

bbat24

This module contains predefined batteries of statistical tests for sequences of uniform random numbers in the interval $[0, 1)$ with at most 24 bits of resolution. To test a RNG for general use, one could first apply the small and fast battery `SmallBird`. If it passes, one could then apply the more stringent battery `Bird`.

The batteries described in this module will write the results of each test (on standard output) with a standard level of details (assuming that the boolean switches of module `swrite` have their default values), followed by a summary report of the suspect p -values obtained from the specific tests included in the batteries. It is also possible to get only the summary report in the output, with no detailed output from the tests, by setting the boolean switch `swrite_Basic` to `FALSE`.

```
#include "unif01.h"
```

```
extern int bbat24_NTests;
```

The maximum number of p -values in the array `bbat24_pVal`. For small sample size, some of the tests in the battery may not be done. Furthermore, some of the tests computes more than one statistic and its p -value, so `bbat24_NTests` will usually be larger than the number of tests in the battery.

```
extern double bbat24_pVal[];
```

This array keeps the p -values resulting from the battery of tests that is currently applied (or the last one that has been called). It is used by any battery in this module. The p -value of the j -th test in the battery is kept in `bbat24_pVal[j - 1]`, for $1 \leq j \leq \text{bbat24_NTests}$.

```
extern char *bbat24_TestNames[];
```

This array keeps the names of each test from the battery that is currently applied (or the last one that has been called). It is used by any battery in this module. The name of the j -th test in the battery is kept in `bbat24_TestNames[j - 1]`, for $1 \leq j \leq \text{bbat24_NTests}$.

The batteries of tests

```
void bbat24_SmallBird (unif01_Gen *gen);  
void bbat24_SmallBirdFile (char *filename);
```

Both functions applies `SmallBird`, a small and fast battery of tests, to a RNG. The function `bbat24_SmallBirdFile` applies `SmallBird` to a RNG given as a text file of floating-point numbers in $[0, 1)$. The file will be rewound to the beginning before each test. Thus `bbat24_SmallBird` applies the tests on one unbroken stream of successive numbers, while `bbat24_SmallBirdFile` applies each test on the same sequence of numbers. No test requires that `gen` returns more than 24 bits of resolution.

The following tests are applied:

1. `smarsa_BirthdaySpacings` with $N = 1$, $n = 5 * 10^6$, $r = 0$, $d = 2^{21}$, $t = 3$, $p = 1$.
2. `sknuth_Collision` with $N = 1$, $n = 5 * 10^6$, $r = 0$, $d = 2^{16}$, $t = 2$.
3. `sknuth_Gap` with $N = 1$, $n = 2 * 10^5$, $r = 16$, $\text{Alpha} = 0$, $\text{Beta} = 1/256$.
4. `sknuth_SimpPoker` with $N = 1$, $n = 4 * 10^5$, $r = 18$, $d = 64$, $k = 64$.
5. `sknuth_CouponCollector` with $N = 1$, $n = 5 * 10^5$, $r = 0$, $d = 16$.
6. `sknuth_MaxOft` with $N = 1$, $n = 2 * 10^6$, $r = 0$, $d = 10^5$, $t = 6$.
7. `svaria_WeightDistrib` with $N = 1$, $n = 2 * 10^5$, $r = 21$, $k = 256$, $\text{Alpha} = 0$, $\text{Beta} = 1/8$.
8. `smarsa_MatrixRank` with $N = 1$, $n = 50000$, $r = 0$, $s = 24$, $L = k = 48$.
9. `sstring_HammingIndep` with $N = 1$, $n = 5 * 10^5$, $r = 0$, $s = 24$, $L = 240$, $d = 0$.
10. `swalk_RandomWalk1` with $N = 1$, $n = 10^6$, $r = 0$, $s = 24$, $L_0 = 120$, $L_1 = 120$.

```
void bbat24_RepeatSmallBird (unif01_Gen *gen, int rep[]);
```

This function applies specific tests of `SmallBird` on generator `gen`. Test numbered i in the enumeration above will be applied `rep[i]` times successively on `gen`. Those tests with `rep[i] = 0` will not be applied. This is useful when a test in `SmallBird` had a suspect p -value, and one wants to reapply the test a few more times to find out whether the generator failed the test or whether the suspect p -value was a statistical fluke. Restriction: Array `rep` must have one more element than the number of tests in `SmallBird`.

```
void bbat24_Bird (unif01_Gen *gen);
```

Applies the battery `Bird`, a suite of stringent statistical tests, to the 24-bit generator `gen`. The battery includes the classical tests described in Knuth [63] as well as many other tests. None of the tests assume that `gen` returns more than 24 bits of resolution. On a PC with an AMD Athlon 64 Processor 4000+ of clock speed 2400 MHz running with Red Hat Linux, `Bird` will require around ?? 1 hour of CPU time. `Bird` uses approximately 2^{35} ?? random numbers.

The following tests are applied:

1. `smarsa_SerialOver` with $N = 1$, $n = 5 * 10^8$, $r = 0$, $d = 2^{12}$, $t = 2$.
2. `smarsa_SerialOver` with $N = 1$, $n = 3 * 10^8$, $r = 0$, $d = 2^6$, $t = 4$.
3. `smarsa_CollisionOver` with $N = 10$, $n = 10^7$, $r = 0$, $d = 2^{20}$, $t = 2$.
4. `smarsa_CollisionOver` with $N = 10$, $n = 10^7$, $r = 4$, $d = 2^{20}$, $t = 2$.

5. smarsa_CollisionOver with $N = 10$, $n = 10^7$, $r = 0$, $d = 2^{10}$, $t = 4$.
6. smarsa_CollisionOver with $N = 10$, $n = 10^7$, $r = 14$, $d = 2^{10}$, $t = 4$.
7. smarsa_CollisionOver with $N = 10$, $n = 10^7$, $r = 0$, $d = 32$, $t = 8$.
8. smarsa_CollisionOver with $N = 10$, $n = 10^7$, $r = 19$, $d = 32$, $t = 8$.
9. smarsa_CollisionOver with $N = 10$, $n = 10^7$, $r = 0$, $d = 4$, $t = 20$.
10. smarsa_CollisionOver with $N = 10$, $n = 10^7$, $r = 22$, $d = 4$, $t = 20$.
11. smarsa_BirthdaySpacings with $N = 5$, $n = 2 * 10^7$, $r = 0$, $d = 2^{21}$, $t = 3$, $p = 1$.
12. smarsa_BirthdaySpacings with $N = 5$, $n = 2 * 10^7$, $r = 0$, $d = 2^{16}$, $t = 4$, $p = 1$.
13. smarsa_BirthdaySpacings with $N = 3$, $n = 2 * 10^7$, $r = 0$, $d = 2^9$, $t = 7$, $p = 1$.
14. smarsa_BirthdaySpacings with $N = 3$, $n = 2 * 10^7$, $r = 7$, $d = 2^9$, $t = 7$, $p = 1$.
15. smarsa_BirthdaySpacings with $N = 3$, $n = 2 * 10^7$, $r = 14$, $d = 2^8$, $t = 8$, $p = 1$.
16. snpair_ClosePairs with $N = 200$, $n = 50000$, $r = 0$, $t = 2$, $p = 0$, $m = 4$.
17. snpair_ClosePairs with $N = 10$, $n = 10^6$, $r = 0$, $t = 3$, $p = 0$, $m = 20$.
18. snpair_ClosePairs with $N = 10$, $n = 10^6$, $r = 0$, $t = 4$, $p = 0$, $m = 30$.
19. snpair_ClosePairs with $N = 5$, $n = 10^6$, $r = 0$, $t = 7$, $p = 0$, $m = 30$.
20. snpair_ClosePairsBitMatch with $N = 4$, $n = 4 * 10^6$, $r = 0$, $t = 2$.
21. snpair_ClosePairsBitMatch with $N = 2$, $n = 4 * 10^6$, $r = 0$, $t = 4$.
22. sknuth_SimpPoker with $N = 1$, $n = 4 * 10^7$, $r = 0$, $d = 16$, $k = 16$.
23. sknuth_SimpPoker with $N = 1$, $n = 4 * 10^7$, $r = 20$, $d = 16$, $k = 16$.
24. sknuth_SimpPoker with $N = 1$, $n = 10^7$, $r = 0$, $d = 64$, $k = 64$.
25. sknuth_SimpPoker with $N = 1$, $n = 10^7$, $r = 18$, $d = 64$, $k = 64$.
26. sknuth_CouponCollector with $N = 1$, $n = 4 * 10^7$, $r = 0$, $d = 4$.
27. sknuth_CouponCollector with $N = 1$, $n = 4 * 10^7$, $r = 22$, $d = 4$.
28. sknuth_CouponCollector with $N = 1$, $n = 10^7$, $r = 0$, $d = 16$.
29. sknuth_CouponCollector with $N = 1$, $n = 10^7$, $r = 20$, $d = 16$.
30. sknuth_Gap with $N = 1$, $n = 10^8$, $r = 0$, Alpha = 0, Beta = 1/8.
31. sknuth_Gap with $N = 1$, $n = 10^8$, $r = 20$, Alpha = 0, Beta = 1/8.

32. `sknuth_Gap` with $N = 1$, $n = 5 * 10^6$, $r = 0$, $\text{Alpha} = 0$, $\text{Beta} = 1/256$.
33. `sknuth_Gap` with $N = 1$, $n = 5 * 10^6$, $r = 16$, $\text{Alpha} = 0$, $\text{Beta} = 1/256$.
34. `sknuth_Run` with $N = 1$, $n = 5 * 10^8$, $r = 0$, $\text{Up} = \text{TRUE}$.
35. `sknuth_Run` with $N = 1$, $n = 5 * 10^8$, $r = 10$, $\text{Up} = \text{FALSE}$.
36. `sknuth_Permutation` with $N = 1$, $n = 5 * 10^7$, $r = 0$, $t = 10$.
37. `sknuth_Permutation` with $N = 1$, $n = 5 * 10^7$, $r = 10$, $t = 10$.
38. `sknuth_CollisionPermut` with $N = 5$, $n = 10^7$, $r = 0$, $t = 13$.
39. `sknuth_CollisionPermut` with $N = 5$, $n = 10^7$, $r = 10$, $t = 13$.
40. `sknuth_MaxOft` with $N = 50$, $n = 500000$, $r = 0$, $d = 10000$, $t = 5$.
41. `sknuth_MaxOft` with $N = 40$, $n = 500000$, $r = 0$, $d = 10000$, $t = 10$.
42. `sknuth_MaxOft` with $N = 30$, $n = 500000$, $r = 0$, $d = 10000$, $t = 20$.
43. `sknuth_MaxOft` with $N = 30$, $n = 500000$, $r = 0$, $d = 10000$, $t = 30$.
44. `svaria_SampleProd` with $N = 1$, $n = 10^7$, $r = 0$, $t = 10$.
45. `svaria_SampleProd` with $N = 1$, $n = 10^7$, $r = 0$, $t = 30$.
46. `svaria_SampleMean` with $N = 10^7$, $n = 20$, $r = 0$.
47. `svaria_SampleCorr` with $N = 1$, $n = 5 * 10^8$, $r = 0$, $k = 1$.
48. `svaria_AppearanceSpacings` with $N = 1$, $Q = 10^7$, $K = 10^8$, $r = 0$, $s = 24$, $L = 12$.
49. `svaria_AppearanceSpacings` with $N = 1$, $Q = 10^7$, $K = 10^8$, $r = 12$, $s = 12$, $L = 12$.
50. `svaria_WeightDistrib` with $N = 1$, $n = 2 * 10^6$, $r = 0$, $k = 256$, $\text{Alpha} = 0$, $\text{Beta} = 1/8$.
51. `svaria_WeightDistrib` with $N = 1$, $n = 2 * 10^6$, $r = 8$, $k = 256$, $\text{Alpha} = 0$, $\text{Beta} = 1/8$.
52. `svaria_WeightDistrib` with $N = 1$, $n = 2 * 10^6$, $r = 16$, $k = 256$, $\text{Alpha} = 0$, $\text{Beta} = 1/8$.
53. `svaria_WeightDistrib` with $N = 1$, $n = 2 * 10^6$, $r = 20$, $k = 256$, $\text{Alpha} = 0$, $\text{Beta} = 1/8$.
54. `svaria_SumCollector` with $N = 1$, $n = 2 * 10^7$, $r = 0$, $g = 10$.
55. `smarsa_MatrixRank` with $N = 1$, $n = 10^6$, $r = 0$, $s = 24$, $L = k = 48$.

56. smarsa_MatrixRank with $N = 1, n = 10^6, r = 16, s = 8, L = k = 48$.
57. smarsa_MatrixRank with $N = 1, n = 50000, r = 0, s = 24, L = k = 240$.
58. smarsa_MatrixRank with $N = 1, n = 50000, r = 16, s = 8, L = k = 240$.
59. smarsa_MatrixRank with $N = 1, n = 2000, r = 0, s = 24, L = k = 960$.
60. smarsa_MatrixRank with $N = 1, n = 2000, r = 16, s = 8, L = k = 960$.
61. smarsa_Savir2 with $N = 1, n = 2 * 10^7, r = 0, m = 2^{20}, t = 30$.
62. smarsa_GCD with $N = 1, n = 10^8, r = 0, s = 24$.
63. swalk_RandomWalk1 with $N = 1, n = 5 * 10^7, r = 0, s = 24, L_0 = L_1 = 72$.
64. swalk_RandomWalk1 with $N = 1, n = 10^7, r = 16, s = 8, L_0 = L_1 = 72$.
65. swalk_RandomWalk1 with $N = 1, n = 5 * 10^6, r = 0, s = 24, L_0 = L_1 = 600$.
66. swalk_RandomWalk1 with $N = 1, n = 10^6, r = 16, s = 8, L_0 = L_1 = 600$.
67. swalk_RandomWalk1 with $N = 1, n = 5 * 10^5, r = 0, s = 24, L_0 = L_1 = 6000$.
68. swalk_RandomWalk1 with $N = 1, n = 10^5, r = 16, s = 8, L_0 = L_1 = 6000$.
69. scomp_LinearComp with $N = 1, n = 120000, r = 0, s = 1$.
70. scomp_LinearComp with $N = 1, n = 120000, r = 23, s = 1$.
71. scomp_LempelZiv with $N = 10, k = 25, r = 0, s = 24$.
72. sspectral_Fourier3 with $N = 50000, k = 14, r = 0, s = 24$.
73. sspectral_Fourier3 with $N = 50000, k = 14, r = 16, s = 8$.
74. sstring_LongestHeadRun with $N = 1, n = 1000, r = 0, s = 24, L = 10^7$.
75. sstring_LongestHeadRun with $N = 1, n = 300, r = 16, s = 8, L = 10^7$.
76. sstring_PeriodsInStrings with $N = 1, n = 3 * 10^8, r = 0, s = 24$.
77. sstring_PeriodsInStrings with $N = 1, n = 3 * 10^8, r = 5, s = 15$.
78. sstring_HammingWeight2 with $N = 100, n = 10^8, r = 0, s = 24, L = 10^6$.
79. sstring_HammingWeight2 with $N = 30, n = 10^8, r = 16, s = 8, L = 10^6$.
80. sstring_HammingCorr with $N = 1, n = 5 * 10^8, r = 0, s = 24, L = 24$.
81. sstring_HammingCorr with $N = 1, n = 5 * 10^7, r = 0, s = 24, L = 240$.
82. sstring_HammingCorr with $N = 1, n = 10^7, r = 0, s = 24, L = 1200$.

83. `sstring_HammingIndep` with $N = 1, n = 3 * 10^8, r = 0, s = 24, L = 24, d = 0$.
84. `sstring_HammingIndep` with $N = 1, n = 10^8, r = 16, s = 8, L = 24, d = 0$.
85. `sstring_HammingIndep` with $N = 1, n = 3 * 10^7, r = 0, s = 24, L = 240, d = 0$.
86. `sstring_HammingIndep` with $N = 1, n = 10^7, r = 16, s = 8, L = 240, d = 0$.
87. `sstring_HammingIndep` with $N = 1, n = 10^7, r = 0, s = 24, L = 1200, d = 0$.
88. `sstring_HammingIndep` with $N = 1, n = 10^6, r = 16, s = 8, L = 1200, d = 0$.
89. `sstring_Run` with $N = 1, n = 10^9, r = 0, s = 24$.
90. `sstring_Run` with $N = 1, n = 10^9, r = 16, s = 8$.
91. `sstring_AutoCor` with $N = 10, n = 10^9, r = 0, s = 24, d = 1$.
92. `sstring_AutoCor` with $N = 5, n = 10^9, r = 16, s = 8, d = 1$.
93. `sstring_AutoCor` with $N = 10, n = 10^9, r = 0, s = 24, d = 24$.
94. `sstring_AutoCor` with $N = 5, n = 10^9, r = 16, s = 8, d = 8$.

```
void bbat24_RepeatBird (unif01_Gen *gen, int rep[]);
```

Similar to `bbat24_RepeatSmallBird` above but applied on `Bird`.