STS-2.1.2 and SP800-90B Assessment Suite Anomalous results.

2017/5/23

David Johnston

dj.johnston@intel.com

Contents

Introduction

The SP800-90B Assessment suite published by NIST at https://github.com/usnistgov/SP800-90B_EntropyAssessment is expected to be the software tool used by FIPS certification labs to assess the entropy from entropy sources to establish compliance to the input requirements of SP800-90B entropy extractors.

I have been involved in designing, building and assessing RNGs in high volume silicon products at Intel for a number of years. A primary algorithm used for assessing min-entropy is the Markov-Renye min entropy estimate in its original form where -s (number of steps) and –g (group length) parameters were available. This allowed the algorithm to be tried across a number of parameter settings in a way that would pick up on periodicities and correlations that occurred over particular numbers of bits. It has proved affective against modeled data and physical data at producing estimates that are consistently pessimistic and yet sufficiently close to the actual min entropy that the number was usable in claims of min entropy for the purposes of showing input requirements to an extractor are met.

Draft releases of SP800-90B included a number of other tests, some for IID sources and some for Non-IID sources. The IID tests are generally without purpose since no entropy source is IID. The NonIID set include, Most Common Value, Collision, Markov Estimate, Compression, t-Tuple Estimate and LRS Estimate. We only look at the NonIID tests here. No work has been done on the IID tests.

The Non IID tests other than the Markov test proved to be unreliable and give nonsense answers to real world data. The form of the Markov test in SP800-90B differed from the original paper in that it is no longer parametric and so it's utility in being able to 'hunt' for the optimal min entropy parameters is lost.

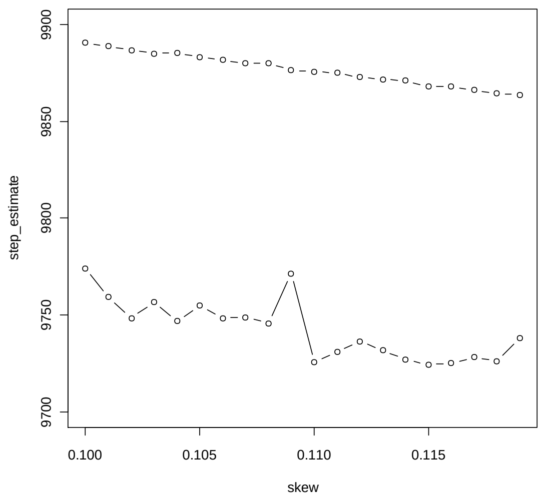
The STS-2.1.2 Suite is an implementation of the SP800-22-Rev1a statistical test suite available at http://csrc.nist.gov/groups/ST/toolkit/rng/documentation_software.html . This is a collection of 15 pass/fail tests of distinguishability from random, suitable for testing uniform PRNGs and data claimed to be full entropy.

Experiences running these tests is discussed here. Also an implementation that is more reliable has been written and is available here: https://github.com/dj-on-github/sp800_22_tests .

For both STS-2.1.2 and the 90B Assessment Suite, the output of the tests was compared with the known properties of calibrated random data.

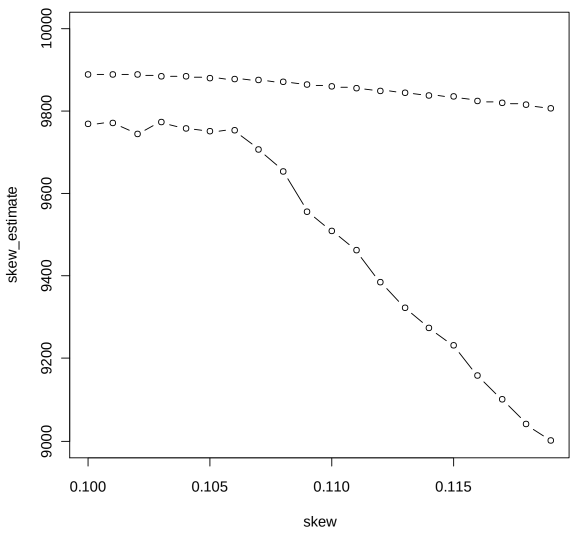
Previous Results with the Markov Test

Results on the performance of the Markov test were shared with the NSA a number of years ago. The following plot shows the min-entropy from a metastable feedback model, with the skew (the relative size of the left and right step sizes) being varied. The upper trace is the changing min entropy. The lower trace is the min entropy estimate of the original Markov test.

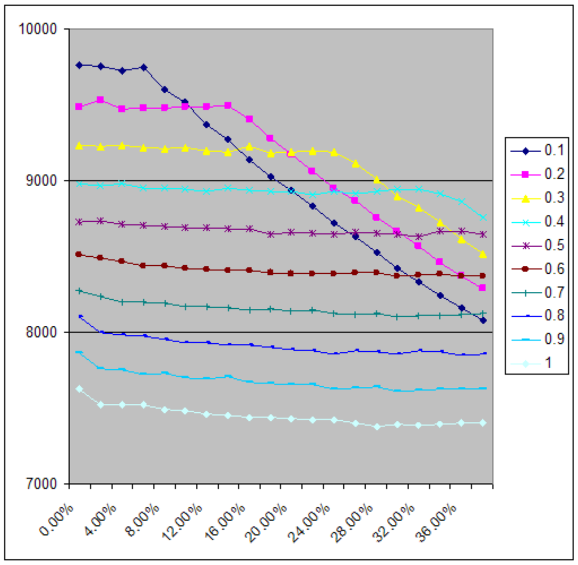


The x axis is set to reflect the genuine range of skew to be expected in silicon at the time. We see the estimate be not far from the actual min entropy and consistently below it by a small amount. This is what we would expect from a good min-entropy test.

If we run the same test with a 2X larger baseline step size (which reduces the entropy by increasing the serial correlation) we find the results go non-linearly pessimistic.



Plotting these estimate curves with more starting points show how the output can yield paradoxical results, where worse entropy is reported as being better. In the plots below, with increasing skew, data with worse entropy is measure as having better entropy, as can be seen on the right hand side, where the higher entropy data sets are being measured as worse than lower entropy data sets, when faces with both bias and serial correlation in the same data.



Entropy Test Validation Procedure

So when using the test, it is necessary to an eye on the results of the tools and tests against modelled data that matches the physical data well in order to check the entropy assessment is realistic. The Entropy Test Validation procedure being

A) Measure base statistics (mean, serial correlation) of the raw data, using a tools such as ent or djent (https://github.com/dj-on-github/djent).

B) Find parameters for the SUMS model (see below for details on SUMS model) that yield the same bias and serial correlation when measured.

C) Check that the measured min-entropy with the Markov test does not stray too far from the actual min entropy of the model data.

The SP800-90B Assessment Algorithms

The SP800-90B Assessment software includes predictor tests that are not in the published 90B drafts. These follow a model I explained at a NIST RNG workshop some years ago. An optimal guessing attack can be defined for most entropy sources, based on the attacker knowing the internal state of the entropy source at any point in time. By measuring the success of this attack at guessing next state output bits will yield an accuract min-entropy estimate. A critical aspect of this is that the predictor was optimal and therefore specific to the entropy source being tested. The versions in the 90B assessment suite use 4 fixed predictors: MultiMCW, Lag, MultiMMC and LZ78Y. These are unlikely to be optimal, in particular because they do not know the internal state of the entropy source before each bit is generated.

What Precipitated This Work?

In writing a guide to testing entropy sources for a forthcoming book, the STS-2.1.2 and SP800-90B assessment suites were examined. However the results were not as expected and this led to looking more deeply.

First the simple things.

```
./noniid_main.py: line 17: import: command not found
from: can’t read /var/mail/util90b
from: can’t read /var/mail/mostCommonValue
from: can’t read /var/mail/noniid_collision
from: can’t read /var/mail/markov
from: can’t read /var/mail/maurer
from: can’t read /var/mail/SP90Bv2_predictors
from: can’t read /var/mail/tuple
from: can’t read /var/mail/LRS
./noniid_main.py: line 35: syntax error near unexpected token
`(’
./noniid_main.py: line 35: ` args = get_parser(’non-IID’).parse_args()’
```

This turns out to be because the code doesn’t include a hash-bang line at the start to tell the system to run the program as python. To fix this, add #!/usr/bin/env python to the first line of noniid_main.py or invoke it with python noniid_main.py.

Next trying it out..

Serially correlated data was fed in with serial correlation = -0.2. 1 bit symbols were used, since the symbol size is actually 1 bit. The probability that a bit differs from the previous is 0.5 + 0.2/|-2| = 0.6 so the probability of correctly guessing the next 8 bits, one bit at a time, while being given the result of each bit before guessing the next is 0.6^8 so the entropy in a byte is $-log_2(0.6^8) \approx 5.89572$ which is 74%.

The entropy reported by the tool was 0.896712 bits of entropy per bit of data. An over assessment.

Trying with 8 bit symbols, which doesn't alter the entropy at all, gave a result of 3.77848 bits of entropy per byte or 47%. This falls below a critical threshold for the SP800-90 specifications which effectively require > 50% min entropy from sources. Also the mathematical security proof of the CBC-MAC extractor relies on the input entropy rate being above 50%.

So the test has rejected a good entropy source with a wildly low entropy assessment compared to the actual entropy.

In the latter test, the individual assessments were.

- MCV: 6.12296 = 76.5%
- Collision: 4.39941 = 55%
- Markov: 4.66484 = 58.3%
- Compression: 3.77848 = 47.2%
- T-Tuple: 5.98329 = 74.8%
- LRS : 7.4 = 92.5%
- MultiMCW Predictor: 6.16595 =77.1%
- Lag Predictor: 7.54826 = 94.35%
- MultiMMC Predictor: 6.059 = 75.7%
- LZ78Y Predictor: 6.05907 = 75.7%

So the Compression test was the one to give the worst result. The Collision and Markov tests substantially underestimated the entropy, the LRS test and lag predictor substantially overestimated the entropy.

Since the defined procedure is to perform all the tests and take the lowest result, it is a significant problem if tests are significantly underestimating the min entropy. It seems sensible that a validation procedure such as the Entropy Test Validation Procedure above be permitted as a way to filter test results of this sort.

Assessing the Assessment Algorithms with djenrandom

The djenrandom program https://github.com/dj-on-github/djenrandom was developed to test entropy assessment algorithms, by feeding in calibrated random data and also outputting the data necessary to see the efficacy of an optimal attack by outputting the actual min-entropy for each output bit, based on the state of the model generating the data.

Three generator models were used to test the 90B Assessment Suite.

- Biased. Each bit has the same bias.

- Correlated. Produces data with a stationary correlation coefficient
- SUMS (Step Update Metastable Source) This implements a feedback variable moving left or right along the metastable resolution probability curve based on the previous bit. Data from this model has both bias and serial correlation and is non stationary, since the feedback variable changes with time. It has been shown to very accurately model physical entropy sources used in Intel products.

For each of these, we can produce data with a well defined min entropy and total entropy compare against the results of the tests.

The procedure for testing the algorithms is to generate a number of random data files across a range of a parameter for each model. For each data file, the jfile is also produced, which gives the entropy of each bits.

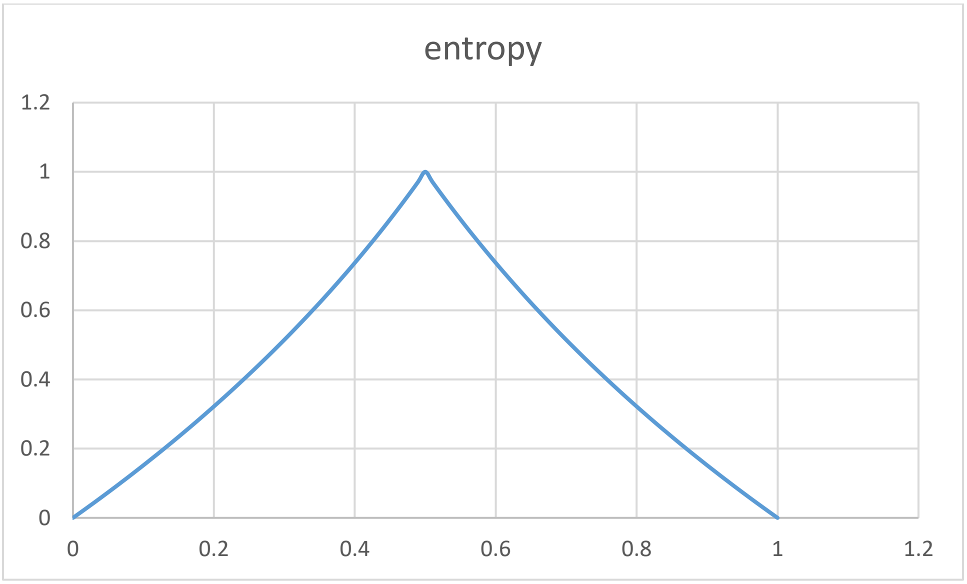
The results of the entropy assessment suite, against the correct answer in the jfile are compared.

Biased Model

In the bias model, data was generated with P(1) from 0.0 to 1.0 in 0.01 steps.

The min entropy of biased data is simply $-\log_2(\max(bias,1-bias))$.

The min entropy against bias is shown in this standard textbook plot.



The data files were generated with 1 bit per symbol and 1 Mibitbit of data each.

```
# od -x  biased_0.40_1mibibit.bin | head
0000000 0001 0000 0000 0000 0100 0001 0100 0100
0000020 0001 0000 0101 0100 0100 0100 0001 0000
0000040 0000 0001 0101 0001 0001 0001 0001 0100
0000060 0001 0001 0000 0000 0001 0001 0000 0000
0000100 0100 0000 0100 0000 0101 0000 0000 0100
```

```
0000120 0100 0100 0001 0000 0100 0100 0101 0101
0000140 0000 0000 0100 0101 0101 0000 0101 0000
0000160 0000 0000 0001 0001 0001 0100 0000 0101
0000200 0100 0000 0000 0001 0001 0101 0100 0000
0000220 0100 0001 0100 0000 0001 0001 0000 0100
```

The jfile contains the entropy of each bit. In the biased file they are all the same.

```
# head  biased_0.40_1mibibit.jfile | head
0.736966
0.736966
0.736966
0.736966
0.736966
0.736966
0.736966
0.736966
0.736966
0.736966
```

A bug that occurs is that the runtime of the tests becomes effectively infinite when the bias is very low E.G when P(1) = 0.01. It seems to be the tuple test that is hanging in the following loop:

```
    while True:
        i += 1
        c = Counter(find_tuples(s, i))
        if verbose:
          print ("\ntuples (and counts):", c)
          print ("max occurances:",c.most_common(1))
        count = c.most_common(1)[0][1]
        Q.append(c.most_common(1)[0][1])

        if Q[i-1]<35:
            break
```

Presumably because the break condition is never met. This complicates the test procedure, since the plan to run data across the full 0.0→1.0 bias range fails to complete at the first data point.

The test sequence was amended to run from the middle out and testing was stopped when the t-tuple test locked up. E.G.:

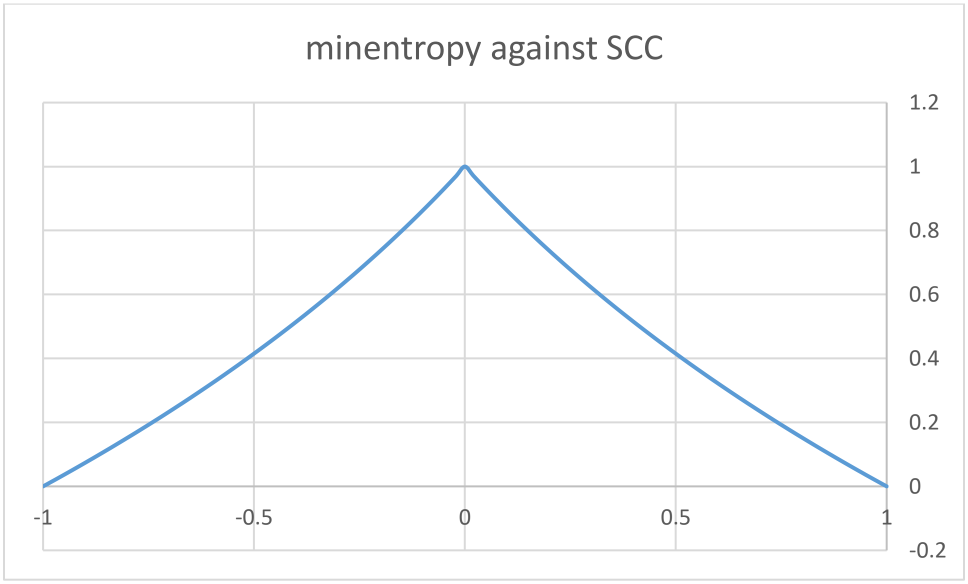
```
#!/usr/bin/env bash
python ../noniid-main.py -v ../testdata/biased_0.50_1mibibit.bin 1
> ../testdata/bias_0.50_result
python ../noniid-main.py -v ../testdata/biased_0.49_1mibibit.bin 1
> ../testdata/bias_0.49_result
python ../noniid-main.py -v ../testdata/biased_0.51_1mibibit.bin 1
> ../testdata/bias_0.51_result
python ../noniid-main.py -v ../testdata/biased_0.48_1mibibit.bin 1
> ../testdata/bias_0.48_result
python ../noniid-main.py -v ../testdata/biased_0.52_1mibibit.bin 1
> ../testdata/bias_0.52_result
python ../noniid-main.py -v ../testdata/biased_0.47_1mibibit.bin 1
> ../testdata/bias_0.47_result
…
```

This sequence seized after 18 hours computation and biases between 0.00 and 0.04, and between 0.97 and 1.00 were unable to complete. So the graphs below are missing those regions of data.

Correlated Model

The correlated model files were generated with serial correlation coefficient from -1 to +1 in 0.02 steps.

The min entropy of serially correlated data is –log2((0.5 * SCC)+0.5) which is shown below



SUMS Model

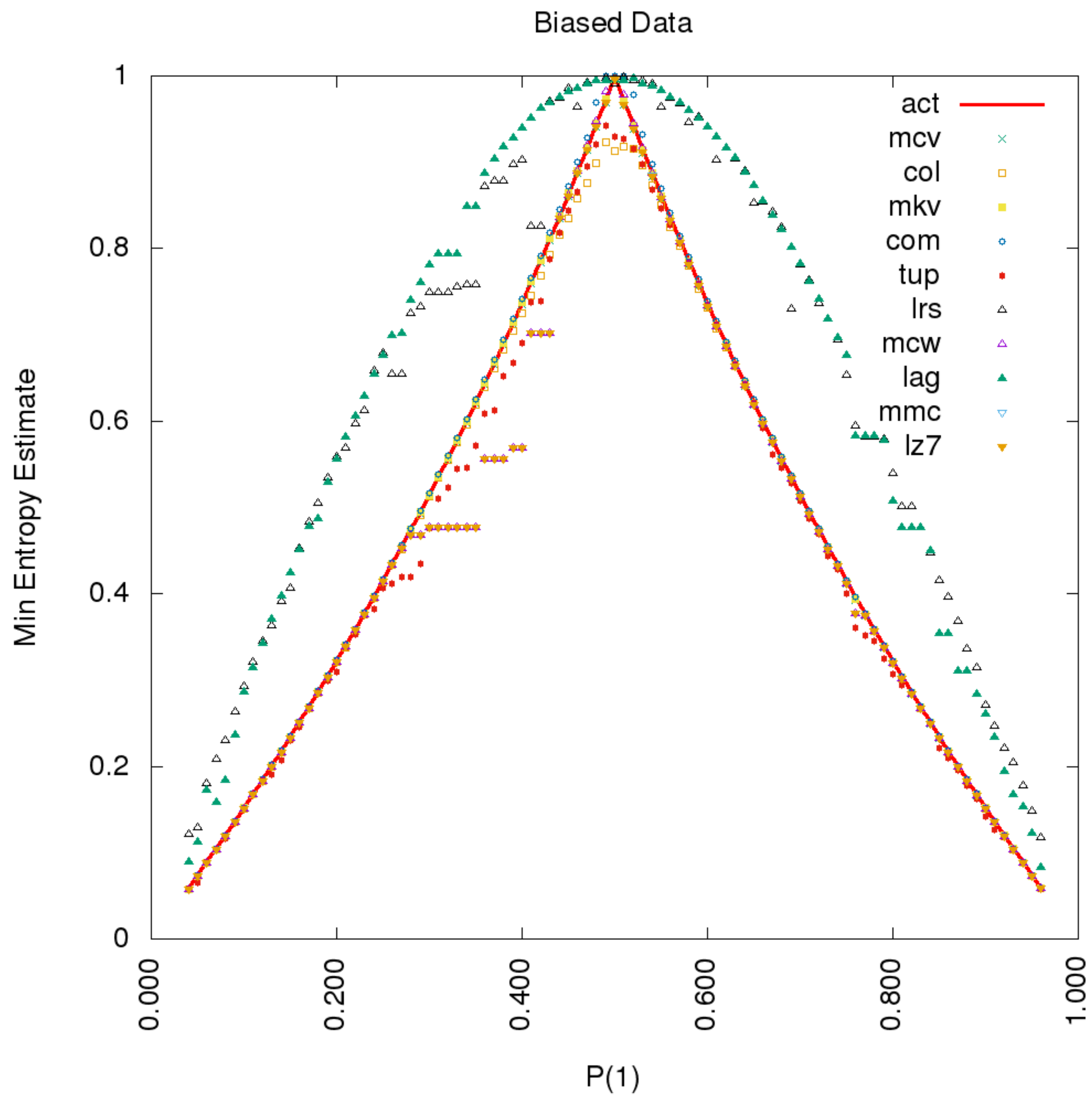
The SUMS (Step Update Metastable Source) models a feedback network balancing a metastable latch, where the output of 1 causes the feedback network to shift 1 step towards producing more zeroes and the output of 0 causes the feedback network to shift 1 step towards producing more ones. The step size in terms of the standard deviation of the noise in the circuit is chosen and can be chosen independently for the left step and right step, which models variation in the relative strength of P and N transistors in such a circuit.

With an asymmetric step size, both bias and correlation will be present. The data generated is will with ration of the left step size to the right step size going from 2.00 to 0.0952. The entropy per bit is variable since the statistics of SUMS sources are non stationary. The actual entropy is taken as the mean of the min entropy of the bits. This is a reasonable assumption given that the input to extractors is generally 768 to 2048 bits and so the total entropy per bit will be close to the mean.

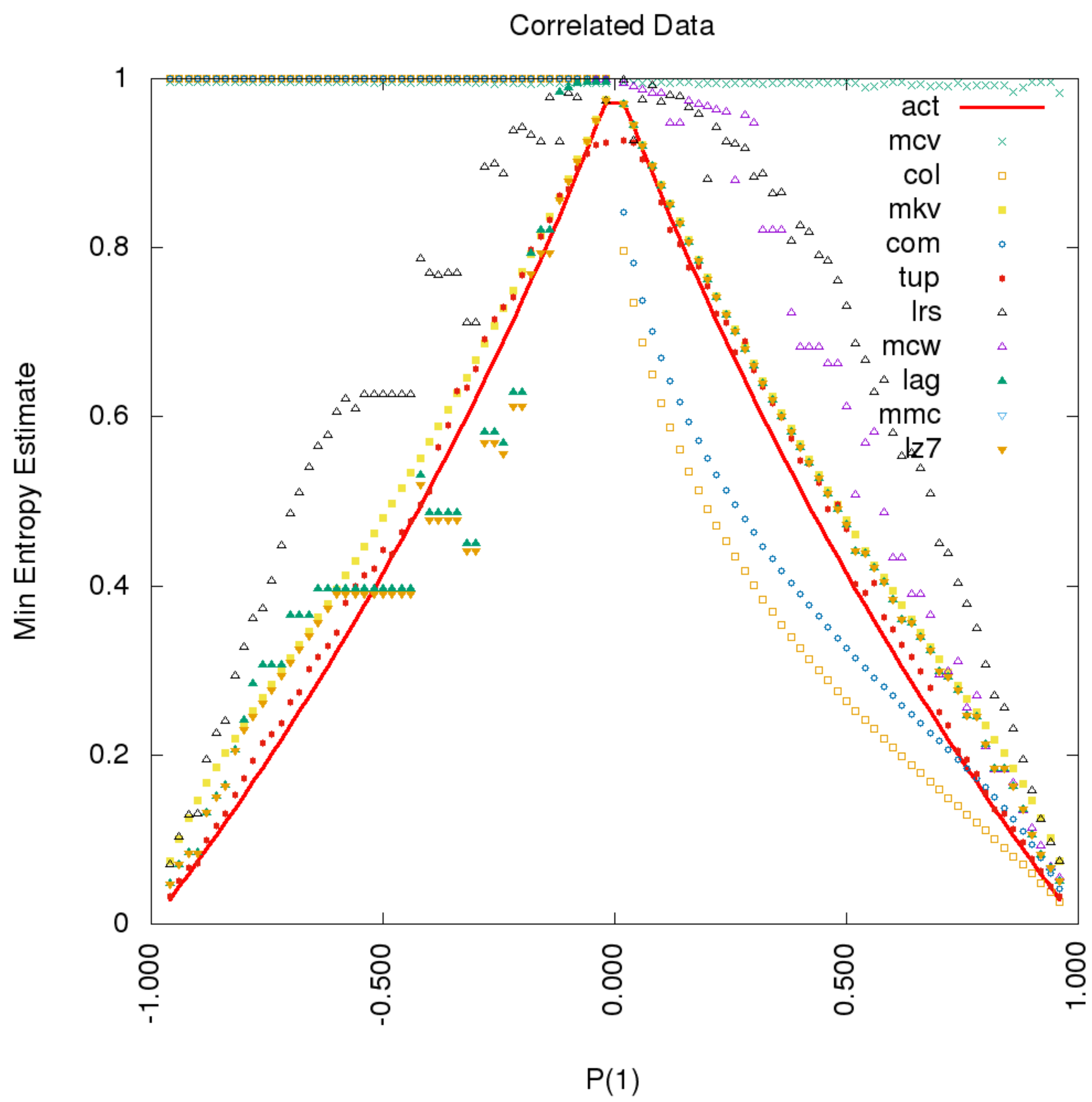
Results

A total of 238 hours of compute time on a 4 core, 8 thread Broadwell i7 CPU were required, resulting from the slow execution time of the nonIID test programs.

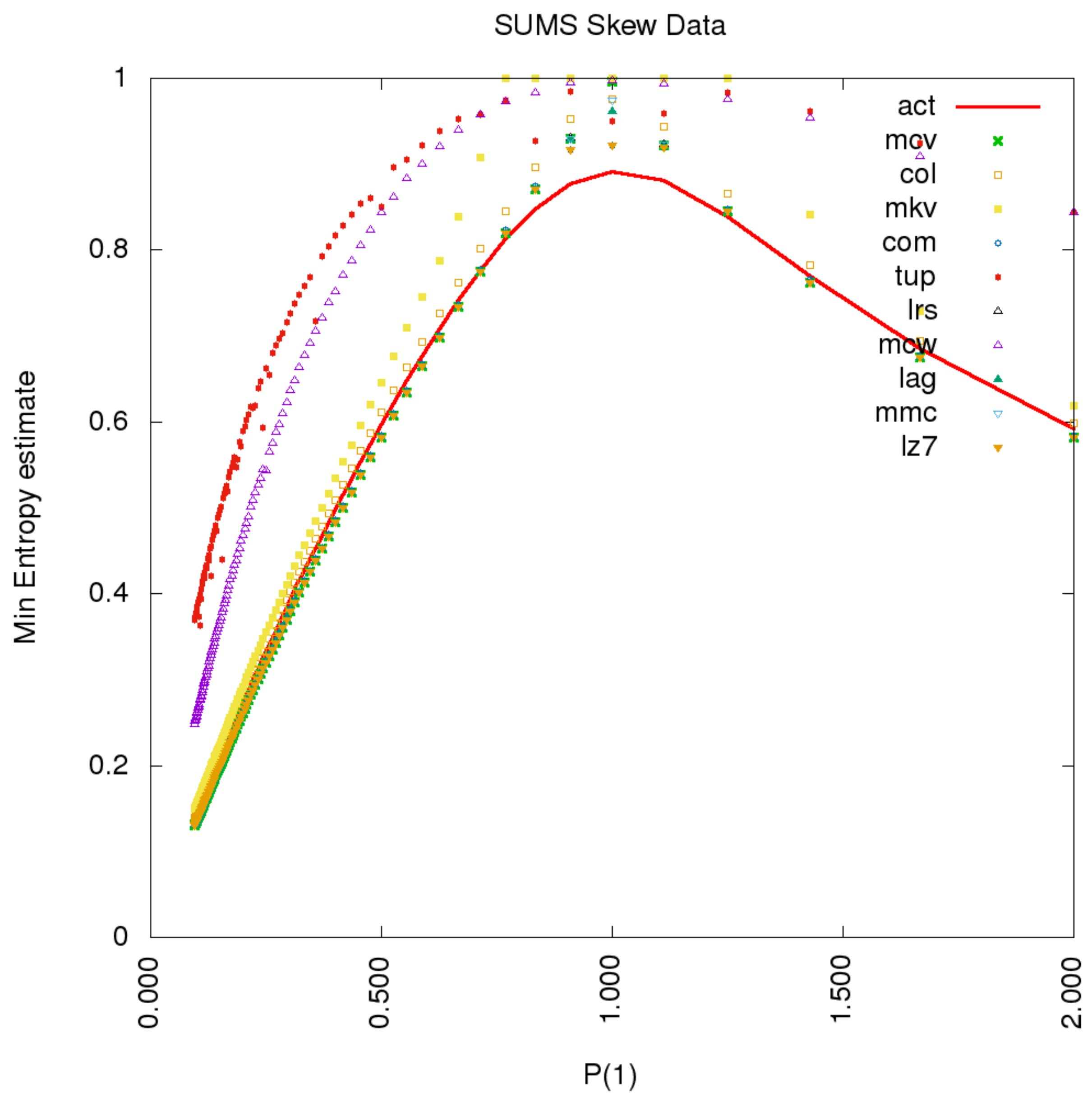
Each graph shows the min entropy estimate by each of the 90B assessment algorithms. The actual entropy is the solid red line. Points below the line are underestimates of the entropy and points above the line are over estimates. The lowest point is the measured min entropy according to the 90B procedure.



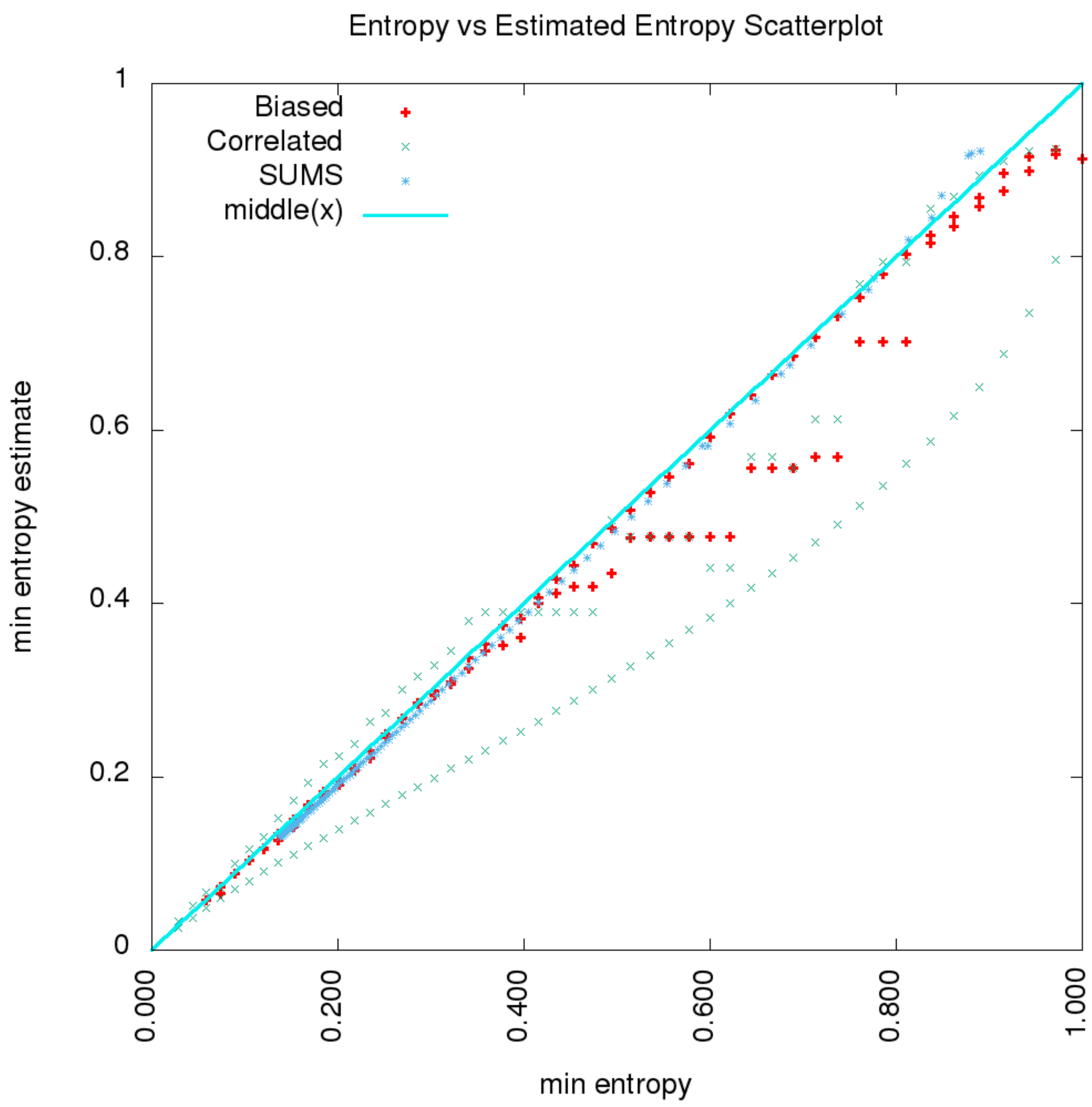
In the biased data we see more under estimation with low bias than high bias.



In the correlated data, we see gross under estimation of the entropy from the collision test. Particularly close to P(1)=0.75, where the measured entropy is well below 0.5 and so marked as not meeting the input requirements of a CBC-MAC extractor when in reality it does.



The SUMS model is relatively well estimated, with the estimated value being both close to and below the actual. The MCV and lz7 predictor are the tests generating these appropriate results.



In this diagram, the blue diagonal line is the actual min entropy line. The points are the min entropy estimate of the 90B assessment suite for the biased, correlated and SUMS models. We can see clearly the excess under estimation with the biased and correlated models.

STS-2.1.2

STS is simpler to analyze. It should pass uniform, unbiased random data and fail non uniform or biased data.

Software usage issues

First we generate a uniform random files and a serially correlated file

```
[root@localhost sts-2.1.2]# djenrandom -b -k 1024 -m correlated --
correlation=0.2 > badmegrand.bin
[root@localhost sts-2.1.2]# djenrandom -b -k 1024 -m pure > megrand.bin
```

When presented with the serially correlated file, it locks up in a loop.

```
[root@localhost sts-2.1.2]# ./assess 8388608
           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: badmegrand.bin

                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.........

igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
```

```
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
igamc: UNDERFLOW
^C
```

When presented with the uniform, unbiased data this happens.

```
[root@localhost sts-2.1.2]# ./assess 8388608
           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: megrand.bin

                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!!!!!!!!!!!!
```

My own implementation of the tests at https://github.com/dj-on-github/sp800_22_tests have no such problems.

```
[root@localhost sts-2.1.2]# djenrandom -b -k 128 -m correlated --
correlation=0.2 > badmegabitrand.bin
[root@localhost sts-2.1.2]# djenrandom -b -k 128 -m pure > megabitrand.bin
[root@localhost sp800_22_tests]# ./sp800_22_tests.py megabitrand.bin
Tests of Distinguishability from Random
TEST: monobit_test
  Ones count   = 523749
  Zeroes count = 524827
  PASS
  P=0.292462749356
TEST: frequency_within_block_test
  n = 1048576
  N = 99
  M = 10591
  PASS
  P=0.230443081147
TEST: runs_test
  prop  0.499485969543
  tau  0.001953125
  vobs  524634.0
  PASS
  P=0.4984920945
TEST: longest_run_ones_in_a_block_test
  n = 1048576
  K = 6
  M = 10000
  N = 75
  chi_sq = 6.9440321132
  PASS
  P=0.326054130504
TEST: binary_matrix_rank_test
  Number of blocks 1024
  Data bits used: 1048576
  Data bits discarded: 0
  Full Rank Count  =  271
  Full Rank -1 Count =  620
  Remainder Count =  133
  Chi-Square =  3.55350014332
  PASS
  P=0.169187100709
TEST: dft_test
  N0 = 498073.600000
  N1 = 498156.000000
  PASS
  P=0.815363333786
TEST: non_overlapping_template_matching_test
  PASS
  P=0.999999999944
TEST: overlapping_template_matching_test
  B =  [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
  m =  10
  M =  1062
  N =  968
  K =  5
  model =  [352, 179, 134, 97, 68, 135]
  v[j] =   [583, 142, 109, 53, 26, 55]
  chisq =  5.92694666222
  PASS
  P=0.313392365595
TEST: maurers_universal_test
  sum = 920272.148609
  fn = 6.19645121474
  PASS
  P=0.999909497336
TEST: linear_complexity_test
  M =  512
  N =  2048
  K =  6
  chisq =  10.4642992262
  P =  0.106412399501
  PASS
  P=0.106412399501
TEST: serial_test
  psi_sq_m   =  20.6425170898
  psi_sq_mm1 =  8.20275878906
  psi_sq_mm2 =  2.67317962646
  delta1     =  12.4397583008
  delta2     =  6.91017913818
  P1         =  0.132634957334
  P2         =  0.140711666172
  PASS
P=0.132634957334
P=0.140711666172
TEST: approximate_entropy_test
  n         =  1048576
  m         =  3
  Pattern 1 of 8, count = 130999
  Pattern 2 of 8, count = 131511
  Pattern 3 of 8, count = 131573
  Pattern 4 of 8, count = 130744
  Pattern 5 of 8, count = 131511
  Pattern 6 of 8, count = 130806
  Pattern 7 of 8, count = 130744
  Pattern 8 of 8, count = 130688
  phi(3)    = -4.382023
  Pattern 1 of 16, count = 65363
  Pattern 2 of 16, count = 65636
  Pattern 3 of 16, count = 65778
  Pattern 4 of 16, count = 65733
  Pattern 5 of 16, count = 65710
  Pattern 6 of 16, count = 65863
  Pattern 7 of 16, count = 65509
  Pattern 8 of 16, count = 65235
  Pattern 9 of 16, count = 65636
  Pattern 10 of 16, count = 65875
  Pattern 11 of 16, count = 65795
  Pattern 12 of 16, count = 65011
  Pattern 13 of 16, count = 65801
  Pattern 14 of 16, count = 64943
  Pattern 15 of 16, count = 65235
  Pattern 16 of 16, count = 65453
  phi(3)    = -5.075164
  AppEn(3)  = 0.693141
  ChiSquare =  12.4640873389
  PASS
  P=0.131667431003
TEST: cumulative_sums_test
PASS
  PASS
P=0.305577573144
P=0.442430147557
TEST: random_excursion_test
J=418
x = -4  chisq = 5.199765        p = 0.391990
x = -3  chisq = 7.793769        p = 0.167974
x = -2  chisq = 9.135702        p = 0.103772
x = -1  chisq = 10.021014       p = 0.074642
x = 1   chisq = 8.065426        p = 0.152667
x = 2   chisq = 0.957480        p = 0.965922
x = 3   chisq = 1.125506        p = 0.951821
x = 4   chisq = 7.686259        p = 0.174395
J too small (J < 500) for result to be reliable
  PASS
P=0.391990457355
P=0.167973649757
P=0.103772351199
P=0.0746419980822
P=0.152666536786
P=0.965921882263
P=0.951821232914
P=0.174395262548
TEST: random_excursion_variant_test
J= 418
x = -9   count=315      p = 0.610934
x = -8   count=325      p = 0.587245
x = -7   count=349      p = 0.468014
x = -6   count=345      p = 0.538280
x = -5   count=326      p = 0.749978
x = -4   count=309      p = 1.007533
x = -3   count=289      p = 1.410869
x = -2   count=293      p = 1.764945
```

```
x = -1   count=357      p = 1.491804
x = 1    count=409      p = 0.220102
x = 2    count=398      p = 0.282391
x = 3    count=446      p = 0.306235
x = 4    count=506      p = 0.813421
x = 5    count=523      p = 0.855953
x = 6    count=501      p = 0.612017
x = 7    count=485      p = 0.454449
x = 8    count=472      p = 0.340981
x = 9    count=489      p = 0.421130
J too small (J=418 < 500) for result to be reliable
  PASS
P=0.61093446716
P=0.587244803579
P=0.46801446709
P=0.538280170116
P=0.749977848335
P=1.00753314331
P=1.41086861161
P=1.76494529602
P=1.49180376354
P=0.22010219462
P=0.282391247362
P=0.306235047482
P=0.813421253311
P=0.855952979078
P=0.612017179721
P=0.454448830363
P=0.340980853691
P=0.421129584159

SUMMARY
-------
monobit_test                              0.292462749356     PASS
frequency_within_block_test               0.230443081147     PASS
runs_test                                 0.4984920945       PASS
longest_run_ones_in_a_block_test          0.326054130504     PASS
binary_matrix_rank_test                   0.169187100709     PASS
dft_test                                  0.815363333786     PASS
non_overlapping_template_matching_test    0.999999999944     PASS
overlapping_template_matching_test        0.313392365595     PASS
maurers_universal_test                    0.999909497336     PASS
linear_complexity_test                    0.106412399501     PASS
serial_test                               0.132634957334     PASS
approximate_entropy_test                  0.131667431003     PASS
cumulative_sums_test                      0.305577573144     PASS
random_excursion_test                     0.0746419980822    PASS
random_excursion_variant_test             0.22010219462      PASS
[root@localhost sp800_22_tests]#
```

When presented with bad data, a number of the tests fail as expected.

```
Tests of Distinguishability from Random
TEST: monobit_test
  Ones count   = 523636
  Zeroes count = 524940
  PASS
  P=0.202862837549
TEST: frequency_within_block_test
  n = 1048576
  N = 99
  M = 10591
  FAIL
  P=0.000558334264137
TEST: runs_test
  prop  0.499378204346
  tau  0.001953125
  vobs  419012.0
  FAIL
  P=0.0
TEST: longest_run_ones_in_a_block_test
  n = 1048576
  K = 6
  M = 10000
  N = 75
  chi_sq = 326.001032551
  FAIL
  P=2.17997651876e-67
TEST: binary_matrix_rank_test
  Number of blocks 1024
  Data bits used: 1048576
  Data bits discarded: 0
  Full Rank Count  =  312
  Full Rank -1 Count =  605
  Remainder Count =  107
  Chi-Square =  7.71555439421
  FAIL
  P=0.0211148816189
TEST: dft_test
  N0 = 498073.600000
  N1 = 495081.000000
  FAIL
  P=2.23835386643e-17
TEST: non_overlapping_template_matching_test
  PASS
  P=0.968197614405
TEST: overlapping_template_matching_test
  B =  [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
  m =  10
  M =  1062
  N =  968
  K =  5
  model =  [352, 179, 134, 97, 68, 135]
  v[j] =   [124, 106, 106, 82, 88, 462]
  chisq =  3592.97054309
  FAIL
  P=0.0
TEST: maurers_universal_test
  sum = 894459.269203
  fn = 6.02264583751
  PASS
  P=0.92176892431
TEST: linear_complexity_test
  M =  512
  N =  2048
  K =  6
  chisq =  9.12166377323
  P =  0.166850517359
  PASS
  P=0.166850517359
TEST: serial_test
  psi_sq_m   =  131877.881104
  psi_sq_mm1 =  86194.9239502
  psi_sq_mm2 =  42281.6710815
  delta1     =  45682.9571533
  delta2     =  1769.70428467
  P1         =  0.0
  P2         =  0.0
  FAIL
P=0.0
P=0.0
TEST: approximate_entropy_test
  n         =  1048576
  m         =  3
  Pattern 1 of 8, count = 189612
  Pattern 2 of 8, count = 125822
  Pattern 3 of 8, count = 83518
  Pattern 4 of 8, count = 125988
  Pattern 5 of 8, count = 125822
  Pattern 6 of 8, count = 83684
  Pattern 7 of 8, count = 125988
  Pattern 8 of 8, count = 188142
  phi(3)    = -4.341428
  Pattern 1 of 16, count = 113941
  Pattern 2 of 16, count = 75671
  Pattern 3 of 16, count = 50158
  Pattern 4 of 16, count = 75664
  Pattern 5 of 16, count = 50040
  Pattern 6 of 16, count = 33478
  Pattern 7 of 16, count = 50488
  Pattern 8 of 16, count = 75500
  Pattern 9 of 16, count = 75671
  Pattern 10 of 16, count = 50151
  Pattern 11 of 16, count = 33360
  Pattern 12 of 16, count = 50324
  Pattern 13 of 16, count = 75782
  Pattern 14 of 16, count = 50206
  Pattern 15 of 16, count = 75500
  Pattern 16 of 16, count = 112642
  phi(3)    = -5.014275
  AppEn(3)  = 0.672847
  ChiSquare =  42572.874713
  FAIL
  P=0.0
TEST: cumulative_sums_test
PASS
  PASS
P=0.101609561292
P=0.288041757035
TEST: random_excursion_test
J=116
x = -4  chisq = 2.452981        p = 0.783559
x = -3  chisq = 8.022292        p = 0.155011
x = -2  chisq = 7.863170        p = 0.163940
x = -1  chisq = 5.476906        p = 0.360485
x = 1   chisq = 4.494147        p = 0.480667
x = 2   chisq = 8.762259        p = 0.118931
x = 3   chisq = 14.140014       p = 0.014744
x = 4   chisq = 15.040973       p = 0.010189
J too small (J < 500) for result to be reliable
  PASS
P=0.78355896771
P=0.155011367263
P=0.163939583535
P=0.360484895778
P=0.480666914291
P=0.118930921033
P=0.0147440674052
```

```
P=0.0101886955514
TEST: random_excursion_variant_test
J= 116
x = -9   count=67       p = 0.551712
x = -8   count=74       p = 0.503436
x = -7   count=70       p = 0.592280
x = -6   count=70       p = 0.643876
x = -5   count=79       p = 0.572561
x = -4   count=82       p = 0.596583
x = -3   count=88       p = 0.581318
x = -2   count=102      p = 0.375239
x = -1   count=109      p = 0.324967
x = 1    count=137      p = 0.974901
x = 2    count=167      p = 1.366943
x = 3    count=187      p = 1.474057
x = 4    count=186      p = 1.228259
x = 5    count=182      p = 1.021324
x = 6    count=176      p = 0.839839
x = 7    count=179      p = 0.811166
x = 8    count=186      p = 0.839061
x = 9    count=181      p = 0.731863
J too small (J=116 < 500) for result to be reliable
  PASS
P=0.551712252661
P=0.50343646656
P=0.592280133018
P=0.643876387599
P=0.572560626046
P=0.596583045392
P=0.581318358976
P=0.375239387193
P=0.32496684181
P=0.97490052543
P=1.36694348192
P=1.4740572674
P=1.2282592111
P=1.02132435997
P=0.839838766434
P=0.811166269134
P=0.839060777599
P=0.731863192305

SUMMARY
-------
monobit_test                             0.202862837549     PASS
frequency_within_block_test              0.000558334264137  FAIL
runs_test                                0.0                FAIL
longest_run_ones_in_a_block_test         2.17997651876e-67  FAIL
binary_matrix_rank_test                  0.0211148816189    FAIL
dft_test                                 2.23835386643e-17  FAIL
non_overlapping_template_matching_test   0.968197614405     PASS
overlapping_template_matching_test       0.0                FAIL
maurers_universal_test                   0.92176892431      PASS
linear_complexity_test                   0.166850517359     PASS
serial_test                              0.0                FAIL
approximate_entropy_test                 0.0                FAIL
cumulative_sums_test                     0.101609561292     PASS
random_excursion_test                    0.0101886955514    PASS
random_excursion_variant_test            0.32496684181      PASS
```

STS 2.1.2 Overlapping Template Matching Reference Frequencies

The overlapping template matching test performs a chi squared test of the measured pattern frequencies against the expected reference frequencies.

Section 2.8 gives test frequencies of

$\pi_0 = 0.364091$

$\pi_1 = 0.185659$

$\pi_2 = 0.139381$

$\pi_3 = 0.100571$

$\pi_4 = 0.070432$

$\pi_5 = 0.139865$

But it states that these are just examples for the example data and the real frequencies are

$\pi_0 = 0.324652$

$\pi_1 = 0.182617$

$\pi_2 = 0.142670$

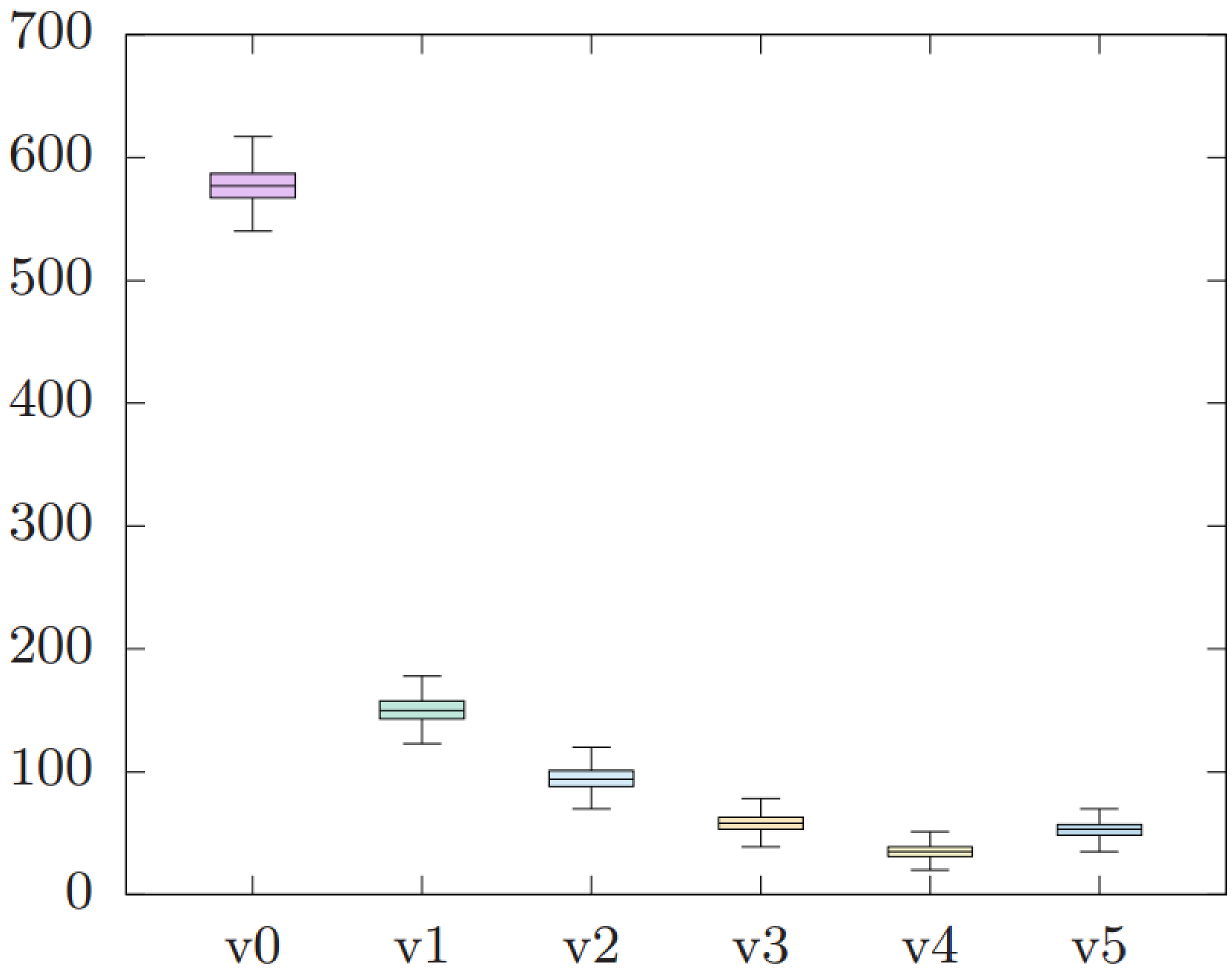
$\pi_3 = 0.106645$

$\pi_4 = 0.077147$

$\pi_5 = 0.166269$

However in section 3.8 and in the STS-2.1.2 software the first ‘example’ set of reference frequencies are used, not the ‘correct’ series.

I chose to empirically verify which is the correct set by passing 800 runs through the test each over 1,028,016 bits (1 MiBit) of data. The plot below shows the empirically measures frequencies as a box plot. The 800 data points for each box are unskewed and normally distributed.



This gives frequencies that are dramatically different to both the ‘example’ and ‘correct’ reference frequencies.

Table 9.4: Measured χ^2 Probabilities from 800 runs of the Overlapping Pattern Matching Test

	v_0	v_1	v_2	v_3	v_4	v_5
Mean	576.91625	150.3025	94.2275	58.3075	35.355	52.89125
Frequency	0.595988	0.155271	0.097342	0.060235	0.036524	0.054640

The conclusion is the reference frequencies and maybe the derivation procedure are wrong, since they do not match the empirical results. The text and the code are inconsistent even with the values that are supplied.

STS-2.1.2 Serial Test.

The serial test includes an equation that is entirely unclear.

$$\frac{2^m}{n}\sum_{i_1 \ldots i_m} v_{i_1 \ldots i_m}^2 - n$$

Is the –n an exponent or a subtraction? Is it inside the summation or outside? That is 4 possible combinations.

Some experimentation shows that the correct interpretation is this:

$$\left(\frac{2^m}{n}\sum_{i_1 \ldots i_m} v_{i_1 \ldots i_m}^2\right) - n$$

The specification would be a lot more easily interpreted if this form of equation was used instead of the version with the ambiguous floating – n.

90B Summary

- The SP800-90B Entropy Assessment python code provided by NIST fails to include the appropriate #! Header.
- When fed serially correlated data, the min entropy measured by the entropy assessment tool is very far below the actual min entropy, making the assessment statistically invalid.

STS-2.1.2 Summary

- The code is simply broken and unusable
    - It fails to complete when given data it should fail.
    - When given good data, it doesn't report any result
- The reference frequencies in the Overlapping Template Test are presented inconsistently and appear to be wrong.
- The equations in the Serial Test are ambiguous.