Fast randomness testing with reimplementation of NIST STS

User's and developer's manual
Version 5.0.2x based on STS 2.1.2

November 2016

Zdeněk Říha
Marek Sýs

Before you start

Please make sure you use the latest version of the SW. Check the website of the project:

`https://randomness-tests.fi.muni.cz/`

Introduction

This manual describes the use of the program Fast STS, offering a significantly faster reimplementation of the NIST STS tests. The program offers complete backward compatibility with the interface of NIST STS 2.1.2 plus a set of new features.

Requirements

The program is written in C and compiles on most modern UNIX and Windows platforms. A makefile is included, a Visual Studio 2015 solution files is included as well. To compile type 'make' on the command line (in the same directory where the makefile is present). The compilation will result in creating a single binary executable ('assess' in case of Unix systems).

To be able to successfully use the binary program, you will need:

1. Subdirectory 'templates' with template files for the Nonoverlapping template matching test.
2. Subdirectory 'experiments' with a complex directory tree for storing the results. If the complex directory structure does not exist, it can be created using the provided 'create-dir-script' on Unix systems. For windows system either use the folder structure provided or adapt the script.
3. Subdirectory 'data' with test data if you plan to use the selftest feature.

Using the program

The program offers full backward compatibility with the original NIST STS program. The original program requires and allows only a single command line argument. That's the length of the tested sequence in bits.

Example:
`'./assess 1000000'`

Then the rest of the parameters is negotiated in the program textual interface. An example follows:

G E N E R A T O R S E L E C T I O N

[0] Input File	[1] Linear Congruential
[2] Quadratic Congruential I	[3] Quadratic Congruential II
[4] Cubic Congruential	[5] XOR
[6] Modular Exponentiation	[7] Blum-Blum-Shub
[8] Micali-Schnorr	[9] G Using SHA-1

Enter Choice: 0

User Prescribed Input File: /dev/urandom

S T A T I S T I C A L T E S T S

[01] Frequency	[02] Block Frequency
[03] Cumulative Sums	[04] Runs
[05] Longest Run of Ones	[06] Rank
[07] Discrete Fourier Transform	[08] Nonperiodic Template Matchings
[09] Overlapping Template Matchings	[10] Universal Statistical
[11] Approximate Entropy	[12] Random Excursions
[13] Random Excursions Variant	[14] Serial
[15] Linear Complexity	

INSTRUCTIONS

Enter 0 if you DO NOT want to apply all of the statistical tests to each sequence and 1 if you DO.

Enter Choice: 1

P a r a m e t e r A d j u s t m e n t s

[1] Block Frequency Test - block length(M):	128
[2] NonOverlapping Template Test - block length(m):	9
[3] Overlapping Template Test - block length(m):	9
[4] Approximate Entropy Test - block length(m):	10
[5] Serial Test - block length(m):	16
[6] Linear Complexity Test - block length(M):	500

Select Test (0 to continue): 0

How many bitstreams? 1

Input File Format:

[0] ASCII - A sequence of ASCII 0's and 1's
[1] Binary - Each byte in data file contains 8 bits of data

Select input mode: 1
Statistical Testing In Progress.....
Statistical Testing Complete!!!!!!!!!!!!

A detailed description of the program interface is available in the Section 5 User's Guide of the NIST SP 800-22 Revision 1a.
A modification of the original command line arguments is provided to update the functionality of the program. A '-fast' switch on the command line invokes the new reimplemented tests. Please include also the '-fileoutput' switch to obtain the full result files.

Example:
`'./assess -fast -fileoutput 1000000'`

This will provide the original functionality using the fast test functions. The remaining parameters are obtained in the following dialogs and the output files with results are the same as in the original version.

The new functionality

The program also offers new functionality. This added functionality can be divided into 4 categories:

0. The testing is REALLY FAST now!
1. New command line arguments to be able to use the program in scripts without additional manual intervention.
2. Further speed up of the program by smarter result processing.
3. Incorporation of the Kolmogorov-Smirnov (KS) test.
4. Self test.

0. Fast algorithms

Just use the `'-fast'` switch as described above. Then the new significantly faster algorithms are used. The results will be equal to the original ones.

1. New scripting command line options

Note: all options can be used with a single dash (Windows convention) or double dash (Unix convention) e.g. both `'-fast'` and `'--fast '` is ok.
The program takes a set of new command line arguments. These arguments allow to specify parameters that normaly would be entered in the interactive mode. Specifying these parameters on the command line will enable the use of the program for automated testing in scripts. Not all the parameters must be specified on the command line. However, if an important parameter is not specified on the command line it will be asked interactively.

The following parameters may now be used:

The testing

These parameters allow to specify which file will be tested, how many streams will be tested and which tests will be run. The bit length of each tested sequence a standard mandatory parameter that must be present on the command line (no preceding keyword is expected in front of that parameter). If you intend to test one of the built-in random number generators, please do it interactively (no command line options are available).

`--file <Path>`	Test the specified file
`--streams <Number>`	Number of bit steams to test
`--tests <Number>`	Specify tests to run e.g. 111111111111111

The order of the tests in this parameter is the following:

[01] Frequency	[02] Block Frequency
[03] Cumulative Sums	[04] Runs
[05] Longest Run of Ones	[06] Rank
[07] Discrete Fourier Transform	[08] Nonperiodic Template Matchings
[09] Overlapping Template Matchings	[10] Universal Statistical
[11] Approximate Entropy	[12] Random Excursions
[13] Random Excursions Variant	[14] Serial
[15] Linear Complexity	

The format of the file

The format of the input file to be analyzed is either binary (all bits are used; unaligned bits between streams are discarded) or ASCII (ASCII digits of '0' and '1').

`--ascii`	Input file is a text file (with 0s and 1s)
`--binary`	Input file is a binary file

The test parameters

Some statistical tests of randomness require additional parameters. The parameters can be either specified on the command line, entered in the interactive session or default parameters are used. You can also explicitly specify the default parameters are to be used. The default parameters are:

Block Frequency Test - block length(M):	128
NonOverlapping Template Test - block length(m):	9
Overlapping Template Test - block length(m):	9
Approximate Entropy Test - block length(m):	10
Serial Test - block length(m):	16
Linear Complexity Test - block length(M):	500

The following test parameters can be specified:

`--blockfreqpar <Number>`	Test parameter (block frequency)
`--nonoverpar <Number>`	Test parameter (nonoverlapping)
`--overpar <Number>`	Test parameter (overlapping)
`--approxpar <Number>`	Test parameter (approximate entropy)
`--serialpar <Number>`	Test parameter (serial)
`--linearpar <Number>`	Test parameter (linear complexity)
`--defaultpar`	Use default test parameters

2. Processing of the results

The traditional way to process the results in NIST STS is to produce textual files for each particular statistical test and then opening the files and processing the results. Writing and reading all the files is relatively slow. To speed up the result processing the program offers a new way result management. In this case the intermediate files with results of particular tests are not created and only the summary file is constructed. All intermediate results are stored in memory only. The typical flow of data testing would first invoke the fast processing and if the summary results indicate a failure, then full results can be obtained and a manual analysis can follow.

The relevant command line options are:

`--fileoutput`	Produce full text outputs
`--onlymem`	Produce simplified output

The default behavior is to use only the simplified output (if KS is defined, see section for developers if needed).

3. Kolmogorov-Smirnov (KS) test

The Kolmogorov Smirnov test of the uniformity of the resulting p-values is executed in addition to the standard ways of result processing. The results of the KS test are added to the results files. (For developers: the KS test is performed if the KS is defined).

4. Selftest

The following new command line option is provided to verify the functionality of the test suite. Test results of fixed previously known data are compared against expected values. The expected results are published in Appendix B of the NIST SP 800-22. (For developers: the selftest is only available if the KS is defined).

`--selftest`	Self-check of the program (check known results)

Examples

```
./assess -selftest

(Perform the selftest of the program, to verify the compilation)

./assess -fast -file data/e.bin -streams 1 -tests 111111111111 -defaultpar
-onlymem -binary 1000000

(Perform the randomness tests of the data/e.bin file, run all the 15 tests
with default parameters, generate only the main result file, use fast test
algoritms, test the first 1 000 000 bits of the binary file)
```

```
./assess -fast -file /dev/urandom -streams 2 -tests 010000000000000
-blockfreqpar 100  -fileoutput -binary 1000000
```

(Run only the Block Frequency test with the parameter of 100, perform the test twice, each with 1 000 000 bits read from the binary file /dev/urandom).

For developers

If the standard functionality is not sufficient for you, then you can play with the source code. The makefile is available in the main directory, header files are located in the 'include' directory and the C source code can be found in the 'src' directory. After changing the source code, issue the command 'make clean' to delete the object files and then use the 'make' command to compile and link the project. The final result is a single binary file 'assess'.
To be able to use the binary you will need the subdirectories 'experiments', 'templates' and 'data' as discussed above.
The program is able to work in 4 modes. The mode can be selected in the config.h file and then the source codes must be completely recompiled. The mode is selected by uncommenting of the one following defines. Note: Exactly one of elements must be uncommented.

```
//#define VERIFY_RESULTS 1
//#define SPEED 1
//#define FILE_OUTPUT 1
#define KS 1
```

KS mode

The standard compilation mode is KS. In this mode the full functionality is offered including the fast algorithms, fast in-memory result processing and the new Kolmogorov Smirnov test. This is the default compilation mode.

FILE_OUTPUT mode

An alternative to the standard KS mode is the FILE_OUTPUT mode. If you select the FILE_OUTPUT mode then the program is performing the randomness testing in a similar way as the KS mode, but the new KS test is not available and the file results produced are exactly the same as in the original version of the NIST STS utility. This allows to use the fast testing algorithms while having exactly the same output files, which is useful e.g. for existing automated parsing of the result files.

VERIFY_RESULTS mode

The KS mode offers a simple test mode, which is calculating resulting p-values for all the tests for a set of preselected sample input files as documented in the Appendix B of the NIST SP 800-22. This is a useful function for compilation verification, but developers finetuning the algorithms need more thorough testing. In the VERIFY_RESULTS mode the program tests whether the results of the new faster algorithms are exactly the same as the results of the original algorithms.
In this very special mode the program generates a set of bit sequences then calculates the results by calling the original and new functions, comparing both the results.

In the VERIFY_RESULTS mode the program expects exactly one argument specifying the number of the statistical test to be tested. A comprehensive set of sequences of different lengths is being generated and tested. Depending on a particular test selected and speed of your machine the tests can take from hours to years to complete. If you want to modify the particularities of the tests performed, update the source code accordingly. Look in the main.c file.

Examples:
`./assess 3`
(This tests the new implementation of the Cumulative Sums randomness test. The tested lengths are being printed. If a test fails, then the symbol '!' is printed.)
`./assess 15`
(This tests the implementation of the Linear Complexity test).

SPEED mode

The SPEED mode has been programmed to measure the speed up of the new code compared to the original NIST code. In this case sample bit streams are generated, the original and new test functions are executed and the time need for the execution of the functions is measured. The execution is repeated several times to get more accurate results. Between the several runs of the tests the CPU cache is flushed.

In the SPEED mode the program expects three parameters:

- The first parameter specified the length of the bit sequence to be tested. Either varying (1) or fixed at 20MB (0).
- The second parameter specified how many times each test should be executed. The final measurement is takes as the minimum of all the executions.
- The second and third parameter specifies the range of statistical tests to be executed. The tests are numbered 1 to 36 and include the 15 statistical tests with different block sizes. For the details of the tests numbering, please see the function speed in main.c

Examples:
`./assess 0 10 1 1`
(This executes the Frequency (#1) test on 20MB of data, measure the speed and presents the time needed to run the test and the speed up achieved. The results are presented in CPU cycles and seconds. The Frequency test is run 10x and the minimum values are presented.)

`./assess 1 5 1 2`
(The Frequency (#1) and Block Frequency (#2) tests are executed 5 times with different bit stream lengths).

In the source codes each test can be found in at least two versions v1, v2 (for instance Frequency_v1 and Frequency_v2), where v1 (Frequency_v1) defines original implementation of the test and v2 (Frequency_v2) defines a new implementation.

You can add new implementation of an arbitrary test as follows:

1. Add the new function (Frequency_new(int n)...) to file frequency.h.
2. Add its prototype (Frequency_new(int n);) to stat-fncs.h
3. Change #define Frequency_v2 Frequency3 to #define Frequency_v2 Frequency_new

Support for unusual parameters

No matter which compilation mode you use this is a way to introduce support for m> 25 (and m<= 32) in Serial and Overlapping Template Matching tests.

If you feel limited with the maximum value of m (m<= 25) in the Serial and Overlapping Template Matching tests then simply replace the calls of get_nth_block4() with the calls of get_nth_block_effect() in the relevant parts of the source code. For the serial test search for function Serial4() in serial.c. For the Overlapping Template Matching test search for the function OverlappingTemplateMatchings2() in overlappingTemplateMatchings.c