Computational Physics – Lecture 7

Andrei Alexandru

March 23, 2020

1 Introduction

This week we will discuss error estimates and error propagation. Last week we discussed χ^2-fitting and showed how to use the value of χ^2 to determine whether the fit model describes the data well. As we discussed last week, fitting can be used to extract physical parameters from empirical data when we have an accurate model for the process generating the data. However, the discussion last week only focused on extracting the parameters that minimize χ^2 and we did not discuss how to estimate the error-bars on these parameters. This week we will discuss *bootstrap*, a powerful technique that allows us to estimate the errors via *resampling*. Note that while we use χ^2-fitting as a motivation for studying *bootstrap*, this method is very general and can be easily applied in other contexts.

To motivate the material consider the following situation: we are interested in computing the gravitational constant g in a class experiment. Each student is given a photograph of a falling object, shot using a stroboscopic method that takes snapshots of the object at fixed time intervals: $t_0, t_1, \ldots$. Each student measures the vertical position of the object for each snapshot producing a list of datapoints $(t_0, y_0), (t_1, y_1), \ldots$. At the end of class we collect all these lists and we use them to produce a set of mean values and errors: $(t_0, \bar{y}_0, \sigma_0), (t_1, \bar{y}_1, \sigma_1), \ldots$ and fit a quadratic curve $y(t) = y_0 + v_0 t + \frac{1}{2} g t^2$, to extract the gravitational constant g. The methods discussed last week can be used to produce an estimate for g. This week we plan to show how to provide error bars for our estimate. Note that we assume that the time intervals $t_0, t_1, \ldots$ are fixed, that is they are all the same for all students, the only difference in the data lists collected from the students have to do with their measurements of y's.

A more general statement of this problem is to take an empirical data set to be a list of $\{X\} = \{X_1, \ldots, X_n\}$, where we assume X_i to be a complete subset of data, statistically independent from the other subsets, and assumed to be drawn from the same distribution (for the example above X_i is the entire list of measurements produced by one student and $\{X\}$ is the collection of measurements produced by the entire class.) Our measurement process, for example extracting g from the data provided by all students, is just a map between the empirical data set $\{X\}$ and the measurement space: different datasets map to different measurement values. We will called functions of this type a *statistic*s $\hat{\alpha}[\{X\}]$'s (see definition below). We want to estimate the errors on these measurements.

The plan for this week is the following: we will discuss first *error propagation*, that is how to relate the error in the inputs in our analysis with the error estimates for the outputs. We will discuss how to propagate errors analytically, at least for simple functions, and then write a code to check this numerically. We will use this code to show one of the limitations of this method, when the underlying assumptions fail. We will then discuss another limitation, that is when an analytical connection is not easily available between the inputs and the outputs. In this case numerical methods can be used to simulate the input fluctuation to help determine the induced fluctuations in the outputs. This is basically the method we used last week to map out the fluctuations of χ^2 when we generated synthetic data with correct fluctuations. Finally, we will discuss *bootstrap* as a means to generate input fluctuations without having to repeat the process that generated the empirical inputs for our analysis, since in many cases of interest this process is not feasible.

2 Error propagation

It is often the case that we need to compute derived quantities based on stochastic inputs. The stochasticity of the inputs can be traced to experimental errors—tolerances of the measuring devices and methods, etc—or

they could be intrinsic, as is the case for quantum mechanics. In any case, we are also usually required to provide estimates for the errors associated with these derived quantities, so we need to *propagate the errors*.

When the errors on the intrinsic inputs is small, that is small enough that the changes induced on the final results due to fluctuations are small enough to neglect correction beyond the linear terms, a standard formula can be used. Assume that $f(x_i)$ is the derived quantity we wish to evaluate and x_i are the stochastic inputs each with mean $\overline{x}_i$ and standard deviation σ_i[1]. The expected value for f is then $\overline{f} \equiv f(\overline{x}_i)$ and its standard deviation, its error estimate[2], is

$$\sigma_f^2 = \sum_i \left| \frac{\partial f}{\partial x_i}(\overline{x}_i) \right|^2 \sigma_i^2 \,. \tag{1}$$

This formula is correct assuming that σ_i are small and that the fluctuations of x_i around its mean as statistically independent (see the Appendix B for a derivation.)

As simple applications of this consider the fluctuations of the difference, $d = x - y$, and ratio of two variables, $r = y/x$. We have

$$\begin{aligned} \sigma_d^2 &= \left(\frac{\partial s}{\partial x}\right)^2 \sigma_x^2 + \left(\frac{\partial s}{\partial y}\right)^2 \sigma_y^2 = \sigma_x^2 + \sigma_y^2 \,, \\ \sigma_r^2 &= \left(\frac{\partial s}{\partial x}\right)^2 \sigma_x^2 + \left(\frac{\partial s}{\partial y}\right)^2 \sigma_y^2 = \frac{\overline{y}^2}{\overline{x}^4}\sigma_x^2 + \frac{1}{\overline{x}^2}\sigma_y^2 \,. \end{aligned} \tag{2}$$

Note that for the difference (and the sum) the errors add in *quadrature*.

To check numerically this facts, we write a code to generate normally distributed variables x and y and compute their ratio. The code is listed below using the Box-Muller routine we implemented last week to generate normally distributed numbers.

```
1  #define _XOPEN_SOURCE
2  #include <stdlib.h>
3  #include <stdio.h>
4  #include <math.h>
5
6  double gausrnd()
7  {
8    double u1 = drand48();
9    double u2 = drand48();
10   return sqrt(-2*log(u1))*cos(2*M_PI*u2);
11 }
12
13 double getratio(double mx, double sx, double my, double sy)
14 {
15   double x = mx + sx*gausrnd();
16   double y = my + sy*gausrnd();
17
18   return x/y;
19 }
20
21 int main(int argc, char** argv)
22 {
23   int N = atoi(argv[1]);
24   double mx = atof(argv[2]);
25   double sx = atof(argv[3]);
26   double my = atof(argv[4]);
```

[1] See the appendix A for a discussion about the importance of mean and standard deviation.
[2] σ^2 is called the *variance* and σ is the standard deviation.

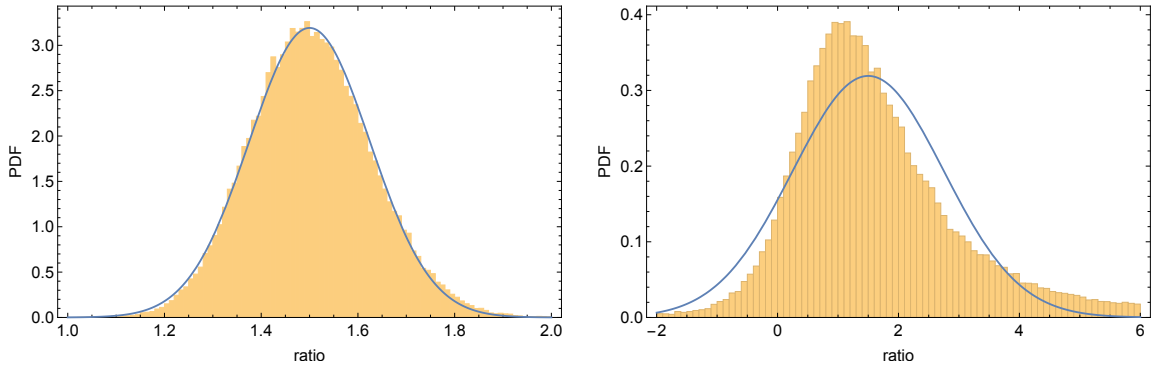

Figure 1: Empirical distribution for the ratio compared with the one derived from error propagations. For both plots we have $\overline{y} = 3$ and $\overline{x} = 2$. Left: $\sigma_x = 0.1$ and $\sigma_y = 0.2$. Right: $\sigma_x = 1$ and $\sigma_y = 2$.

```
27    double sy = atof(argv[5]);
28
29    for(int i=0; i<N; ++i) printf("%e\n", getratio(mx, sx, my, sy));
30
31    return 0;
32  }
```

We run this code to generate a large number of samples both for the case where the fluctuations of x and y are small and large. In Fig. 1 we compare the histogram of the data produced by the code above with a normal distribution centered at $\overline{r} = \overline{y}/\overline{x}$ and standard deviations, σ_r, calculated using the formula above. We see a good agreement with the distribution predicted by the propagation of errors for the case where the fluctuations in x are small. For the case where the fluctuations are large we start seeing deviations from the formula above.

In any case the histogram in the right panel of Fig. 1 represents the correct PDF for the ratio. While the function we analyzed here is relatively simple, the same method can be used to analyze any output function. The idea is that we generate many replica inputs, that are statistically similar to the expected input, and then compute the outputs for all the replicas to generate the probability distribution of the output. This assumes that there is a numerically cheap way to generate replica inputs. For our example this was practical since the underlying distribution for the inputs is a gaussian that we can replicate numerically very efficiently. In the case where this is not feasible, a workaround is provided by the *bootstrap* method discussed in the next section.

3 Bootstrap method

Let us state the general setup for this kind of problem. Assume that the entire data set $\{X\}$ is composed of subsets X_i that are statistically independent a generated by a statistically similar process. In the case discussed in the previous section X_i represents a pair (x_i, y_i).

Assume that we want to estimate the mean value and standard deviation for some *statistic* $f[\{X\}]$. For the case where generating replicas of $\{X\}$ is not feasible, we can use bootstrap to generate similar samples using the following process. Assume that $\{X\}$ has N subsets X_i. To generate a new $\{X\}$ we choose at random one subset X_i, with equal probability for $i \in \{1, 2, \ldots, N\}$. We repeat this step N times, making sure that at each step we pick any of the original subsets, even if they were already picked previously. The union of these N subsets will be our new set $\{X\}'$. On this new sample we compute the statistic $f[\{X\}']$.

We repeat the resampling process $N_{\rm seq}$ times, with $N_{\rm seq} = 10^2$–10^4. The variance σ_f is then estimated using

$$\bar{f} = \frac{1}{N_{\rm seq}} \sum_s f[\{X\}_s], \tag{3}$$

$$\sigma_f \approx \sqrt{\frac{\sum_s (f[\{X\}_s] - \bar{f})^2}{N_{\rm seq}}}. \tag{4}$$

A justification for this method is presented in Appendix C.

As an example, we will use this method to estimate the error of the standard deviation for 100 numbers drawn from a Gaussian distribution with mean at 1 and standard deviation of 2. Note that with a bit of effort we can compute analytically the error of the standard deviation (see Appendix D), but we will use the bootstrap method to evaluate this numerically in a more straightforward process. The *statistics* for the average and standard deviation assume the usual form:

$$\texttt{av}[\{y\}] = \frac{1}{N}\sum_{i=1}^{N} y_i\,, \quad \texttt{std}[\{y\}] = \sqrt{\frac{1}{N}\sum_{i=1}^{N} y_i^2 - \left(\frac{1}{N}\sum_{i=1}^{N} y_i\right)^2}\,. \tag{5}$$

On my machine, the sequence of $N = 100$ numbers generated by my code had an average of 0.908 and a standard deviation of 2.075. To estimate the errors on these quantities we use the bootstrap method described above. Using 100 bootstrap samples my code estimates the error for the average to be 0.229 and the error for the standard deviation to be 0.152. These numbers compare well with the theoretical estimates (see Appendix D):

$$\begin{aligned} \sigma_{\texttt{av}} &= \frac{\sigma}{\sqrt{N}} = \frac{2}{10} = 0.2\,, \\ \sigma_{\texttt{std}} &= \sigma\sqrt{\frac{N-1}{N} - \frac{2}{N}\left(\frac{\Gamma(N/2)}{\Gamma([N-1]/2)}\right)^2} \approx 0.141\,. \end{aligned} \tag{6}$$

Above σ is the standard deviation of the PDF we used to generate the sample of $N = 100$ numbers, which in our case is 2. Note that the formula for the standard deviation of `av` is valid for any underlying PDF, whereas the standard deviation for `std` is derived for a Gaussian PDF. Our bootstrap code makes no assumption about the underlying distribution and should produce a reliable estimate for most cases. The code used to derived these numerical results is listed below.

```
#define _XOPEN_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void compute_stats(double* y, int N, double* av, double* std)
{
  double sum=0, sumsq=0;
  for(int i=0; i<N; ++i)
  {
    sum += y[i];
    sumsq += y[i]*y[i];
  }

  *av = sum/N;
  sumsq /= N;

  *std = sqrt(sumsq - (*av)*(*av));
}
```

```
20
21 double gaussrnd()
22 {
23   double U1 = drand48();
24   double U2 = drand48();
25   return sqrt(-2*log(U1))*cos(2*M_PI*U2);
26 }
27
28 void generate_data(double* y, int N)
29 {
30   for(int i=0; i<N; ++i) y[i] = 1+2*gaussrnd();
31 }
32
33 void bootstrap(double* y, int N, int k, double* av, double* er)
34 {
35   double *res = malloc(k*sizeof(double));
36   double *array = malloc(N*sizeof(double));
37   double dummy;
38
39   for(int i=0; i<k; ++i)
40   {
41     for(int j=0; j<N; ++j) array[j] = y[(int)floor(drand48()*N)];
42     compute_stats(array, N, &dummy, res+i);
43   }
44
45   compute_stats(res, k, av, er);
46
47   free(array);
48   free(res);
49
50 }
51
52 int main(int argc, char** argv)
53 {
54   int N=100;
55   double *y = malloc(N*sizeof(double));
56
57   generate_data(y, N);
58
59   double av, std;
60   compute_stats(y, N, &av, &std);
61
62   printf("av: %e  std: %e \n", av, std);
63   bootstrap(y, N, 100, &av, &std);
64   printf("bootstrap mean of std: %e error on std: %e\n", av, std);
65
66   free(y);
67
68   return 0;
69 }
```

Note that to change the *statistic* we want to estimate the errors for all we have to do is change the code on line 42, assuming that the input data set is a list of numbers as was the case above. In the code above we reused the routine `compute_stats` to evaluate the standard deviation for the sample (note that the routine also computes the mean, but we stored this results in a `dummy` variable and discarded it.)

In a more generic context, where the input is a set of entries (in our example the list of measurements provide by each student) that have more complex information, it may be sensible to define a struct to contain all the relevant info for each entry and redefine the bootstrap procedure to resample the list of entries. An example code is included below, for a problem where the entries are a set of points and we want to fit a quadratic and estimate the error for the linear term. We use `struct dpoint` and `chisq` function discussed in Lecture 6.

```
typedef struct
{
  int n; // number of points in each entry
  double *x; // x coordinates
  double *y; // y coordinates
} entry;

double linear_coef(entry* entries, int nentries)
{
  // assume all entries have the same number of points
  // assume x's to be all the same for all entries
  // average over each y's and get the mean and error for mean
  double *res = malloc(nentries*sizeof(double));
  int npoints = entries[0].n;
  struct dpoint *all = malloc(npoints*sizeof(struct dpoint));
  for(int i=0; i<npoints; ++i)
  {
    for(int j=0; j<nentries; ++j) res[j] = entries[j].y[i];
    compute_stats(res, nentries, &all[i].y, &all[i].sigma);
    all[i].x = entries[0].x[i];
  }
  double coef[3];
  chisq(coef, 3, all, npoints);
  free(all);
  free(res);
  return coef[1];
}

void bootstrap(entry* entries, int N, int k, double* av, double* er)
{
  double *res = malloc(k*sizeof(double));
  entry *array = malloc(N*sizeof(entry));

  for(int i=0; i<k; ++i)
  {
    for(int j=0; j<N; ++j) array[j] = entries[(int)floor(drand48()*N)];
    res[i] = linear_coef(array, N);
  }

  compute_stats(res, k, av, er);

  free(array);
  free(res);
}
```

A Statistics overview: mean and standard deviation

In general the result of the analysis/measurement is a *statistic*, that is a function $\hat{f}[\{X\}]$ that maps a data set to a number $\{X\} \mapsto f[\{X\}] \in \mathbb{R}$ (for example, the slope and the intercepts in a linear fit to the data are statistics). Since the input is stochastic, when you repeat the process the output is generally different but the outcome is still expected to be characterized by a probability function. In general we do not report the full probability distribution as a result of the measurement, but rather some summary characterization. Often this includes the mean value of the measurement and its standard deviation.

Assume that the process that generates the empirical input $\{X\}$ is characterized by the probability distribution function $P[\{X\}]$. The estimated value for this statistic $E[f]$ is the average value for this function when evaluated over all possible data sets:

$$E[f] \equiv \sum_{\{X\}} P[\{X\}] f[\{X\}]. \tag{7}$$

The *standard deviation* of this statistic

$$\sigma_f = \sqrt{E[(f - E[f])^2]} = \sqrt{E[(f)^2] - E[f]^2} \tag{8}$$

is used to indicate its error because it measures the typical fluctuation of this quantity. For most distributions any individual measurement $f(\{X\})$ is very likely to be within a couple of σ_f's of its mean $E[f]$. This happens not only for well behaved distributions (like the normal distribution). A powerful theorem due to Chebyshev states that for any distribution $P(\{X\})$ the probability that $f(\{X\})$ is more than $k \times \sigma_f$ away from $E[f]$ is smaller than $1/k^2$. That means that for any distribution the "real" value $E[f]$ is 93% sure to be within 4 σ_f's of the "computed" value $f[\{X\}]$; for gaussian distributions this probability is 99.9937%.

B Error propagation

Error propagation formula is straightforward to prove via Taylor expansion:

$$\sigma_f^2 = E[f^2] - E[f]^2 \approx E[(\overline{f} + \delta f + \delta^2 f)^2] - E[\overline{f} + \delta f + \delta^2 f]^2 \,, \tag{9}$$

with $\delta f = \sum_i (x_i - \overline{x}_i) \frac{\partial f}{\partial x_i}$ and $\delta f^2 = \frac{1}{2} \sum_{i,j} (x_i - \overline{x}_i) \frac{\partial^2 f}{\partial x_i \partial x_j} (x_j - \overline{x}_j)$, being the first and second order contributions to the fluctuations of f. The expectations values for the fluctuations are:

$$E[\delta f] = 0 \quad \text{and} \quad E[\delta^2 f] = \frac{1}{2} \sum_{i,j} \frac{\partial^2 f}{\partial x_i \partial x_j} \sigma_{ij}^2 \,, \tag{10}$$

where $\sigma_{ij}^2 \equiv E[(x_i - \overline{x}_i)(x_j - \overline{x}_j)]$ is the *covariance* of x_i and x_j. If we expand the expression above and keep only terms up to second order we get:

$$\sigma_f^2 \approx \overline{f}^2 + E[\delta f^2] + 2\overline{f} E[\delta^2 f] - \overline{f}^2 - 2\overline{f} E[\delta^2 f] = E[\delta f^2] = \sum_{ij} \frac{\partial f}{\partial x_i} \frac{\partial f}{\partial x_j} \sigma_{ij}^2 \,. \tag{11}$$

When the fluctuations of x_i are statistically independent, we have $\sigma_{ij}^2 = 0$ for $i \neq j$ and $\sigma_{ii}^2 = E[(x_i - \overline{x}_i)^2] = \sigma_i^2$. In this case the formula above reduces to

$$\sigma_f^2 = \sum_i \left(\frac{\partial f}{\partial x_i} \right)^2 \sigma_i^2 \,. \tag{12}$$

C Justification for the bootstrap method

Let us state the general setup for this kind of problem. Assume that the entire data set $\{X\}$ is composed of subsets X_i that are statistically independent and generated by a statistically similar process. In the case discussed in Section 2 X_i represents a pair (x_i, y_i).

We are usually provided with only one data set, but we could in principle repeat the measurements or run a new simulations and get a new data set, $\{X\}'$ of the same size. We can imagine repeating the process a very large number of times and we see that the data sets $\{X\}$ are distributed with probability $P[\{X\}]$. Note that this is the probability of generating an entire data set, not an individual data subset X_i. If each data subset is drawn from the same distribution, $p(X)$, in a statistically independent way, then the probability for a sample is

$$P[\{X_1, X_2, ..., X_N\}] = \prod_{i=1}^{N} p(X_i). \tag{13}$$

To compute the variance σ_f of a statistic f we would need to have access to the probability $P[\{X\}]$. This is not always feasible since generating $\{X\}$ might be very expensive or time consuming. One solution is to use $\{X\}$ to construct an approximation for p, the individual subset probability:

$$p_{\text{approx}}(X) = \frac{1}{N} \sum_i \delta(X - X_i). \tag{14}$$

This approximation is reasonable, especially when the number of subsets N in the dataset is large. Using this subset probability we construct an approximation for P by drawing N independent points from the distribution p_{approx},

$$P_{\text{approx}}[\{X\}] = \prod_i p_{\text{approx}}(X_i). \tag{15}$$

Practically, this is easily achieved by drawing subsets X_i from $\{X\}$ with equal probability. These subsets are drawn *with replacement*, i.e. each point has equal probability of being extracted even if it was extracted in a previous step. To compute the exact expectation value of the statistic f with respect to the probability distribution P_{approx}, $E'[f]$, we would need to sum over N^N distinct sequences that can be drawn from $\{X\}$. In most cases this is not feasible and an approximation is used based on $N_{\text{seq}} = 10^3 - 10^4$ sequences. The variance σ_f is then estimated using

$$\bar{f} = \frac{1}{N_{\text{seq}}} \sum_s f[\{X\}_s], \tag{16}$$

$$\sigma_f \approx \sqrt{\frac{\sum_s (f[\{X\}_s] - \bar{f})^2}{N_{\text{seq}}}}. \tag{17}$$

D standard deviation of the standard deviation

The *unbiased* sample variance of a set of points $x_1, \ldots, x_n$ is

$$s^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \overline{x})^2 \,. \tag{18}$$

If the x_i's are normally distributed, it is a fact that

$$\frac{(n-1)s^2}{\sigma^2} \sim \chi^2_{n-1} \,, \tag{19}$$

where σ^2 is the true variance. The χ^2_k distribution has probability density

$$p(x) = \frac{(1/2)^{k/2}}{\Gamma(k/2)} x^{k/2-1} e^{-x/2} \,. \tag{20}$$

Using this we can derive the expected value of s

$$\begin{aligned} E(s) &= \sqrt{\frac{\sigma^2}{n-1}} E\left(\sqrt{\frac{s^2(n-1)}{\sigma^2}} \right) \\ &= \sqrt{\frac{\sigma^2}{n-1}} \int_0^\infty \sqrt{x} \frac{(1/2)^{(n-1)/2}}{\Gamma((n-1)/2)} x^{((n-1)/2)-1} e^{-x/2} \, dx \end{aligned} \tag{21}$$

which follows from the definition of expected value and fact that $\sqrt{\frac{s^2(n-1)}{\sigma^2}}$ is the square root of a χ^2 distributed variable. The trick now is to rearrange terms so that the integrand becomes another χ^2 density:

$$\begin{aligned} E(s) &= \sqrt{\frac{\sigma^2}{n-1}} \int_0^\infty \frac{(1/2)^{(n-1)/2}}{\Gamma(\frac{n-1}{2})} x^{(n/2)-1} e^{-x/2}\, dx \\ &= \sqrt{\frac{\sigma^2}{n-1}} \cdot \frac{\Gamma(n/2)}{\Gamma(\frac{n-1}{2})} \int_0^\infty \frac{(1/2)^{(n-1)/2}}{\Gamma(n/2)} x^{(n/2)-1} e^{-x/2}\, dx \\ &= \sqrt{\frac{\sigma^2}{n-1}} \cdot \frac{\Gamma(n/2)}{\Gamma(\frac{n-1}{2})} \cdot \frac{(1/2)^{(n-1)/2}}{(1/2)^{n/2}} \underbrace{\int_0^\infty \frac{(1/2)^{n/2}}{\Gamma(n/2)} x^{(n/2)-1} e^{-x/2}\, dx}_{\chi^2_n \text{ density}} \end{aligned} \tag{22}$$

now we know the integrand the last line is equal to 1, since it is a χ^2_n density. Simplifying constants a bit gives

$$E(s) = \sigma \cdot \sqrt{\frac{2}{n-1}} \cdot \frac{\Gamma(n/2)}{\Gamma(\frac{n-1}{2})}. \tag{23}$$

From the basic properties of variance that

$$var(s) = E(s^2) - E(s)^2 \tag{24}$$

Since the sample variance is unbiased, we have $E(s^2) = \sigma^2$. Above we calculated $E(s)$ so we have the standard deviation

$$\mathrm{SD}(s) = \sqrt{E(s^2) - E(s)^2} = \sigma \sqrt{1 - \frac{2}{n-1} \cdot \left(\frac{\Gamma(n/2)}{\Gamma(\frac{n-1}{2})} \right)^2}. \tag{25}$$

Since our definition for `std` is not the unbiased version, we have $\texttt{std} = \sqrt{(n-1)/n}\, s$ so its standard deviation will be also multiplied by the same factor

$$\mathrm{SD}(\texttt{std}) = \sigma \sqrt{\frac{n-1}{n} - \frac{2}{n} \cdot \left(\frac{\Gamma(n/2)}{\Gamma(\frac{n-1}{2})} \right)^2}. \tag{26}$$