Computational Physics – Lecture 22

(6 May 2020)

Recap

In the previous lecture, we learnt how to obtain non-uniform deviates.

Today's plan

In today's class, we want to see how to test pseudo-random sequences for randomness.

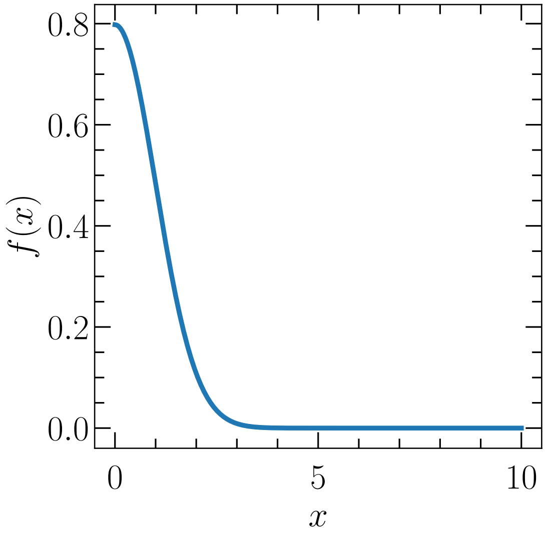
Rejection method

Suppose we want random numbers distributed according to the distribution

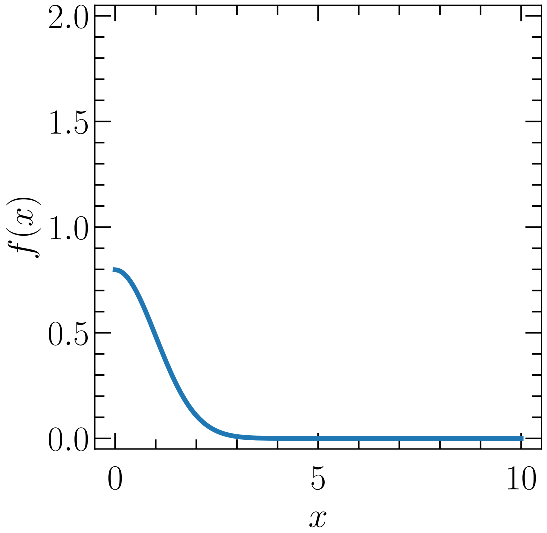
$$f(x) = \sqrt{\frac{2}{\pi}} e^{-x^2/2} \qquad (x \geq 0). \tag{1}$$

(This is the “error function”, $\mathrm{erf}(x)$.)

Rejection method



Rejection method



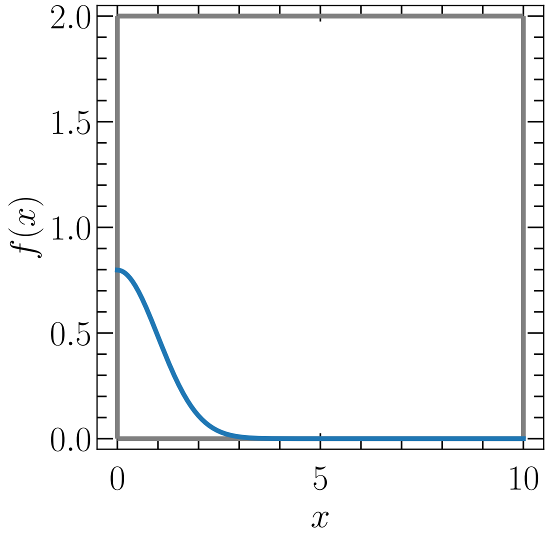
Rejection method

This function $f(x)$ is completely enveloped by a function $g(x)$, which is given by

$$g(x) = \begin{cases} 2 & \text{if } 0 < x < 10 \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

(if we ignore the ≈ 0 probability tail of $f(x)$ at $x > 10$.)

Rejection method



Rejection method

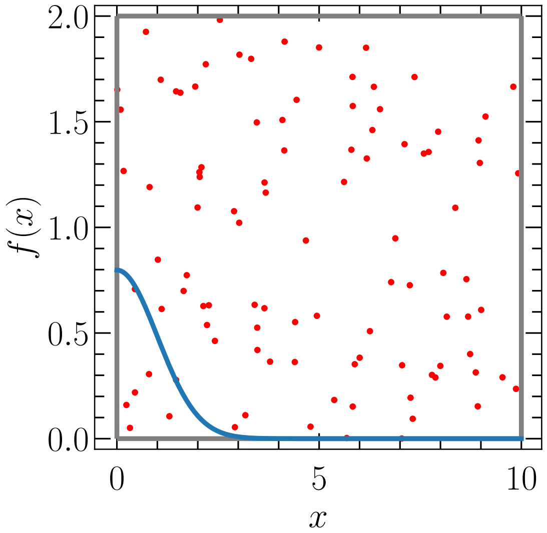
Let us now uniformly sample the two-dimensional area contained within the grey lines.

We can do this by something like

```
x = np.random.rand(100)*10.0
y = np.random.rand(100)*2.0
```

which gives us 100 points within the grey box with coordinates (x, y).

Rejection method

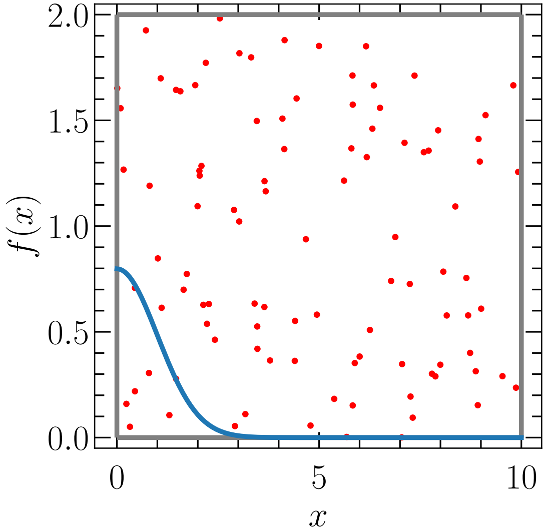


Rejection method

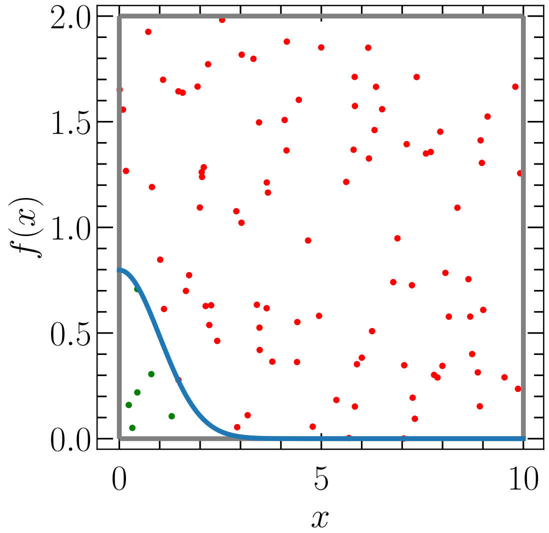
Now the number of points in a part of this figure is proportional to the area of that part.

So the number of points enclosed within the graph of $f(x)$ is proportional to the area under this graph.

Rejection method



Rejection method



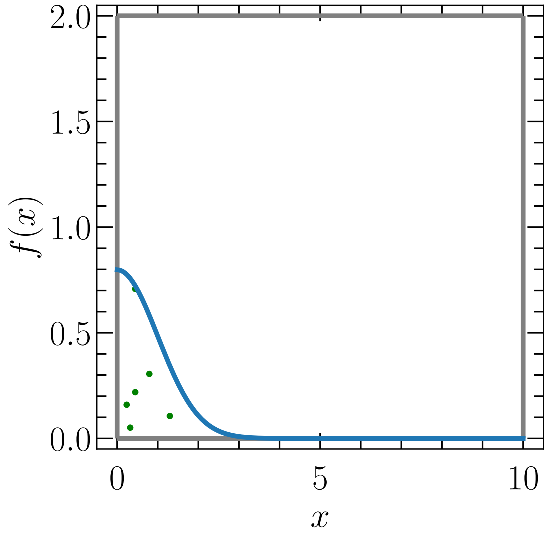
Rejection method

We can now throw away ("reject") the bad points.

```
y_bad = y[y > f(x)]
x_bad = x[y > f(x)]

y_good = y[y < f(x)]
x_good = x[y < f(x)]
```

Rejection method

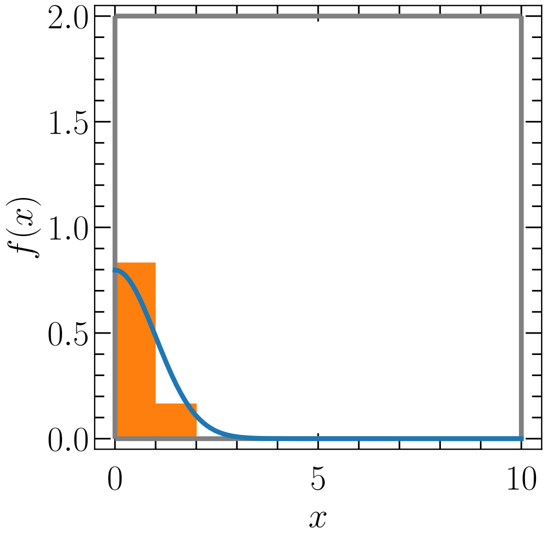


Rejection method

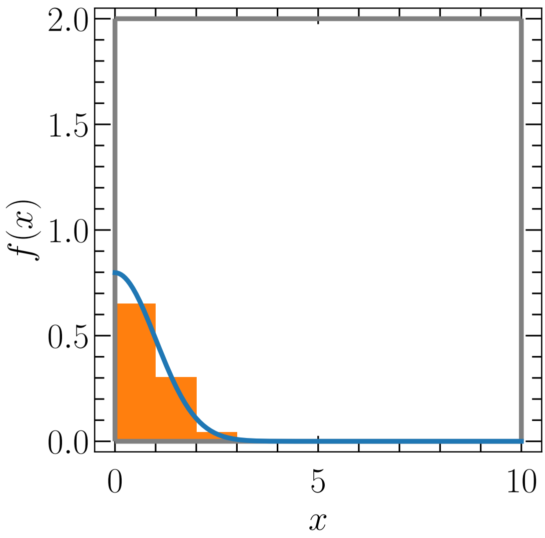
The x coordinates of the good points now have the desired distribution.

```
plt.hist(x_good, range=(0.0,10.0), density=True)
```

Rejection method

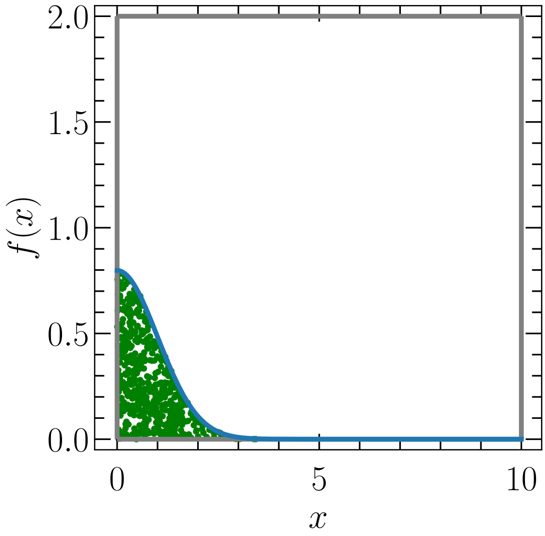


Rejection method



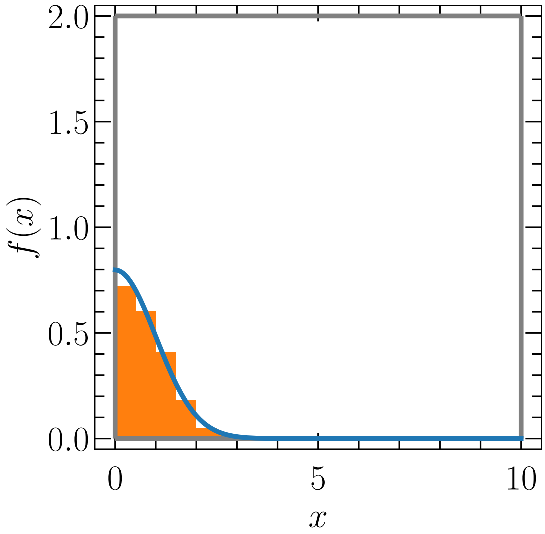
(Result with 1,000 points.)

Rejection method



(Result with 10,000 points.)

Rejection method



(Result with 10,000 points.)

Rejection method

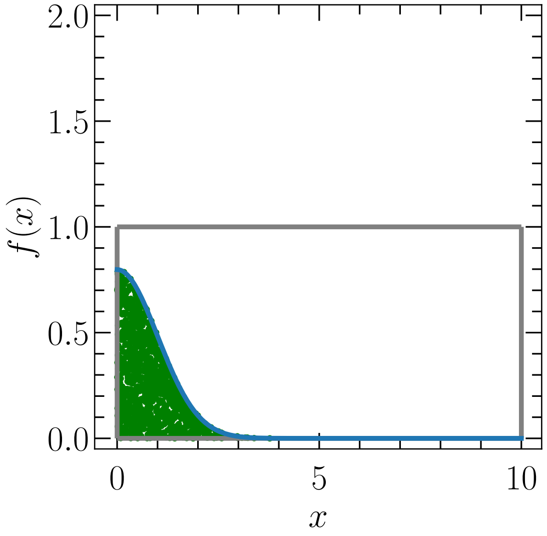
Now in this method if we sample n points in our envelope then only $\approx f_p \cdot n$ points will be accepted, where f_p is the ratio of the area under $f(x)$ to the total area in the envelope.

For example, in our 100-point case, we only got 3 good points (3%). When we took 10,000 points, we got $\approx$ 500 good points (about 5%).

So our method is very inefficient. (The problem worsens exponentially in higher dimensions.)

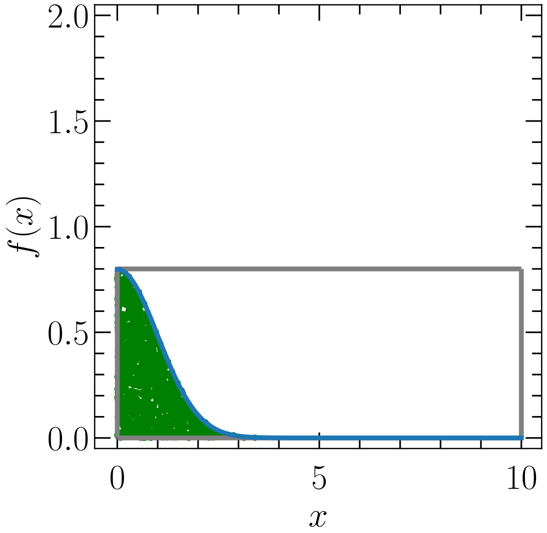
We can reduce this wasted effort by choosing a better envelope so that the rejection area is as small as possible.

Rejection method



(This envelope yields a 10% efficiency: 1,000/10,000 good points.)

Rejection method



(This envelope yields a 12% efficiency: 1,200/10,000 good points.)

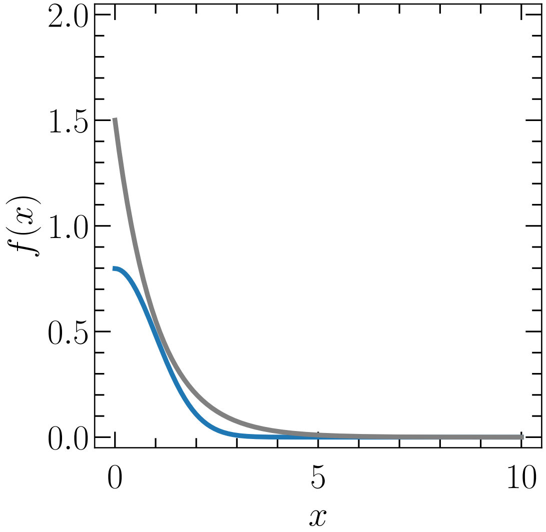
Rejection method

Or we can consider a more nontrivial envelope: a second function $g(x)$, given by

$$g(x) = 1.5 \times e^{-x}, \tag{3}$$

which is a (scaled) exponential distribution.

Rejection method



Rejection method

How do we get points under the exponential? Using the transformation method.

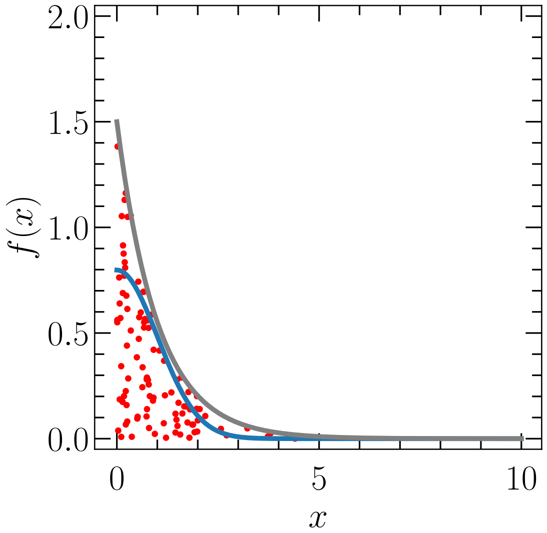
Something like this will work:

```
x = np.random.rand(100)
x = -np.log(x)
```

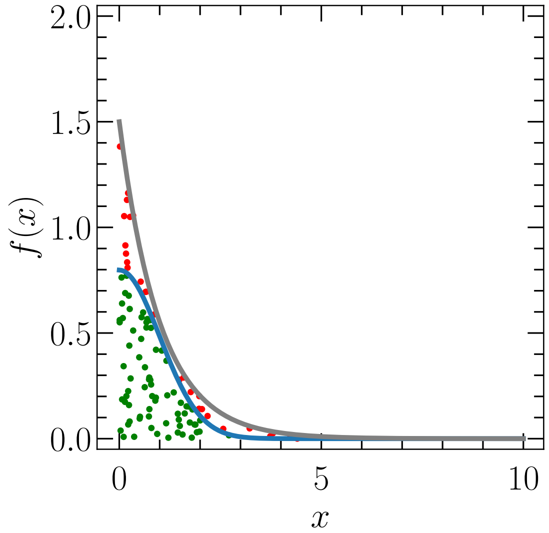
And for each x, let us get a value y that is uniformly random between 0 and $g(x)$:

```
y = np.random.rand(100)*g(x)
```

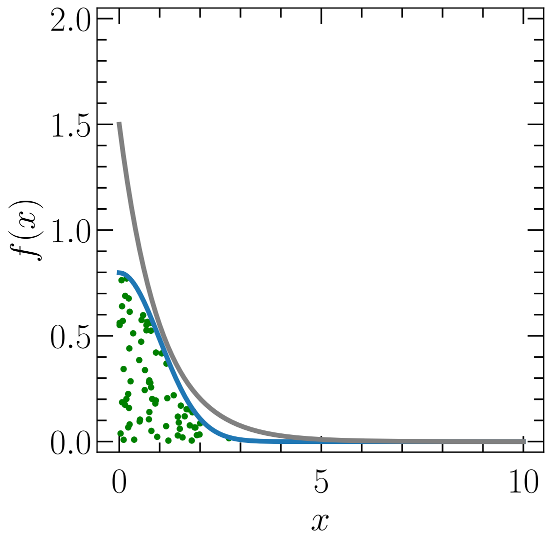
Rejection method



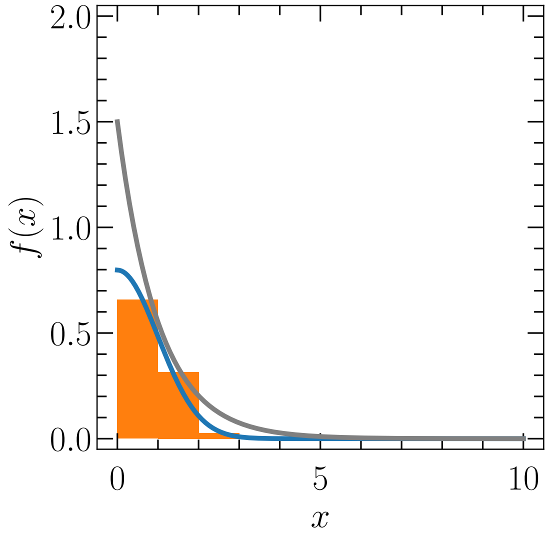
Rejection method



Rejection method



Rejection method



Rejection method

The efficiency is now $\approx 70\%$, which is a big improvement over our original 5%.

A passing comment: Analysis of Random Number Generators is poorly studied subject (unlike Analysis of Algorithms).

Tests of randomness

How do we decide whether an output sequence from a random number generator is “sufficiently random”?

(Somewhat academic topic, but gives good introduction to theoretical frequentist statistics; useful in other applications.)

We ourselves are poor judges of randomness; humans keep noticing patterns even in truly random distributions.

Another problem is that the answer to the question “Is this sequence random?” is itself probabilistic.

Tests of randomness

Suppose we simulate throwing two dice 144 times (recall exercise from a previous lecture.)

That is, we obtain two independent random integers of the six integers 1, 2, 3, 4, 5, and 6, 144 times.

There are 36^{144} possible outcomes, all equally probable.

So your random number can give a sequence consisting of say all 2s.

Would you call it random? Unclear! You can only say that it has a high chance being non-random.

Tests of randomness

So we need objective, unbiased, tests.

These are provided by the theoretical statistics.

There are hundreds of tests available in the literature. We will consider some of them.

Tests of randomness

Unfortunately, if a sequence passes tests $T_1, T_2, \ldots, T_n$, there is no guarantee that it will also pass test T_{n+1}.

In practice, people apply a few tests and then start using the qualifying sequence until something breaks down.

(This is another reminder to use good libraries, whose output sequences are often robustly checked with several tests.)

χ^2 test

A basic test is the chi-squared test.

Consider again our simulation of two dice. Each dice yields an integer 1, 2, 3, 4, 5, 6 with equal probability.

The possible scores and their expected probabilities are:

Score s	2	3	4	5	6	7	8	9	10	11	12
Probability p_s	$\frac{1}{36}$	$\frac{1}{18}$	$\frac{1}{12}$	$\frac{1}{9}$	$\frac{5}{36}$	$\frac{1}{6}$	$\frac{5}{36}$	$\frac{1}{9}$	$\frac{1}{12}$	$\frac{1}{18}$	$\frac{1}{36}$

For example, a score of 4 can be obtained in three ways: $1 + 3$, $2 + 2$, $3 + 1$, so $p_4 = 3/36 = 1/12$.

χ^2 test

Now if I throw these dice n times, I expect score s approximately np_s times on average.

Let us do this on a computer:

```
s = np.random.randint(1, high=7, size=144)
    + np.random.randint(1, high=7, size=144)
```

χ^2 test

I get, e.g.,

```
8 10  4  8  2 10 10  8  8  9 10  6  8 10  9 11  5  8  5
3  7  3  7  6  9 10 10  6  4 11 11 10  7  7  7  9  8  7
4 12  4  6  6  4  8  8  9  2  5  6 11  9  5  6  8  8  2
6 10 10  3 11  4  7 11  6  2  7  2  7 10  6  7  3  8  6
5  9  5  6  5  9 10 10  7  7  4  2  9  7  7  2 10  6  9
4  6 10  7 10  5 11  4  7 10  7  7  4  9  4  4  5 11  2
8 11  7  9  3 10 12  4 11  9 10 10  7  4  6 10  6  2  6
7  8  8  8  6  6  6  3  7  8  5
```

We can count the number of times we got each score by, say, doing

```
v, n = np.unique(s, return_counts=True)
```

χ^2 test

We get

Score s	2	3	4	5	6	7	8	9	10	11	12
Observed count Y_s	9	6	14	10	20	22	17	13	21	10	2
Expected count np_s	4	8	12	16	20	24	20	16	12	8	4

Observed number is different from expected number.

Is the observed sequence random?

χ^2 test

We can improve this question by considering the quantity

$$V = (Y_2 - np_2)^2 + (Y_3 - np_3)^2 + \ldots + (Y_{12} - np_{12})^2 \qquad (4)$$

A bad random number generator will give a high value of V.

In our case, $V = 180$, so we can now ask, “What is probability that V is this high for true dice?”

If the answer is a small number, then we would suspect our random number generator.

χ^2 test

Now in the above definition of V, suppose we weight each term according to its expected count.

We then get

$$V = \frac{(Y_2 - np_2)^2}{np_2} + \frac{(Y_3 - np_3)^2}{np_3} + \ldots + \frac{(Y_{12} - np_{12})^2}{np_{12}}, \qquad (5)$$

In our case we have $V = 18.7625$.

This V is called the χ^2 statistic.

We now ask “What is probability that V is this high?”

χ^2 test

χ^2 test says that this probability is given by the χ^2 distribution

$$f(x, k) = \frac{1}{2^{k/2}\Gamma(k/2)} x^{k/2-1} \exp(-x/2), \tag{6}$$

where the parameter k is called “degrees of freedom” and Γ is the Gamma function.

What is the value of k (degrees of freedom) in our case?

The χ^2 test says that this is equal to the number of distinct values that our random variable can have, minus one.

So for us k is $11 - 1 = 10$.

χ^2 test

Our V is 18.7625. What is the probability of V being 18.7625 or above?

This can be calculated as

```
    1.0 - scipy.stats.chi2.cdf(18.7625, 10.0)
```

and is 0.04338507352406551 so about 4%.

So? Is 4% good? Or is it too small?

χ^2 test

The χ^2 test gives the following prescription.

If your value of V is v, then

- If $P(V > v)$ is less than 1% or greater than 99% then your numbers are “not sufficiently random”
- If $P(V > v)$ is between 1% and 5%, or between 95% and 99%, then your numbers are “suspect”
- If $P(V > v)$ is between 5% and 10%, or between 90% and 95%, then your numbers are “almost suspect”
- If $P(V > v)$ is between 10% and 90% then your numbers are “sufficiently random”

So, by the χ^2 test, our random numbers are “suspect”.

χ^2 test

The χ^2 test is done three times on three runs of your random number generator, and if two of these three results are suspect, the the generator is declared to be suspect.

I did two more runs of my dice simulation and got $V = 6.645833$ and $V = 7.558333$.

The probablity of V to be greater than these values is 0.758403760118509 and 0.6718904732581429.

So our random numbers look good!

Activity

Suppose we got these counts in two runs of our dice simulation:

Score s	2	3	4	5	6	7	8	9	10	11	12
Observed counts 1 Y_s	4	10	10	13	20	18	18	11	13	14	13
Observed count 2 np_s	3	7	11	15	19	24	21	17	13	9	5
Expected count np_s	4	8	12	16	20	24	20	16	12	8	4

Calculate V. Apply the χ^2 test and label the random numbers as “not sufficiently random”, “suspect”, “almost suspect”, or “sufficiently random”.

χ^2 test

So to summarise, the χ^2 method is as follows:

Suppose you have n independent observations in k categories (random numbers in k bins). Then

- Compute the χ^2 statistic V
- Compute the probability of V being this high
- Label your random numbers as good or bad

χ^2 test

Note that the χ^2 test is discrete due to the degrees-of-freedom parameter.

Also notice how *ad hoc* the χ^2 test feels.

KS test

Another classical test of random numbers is the Kolmogorov-Smirnov test.

The KS test is useful when a random variable can take infinitely many values, e.g., random real number between 0 and 1.

KS test

How does the KS test work?

It compares the cumulative distribution function (CDF) of your sample to the CDF of the parent probability distribution function.

KS test

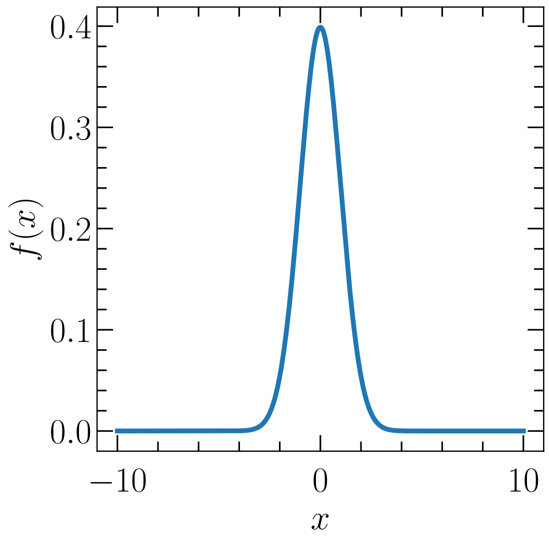
Consider the Normal distribution

$$f(x) = \sqrt{\frac{1}{2\pi}} e^{-x^2/2} \qquad (-\infty < x < \infty). \tag{7}$$

Now, the Numpy function `numpy.random.standard_normal` claims to produce random numbers that follow the normal distribution.

Let us use the KS test to see if this is a good generator.

KS method



KS method

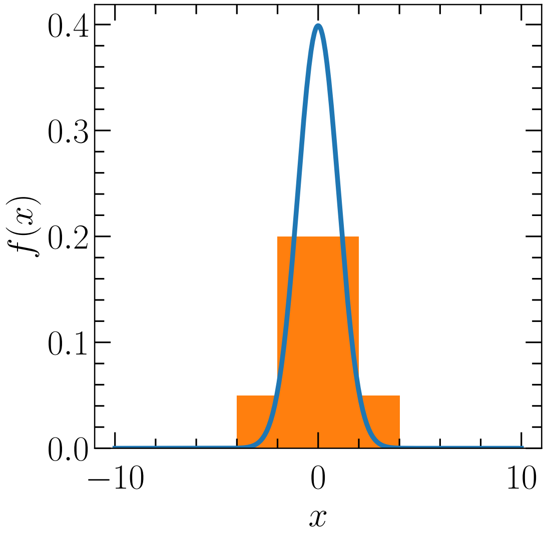
Let us sample 10 random numbers from this distribution using the Numpy function:

```
s = np.random.standard_normal(10)
```

And plot a histogram:

```
plt.hist(s, range=(-10.0,10.0), density=True)
```

KS method



KS method

Now when we have n discrete random numbers in a computer, we define an empirical CDF as

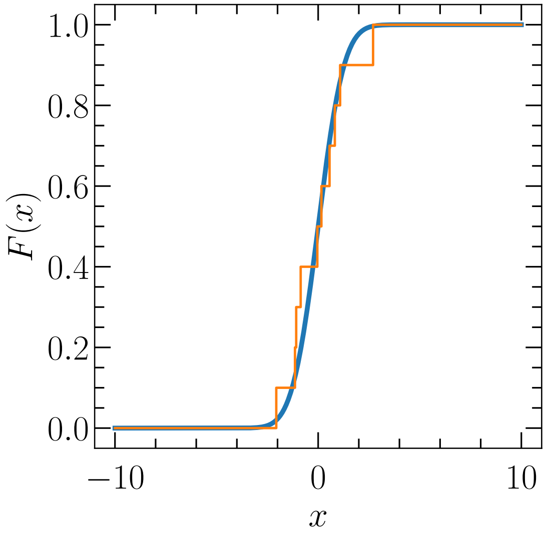
$$F_n(x) = \frac{\text{Number of random numbers less than } x}{n} \tag{8}$$

The analytical CDF is defined as

$$F(x) = \int_{-\infty}^{x} dx' f(x'). \tag{9}$$

Let us check these in our case.

KS method



KS method

The KS test asks if the difference between the two CDFs is small enough.

It asks us to calculate the quantity

$$K_n^+ = \sqrt{n} \sup_{-\infty<x<\infty} (F_n(x) - F(x)) \tag{10}$$

Here sup is the supremum, and the $\sqrt{n}$ is a detail that tries to keep the test equally robust at all sample sizes.

In our case, I get $K_n^+ = 0.640595$.

KS method

Like before, we ask: "What is the probability that K_n^+ is this high?".

The KS test says that this probability is given by the one-parameter Kolmogorov distribution.

Tables of the Kolmogorov distribution are available. (Also see `scipy.stats`.)

KS method

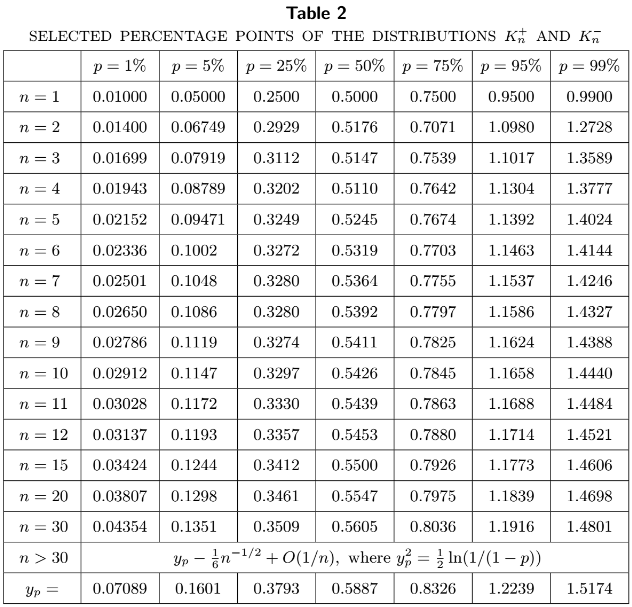


Table 2

SELECTED PERCENTAGE POINTS OF THE DISTRIBUTIONS K_n^+ AND K_n^-

	$p = 1\%$	$p = 5\%$	$p = 25\%$	$p = 50\%$	$p = 75\%$	$p = 95\%$	$p = 99\%$
$n = 1$	0.01000	0.05000	0.2500	0.5000	0.7500	0.9500	0.9900
$n = 2$	0.01400	0.06749	0.2929	0.5176	0.7071	1.0980	1.2728
$n = 3$	0.01699	0.07919	0.3112	0.5147	0.7539	1.1017	1.3589
$n = 4$	0.01943	0.08789	0.3202	0.5110	0.7642	1.1304	1.3777
$n = 5$	0.02152	0.09471	0.3249	0.5245	0.7674	1.1392	1.4024
$n = 6$	0.02336	0.1002	0.3272	0.5319	0.7703	1.1463	1.4144
$n = 7$	0.02501	0.1048	0.3280	0.5364	0.7755	1.1537	1.4246
$n = 8$	0.02650	0.1086	0.3280	0.5392	0.7797	1.1586	1.4327
$n = 9$	0.02786	0.1119	0.3274	0.5411	0.7825	1.1624	1.4388
$n = 10$	0.02912	0.1147	0.3297	0.5426	0.7845	1.1658	1.4440
$n = 11$	0.03028	0.1172	0.3330	0.5439	0.7863	1.1688	1.4484
$n = 12$	0.03137	0.1193	0.3357	0.5453	0.7880	1.1714	1.4521
$n = 15$	0.03424	0.1244	0.3412	0.5500	0.7926	1.1773	1.4606
$n = 20$	0.03807	0.1298	0.3461	0.5547	0.7975	1.1839	1.4698
$n = 30$	0.04354	0.1351	0.3509	0.5605	0.8036	1.1916	1.4801
$n > 30$	$y_p - \frac{1}{6}n^{-1/2} + O(1/n)$, where $y_p^2 = \frac{1}{2}\ln(1/(1-p))$						
$y_p =$	0.07089	0.1601	0.3793	0.5887	0.8326	1.2239	1.5174

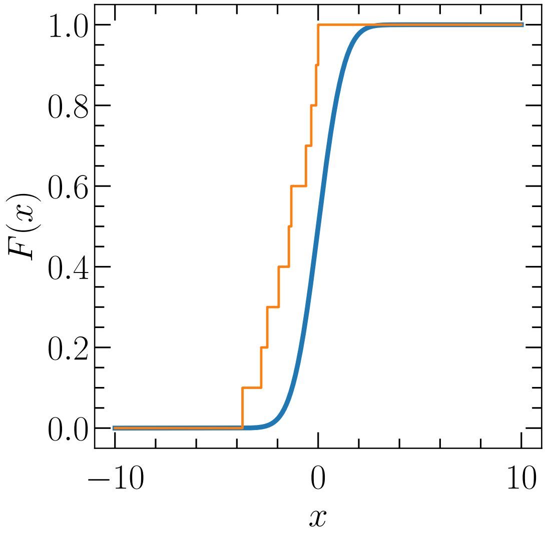
KS method

Our K_n^+ was 0.640595, so the probability of K_n^+ to be higher than this value is between 50% and 25% ($n = 10$).

By similar criteria as before, we can term our random numbers as sufficiently random.

Suppose another random number generator gives me the following output:

KS method



KS method

The value of K_n^+ is 1.597341

From our table, we find that the probability of K_n^+ being this high is less than 1%.

So we conclude that these random numbers are not sufficiently random.

KS method

So to summarise, the KS method is as follows:

Suppose you have n random numbers putatively sampled from a distribution with CDF $F(x)$. Then

- Compute the empirical CDF, $F_n(x)$
- Compute the KS statistic K_n^+
- Consult a table of the Kolmogorov distribution to find the probability of K_n^+ being this high
- Label your random numbers as good or bad

KS method

For many parameter values, the linear congruential generator is known to pass the χ^2 test but fail the KS test.

Other tests

There are many other tests:

- Spectral test
- Frequency test
- Serial test
- Partition test
- Coupon collector's test
- Permutation test
- Run test
- Maximum-of-t test
- Collision test
- ...

Bayesian statistics

The two tests we saw today may feel a bit unreliable, overly complicated, and ad hoc.

In the last 10–20 years, this criticism has led to the development of an entirely new way of doing statistics.

This is called Bayesian Statistics. (The older 20th-century way was called “Frequentist statistics”.)

Bayesian Statistics provides new ways of labelling random numbers good or bad. We will see examples later.