列は1種類」
なので，seed が同じでも
x2 と x3 は異なる乱数列
が生成される

【例2】1種類のseedで乱数を生成

```
data data1(keep=x1) ;
  retain seed1 100;
  do i=1 to 10;
    call ranuni(seed1, x1) ;
    output ;
  end ;
run ;

data data2(keep=x2 x3) ;
  retain seed1 100;
  do i=1 to 5;
    call ranuni(seed1, x2) ;
    call ranuni(seed1, x3) ;
    output ;
  end ;
run ;
```

x1
0.496257
0.008872
0.982431
0.939865
0.160258
0.927735
0.297917
0.169172
0.979403
0.656655

x2	x3
0.496257	0.008872
0.982431	0.939865
0.160258	0.927735
0.297917	0.169172
0.979403	0.656655

call した場合でも，同じ seed から生成されれば「シード列は1種類」なので，x2 と x3 は異なる乱数列が生成される

【例3】2種類のseedで乱数を生成

```
data data1(keep=x1) ;
  retain seed1 100;
  do i=1 to 10;
    x1 = ranuni(seed1) ;
    output ;
  end ;
run ;

data data2(keep=x2 x3) ;
  retain seed1 seed2 100;
  do i=1 to 5;
    x2 = ranuni(seed1) ;
    x3 = ranuni(seed2) ;
    output ;
  end ;
run ;
```

x1
0.496257
0.008872
0.982431
0.939865
0.160258
0.927735
0.297917
0.169172
0.979403
0.656655

x2	x3
0.496257	0.008872
0.982431	0.939865
0.160258	0.927735
0.297917	0.169172
0.979403	0.656655

call しない場合は
「シード列は1種類」
なので，seed が同じでも
x2 と x3 は異なる乱数列
が生成される

【例4】2種類のseedで乱数を生成

```
data data1(keep=x1) ;
  retain seed1 100;
  do i=1 to 10;
    call ranuni(seed1, x1) ;
    output ;
  end ;
run ;

data data2(keep=x2 x3) ;
  retain seed1 seed2 100;
  do i=1 to 5;
    call ranuni(seed1, x2) ;
    call ranuni(seed2, x3) ;
    output ;
  end ;
run ;
```

x1
0.496257
0.008872
0.982431
0.939865
0.160258
0.927735
0.297917
0.169172
0.979403
0.656655

x2	x3
0.496257	0.496257
0.008872	0.008872
0.982431	0.982431
0.939865	0.939865
0.160258	0.160258

call した場合で異なる seed から生成される場合は，「シード列は別々に生成」されるので，x2 と x3 は独立に乱数列が生成される

【例5】rand関数で乱数を生成

```
data data1(keep=x1) ;
  call streaminit(100) ;
  do I=1 to 10 ;
    x1=rand('uniform') ;
    output ;
  end ;
run ;

data data2(keep=x2 x3) ;
  call streaminit(100) ;
  do I=1 to 5 ;
    x2=rand('uniform') ;
    x3=rand('uniform') ;
    output ;
  end ;
run ;
```

x1
0.928479937
0.729600702
0.180460961
0.559488638
0.104863896
0.216393732
0.215407252
0.379288279
0.629759701
0.344815317

x2	x3
0.928479937	0.729600702
0.180460961	0.559488638
0.104863896	0.216393732
0.215407252	0.379288279
0.629759701	0.344815317

同じ seed から生成されるので「シード列は1種類」．よって，x2 と x3 は異なる乱数列が生成される

【例6】途中で seed を設定しなおすと･･･

```
data data2(keep=x2 x3) ;
  call streaminit(100) ;
  do I=1 to 5 ;
    x2=rand('uniform') ;
    output ;
  end ;
  call streaminit(100) ;
  do I=1 to 5 ;
    x3=rand('uniform') ;
    output ;
  end ;
run ;
```

⇒ シード列は（初期値が100の）
同じ列が使われる

x2	x3
0.928479937	.
0.729600702	.
0.180460961	.
0.559488638	.
0.104863896	.
0.104863896	0.216393732
0.104863896	0.215407252
0.104863896	0.379288279
0.104863896	0.629759701
0.104863896	0.344815317

【例7】途中で seed を設定しなおすと・・・

```
data data1(keep=x1) ;
  call streaminit(200) ;
  do I=1 to 10 ;
    x1=rand('uniform') ;
    output ;
  end ;
run ;

data data2(keep=x2 x3) ;
  call streaminit(100) ;
  do I=1 to 5 ;
    x2=rand('uniform') ;
    output ;
  end ;
  call streaminit(200) ;
  do I=1 to 5 ;
    x3=rand('uniform') ;
    output ;
  end ;
run ;
```

x1	x2	x3
0.937681690	0.928479937	.
0.633528548	0.729600702	.
0.671323263	0.180460961	.
0.023460234	0.559488638	.
0.360697330	0.104863896	.
0.575426974	0.104863896	0.216393732
0.385583326	0.104863896	0.215407252
0.550935807	0.104863896	0.379288279
0.722136058	0.104863896	0.629759701
0.637622831	0.104863896	0.344815317

⇒ seed の変更は効かない！

の場合（余談）

- とうの昔に一様乱数生成関数 runif() に「メルセンヌ・ツイスター」を採用（R 1.7.0 より）

⇒ seed を入れても入れなくても乱数を生成 することが出来る

⇒ SAS のようにシード列と乱数列の違いを知る必要無し

- そもそも，一様乱数生成関数が複数あるのは混乱の元なのでは??

⇒ R のように「生成アルゴリズムを引数で指定」する方が，知ってる人にも知らない人にも親切なのでは??

- ranuni 関数（乗算合同法）から rand 関数（メルセンヌ・ツイスター）に乗り換える人はどれだけ居る??

参考文献・引用文献

- SASランゲージ・リファレンス
  「第11章　RANUNI ルーチン」
- SAS Technical News Winter 2005
  ⇒「rand関数で生成可能な乱数の分布」で引用
- 伏見 正則「乱数」東京大学出版会
- 良い乱数・悪い乱数
  http://www001.upp.so-net.ne.jp/isaku/rand.html

終