Bias Correction for Nonlinear Least Squares

Giacomo Petrillo, University of Pisa

June 25, 2019

Abstract

We correct the bias of the least squares estimator by approximating it to second order and find this performs slightly worse than a known least squares-specific correction which can be seen as a special case of ours. We illustrate clearly how to implement such corrections and we advocate for addition to generic least squares fitting programs by showing an example which may look "innocent" but has a significative bias. The algorithm is computationally light and requires only up to the second derivatives of the model function.

1 Introduction

Let $\mathbf{y} \in \mathbb{R}^N$ be a random variable with a probability distribution that depends on parameters $\boldsymbol{\theta} \in \mathbb{R}^K$, with covariance matrix

$$V_{ij} \equiv \mathrm{Cov}[y_i, y_j] \tag{1}$$

that does not depend on $\boldsymbol{\theta}$, and mean given by the function

$$\boldsymbol{\mu}(\boldsymbol{\theta}) \equiv E[\mathbf{y}|\boldsymbol{\theta}]. \tag{2}$$

The generalized least squares estimator for $\boldsymbol{\theta}$ is defined as the $\hat{\boldsymbol{\theta}}(\mathbf{y})$ that minimizes the quadratical form $Q(\mathbf{y}, \boldsymbol{\theta})$ given below:

$$Q(\mathbf{y}, \boldsymbol{\theta}) \equiv \frac{1}{2}(\mathbf{y} - \boldsymbol{\mu}(\boldsymbol{\theta}))^\top V^{-1}(\mathbf{y} - \boldsymbol{\mu}(\boldsymbol{\theta})), \tag{3}$$

$$\hat{\boldsymbol{\theta}}(\mathbf{y}) \equiv \arg\min_{\boldsymbol{\theta}} Q(\mathbf{y}, \boldsymbol{\theta}). \tag{4}$$

It holds that if $\boldsymbol{\mu}(\boldsymbol{\theta})$ is linear in $\boldsymbol{\theta}$, then $\hat{\boldsymbol{\theta}}(\mathbf{y})$ is unbiased (i.e. $E[\hat{\boldsymbol{\theta}}(\mathbf{y})|\boldsymbol{\theta}_0] = \boldsymbol{\theta}_0$) and efficient (i.e. has the minimum variance possible) [1]. In general this is not true, but in practice it works well even if $\boldsymbol{\mu}(\boldsymbol{\theta})$ is nonlinear. In this article we will derive a general way to correct

the bias of the estimator in the nonlinear case, i.e. find $\mathbf{B}(\mathbf{y})$ such that $E[\hat{\boldsymbol{\theta}}(\mathbf{y}) - \mathbf{B}(\mathbf{y})|\boldsymbol{\theta}_0]$ is closer to $\boldsymbol{\theta}_0$ than $E[\hat{\boldsymbol{\theta}}(\mathbf{y})|\boldsymbol{\theta}_0]$, and compare it to an already known procedure [2].

For convenience, we will fix $V_{ij} = \delta_{ij}$, i.e. all y_i are uncorrelated and with unitary variance, because otherwise $\mathbf{y}$ can be put in this form with the following standard transformation: using the Cholesky decomposition, V is written as $V = LL^\top$, where L is a lower triangular matrix. It is easily verified that the variables $\mathbf{y}' \equiv L^{-1}\mathbf{y}$ have covariance $\mathrm{Cov}[y'_i, y'_j] = \delta_{ij}$. So, (3) becomes just the sum of squares

$$Q(\mathbf{y}', \boldsymbol{\theta}) = \frac{1}{2}(\mathbf{y}' - \boldsymbol{\mu}'(\boldsymbol{\theta}))^\top(\mathbf{y}' - \boldsymbol{\mu}'(\boldsymbol{\theta})), \tag{5}$$

where $\boldsymbol{\mu}' \equiv L^{-1}\boldsymbol{\mu}$. In the following derivations, you can check that the transformation, being linear, does not affect the validity of the results.

2 Generic Second Order Bias Correction

In this section we derive a general bias correction procedure that will be shown later to apply to the case of the least squares estimator.

Let $\mathbf{x} \in \mathbb{R}^N$ be a random variable with mean $\boldsymbol{\mu}$ and $f : \mathbb{R}^N \to \mathbb{R}$ a real-valued function. Our goal is to correct the bias of $f(\mathbf{x})$ considered as estimator of $f(\boldsymbol{\mu})$.

To this end, we expand f in Taylor series around $\boldsymbol{\mu}$ up to second order

$$\begin{aligned} f(\mathbf{x}) \approx\ & f(\boldsymbol{\mu}) \\ & + \sum_i \frac{\partial f}{\partial x_i}(\boldsymbol{\mu})(x_i - \mu_i) \\ & + \frac{1}{2}\sum_{ij} \frac{\partial^2 f}{\partial x_i \partial x_j}(\boldsymbol{\mu})(x_i - \mu_i)(x_j - \mu_j), \end{aligned} \tag{6}$$

and compute the expected value of $f(\mathbf{x})$ with this expansion, obtaining the well-known formula

$$E[f(\mathbf{x})] \approx f(\boldsymbol{\mu}) + \frac{1}{2}\sum_{ij} \frac{\partial^2 f}{\partial x_i \partial x_j}(\boldsymbol{\mu})\,\mathrm{Cov}[x_i, x_j]. \tag{7}$$

We now just subtract the quadratic term of (7) from $f(\mathbf{x})$, but evaluating the second derivative of f in $\mathbf{x}$ instead of $\boldsymbol{\mu}$ because the estimator must be a function of "data" only:

$$f_2(\mathbf{x}) \equiv f(\mathbf{x}) - \frac{1}{2}\sum_{ij} \frac{\partial^2 f}{\partial x_i \partial x_j}(\mathbf{x})\,\mathrm{Cov}[x_i, x_j]. \tag{8}$$

To evaluate the bias of (8), we expand $\partial_{ij} f$ in Taylor series up to first order and f up to third order around $\boldsymbol{\mu}$, obtaining:

$$E[f_2(\mathbf{x})] \approx f(\boldsymbol{\mu}) + \frac{1}{6} \sum_{ijk} \frac{\partial^3 f}{\partial x_i \partial x_j \partial x_k}(\boldsymbol{\mu}) C_{ijk}, \tag{9}$$

$$\text{where } C_{ijk} \equiv E[(x_i - \mu_i)(x_j - \mu_j)(x_k - \mu_k)], \tag{10}$$

so $f_2(\mathbf{x})$ is an unbiased estimator of $f(\boldsymbol{\mu})$ up to second order of f. If in (8) we replace $\mathrm{Cov}[x_i, x_j]$ with an unbiased estimator of it, the result still holds apart eventually from the next order term shown in (9).

3 Bias Correction of the Least Squares Estimator

We now apply the formalism of section 2 to the least squares estimator defined in section 1. Let us copy here the relevant definitions from section 1:

- $\mathbf{y} \in \mathbb{R}^N$ has mean $\boldsymbol{\mu}(\boldsymbol{\theta})$, with $\boldsymbol{\theta} \in \mathbb{R}^K$, and covariance matrix $\mathrm{Cov}[y_i, y_j] = \delta_{ij}$;
- $Q(\mathbf{y}, \boldsymbol{\theta}) \equiv \frac{1}{2}(\mathbf{y} - \boldsymbol{\mu}(\boldsymbol{\theta}))^\top (\mathbf{y} - \boldsymbol{\mu}(\boldsymbol{\theta}))$, $\quad \hat{\boldsymbol{\theta}}(\mathbf{y}) \equiv \arg\min_{\boldsymbol{\theta}} Q(\mathbf{y}, \boldsymbol{\theta})$.

We observe that $\hat{\boldsymbol{\theta}}$ has the property that $\hat{\boldsymbol{\theta}}(\boldsymbol{\mu}(\boldsymbol{\theta}_0)) = \boldsymbol{\theta}_0$ because, assuming Q has only one minimum (otherwise $\hat{\boldsymbol{\theta}}$ would not be well defined), the point $Q(\boldsymbol{\mu}(\boldsymbol{\theta}_0), \boldsymbol{\theta}_0) = 0$ must be the minimum since $Q \geq 0$. This means that we are in the case studied in section 2 with $\hat{\boldsymbol{\theta}}$ and $\mathbf{y}$ taking the role of f and $\mathbf{x}$ respectively, so we apply formula (8) obtaining

$$\hat{\boldsymbol{\theta}}_2(\mathbf{y}) \equiv \hat{\boldsymbol{\theta}}(\mathbf{y}) - \mathbf{B}(\mathbf{y}), \tag{11}$$

$$\text{where } \mathbf{B}(\mathbf{y}) \equiv \frac{1}{2} \sum_i \frac{\partial^2 \hat{\boldsymbol{\theta}}}{\partial y_i^2}(\mathbf{y}). \tag{12}$$

The point now is computing $\partial^2 \hat{\boldsymbol{\theta}} / \partial y_i^2$. Let us use latin indices for the $\mathbf{y}$ vector and greek indices for the $\boldsymbol{\theta}$ vector. At the minimum, the gradient of Q is zero, so we have

$$0 = G_\alpha(\mathbf{y}) \equiv g_\alpha(\mathbf{y}, \hat{\boldsymbol{\theta}}(\mathbf{y})), \tag{13}$$

$$\text{where } g_\alpha(\mathbf{y}, \boldsymbol{\theta}) \equiv \frac{\partial Q}{\partial \theta_\alpha}(\mathbf{y}, \boldsymbol{\theta}). \tag{14}$$

We compute the first and second derivatives of G_α:

$$0 = \frac{\partial G_\alpha}{\partial y_i} = \frac{\partial g_\alpha}{\partial y_i} + \sum_\beta \frac{\partial g_\alpha}{\partial \theta_\beta}\frac{\partial \hat{\theta}_\beta}{\partial y_i}, \tag{15}$$

$$\begin{aligned} 0 = \frac{\partial^2 G_\alpha}{\partial y_i \partial y_j} &= \frac{\partial^2 g_\alpha}{\partial y_i \partial y_j} \\ &+ \sum_\beta \left(\frac{\partial^2 g_\alpha}{\partial y_i \partial \theta_\beta}\frac{\partial \hat{\theta}_\beta}{\partial y_j} + \frac{\partial^2 g_\alpha}{\partial y_j \partial \theta_\beta}\frac{\partial \hat{\theta}_\beta}{\partial y_i} \right) \\ &+ \sum_{\beta\gamma} \frac{\partial^2 g_\alpha}{\partial \theta_\beta \partial \theta_\gamma}\frac{\partial \hat{\theta}_\beta}{\partial y_i}\frac{\partial \hat{\theta}_\gamma}{\partial y_j} \\ &+ \sum_\beta \frac{\partial g_\alpha}{\partial \theta_\beta}\frac{\partial^2 \hat{\theta}_\beta}{\partial y_i \partial y_j}, \end{aligned} \tag{16}$$

where we implicitly intend that all derivatives of g_α are evaluated at $(\mathbf{y}, \hat{\boldsymbol{\theta}}(\mathbf{y}))$ and all derivatives of $\hat{\boldsymbol{\theta}}$ at $\mathbf{y}$. Defining

$$A_{\alpha\beta} \equiv \frac{\partial g_\alpha}{\partial \theta_\beta}, \tag{17}$$

$$B_\alpha^{(i)} \equiv -\frac{\partial g_\alpha}{\partial y_i}, \tag{18}$$

$$\begin{aligned} C_\alpha^{(ij)} \equiv &-\frac{\partial^2 g_\alpha}{\partial y_i \partial y_j} - \sum_\beta \left(\frac{\partial^2 g_\alpha}{\partial y_i \partial \theta_\beta}\frac{\partial \hat{\theta}_\beta}{\partial y_j} + \frac{\partial^2 g_\alpha}{\partial y_j \partial \theta_\beta}\frac{\partial \hat{\theta}_\beta}{\partial y_i} \right) \\ &- \sum_{\beta\gamma} \frac{\partial^2 g_\alpha}{\partial \theta_\beta \partial \theta_\gamma}\frac{\partial \hat{\theta}_\beta}{\partial y_i}\frac{\partial \hat{\theta}_\gamma}{\partial y_j}, \end{aligned} \tag{19}$$

we can rewrite equations (15) and (16) as

$$A\partial_i \hat{\boldsymbol{\theta}} = B^{(i)}, \tag{20}$$

$$A\partial_{ij} \hat{\boldsymbol{\theta}} = C^{(ij)}. \tag{21}$$

So, to compute $\partial_{ij}\hat{\boldsymbol{\theta}}$, we have to:

1. compute A (17) and B (18) at the minimum with g_α defined in (14);
2. solve the linear system (20) to find $\partial_i \hat{\boldsymbol{\theta}}$ for all i;
3. compute C (19) using $\partial_i \hat{\boldsymbol{\theta}}$ from the previous step;
4. solve the linear system (21) for all i, j (actually, we just need the diagonal $i = j$).

Note that the matrix A is always the same for all i, j so solving the linear systems is computationally light. We now report explicitly the derivatives of g_α:

$$\frac{\partial g_\alpha}{\partial \theta_\beta} = \sum_i \left(\frac{\partial \mu_i}{\partial \theta_\alpha} \frac{\partial \mu_i}{\partial \theta_\beta} - (y_i - \mu_i) \frac{\partial^2 \mu_i}{\partial \theta_\alpha \partial \theta_\beta} \right), \tag{22}$$

$$\frac{\partial g_\alpha}{\partial y_i} = -\frac{\partial \mu_i}{\partial \theta_\alpha}, \tag{23}$$

$$\frac{\partial^2 g_\alpha}{\partial y_i \partial y_j} = 0, \tag{24}$$

$$\frac{\partial^2 g_\alpha}{\partial y_i \partial \theta_\beta} = -\frac{\partial^2 \mu_i}{\partial \theta_\alpha \partial \theta_\beta}, \tag{25}$$

$$\begin{aligned} \frac{\partial^2 g_\alpha}{\partial \theta_\beta \partial \theta_\gamma} = \sum_i \Bigg(& \frac{\partial^2 \mu_i}{\partial \theta_\alpha \partial \theta_\beta} \frac{\partial \mu_i}{\partial \theta_\gamma} + \frac{\partial^2 \mu_i}{\partial \theta_\beta \partial \theta_\gamma} \frac{\partial \mu_i}{\partial \theta_\alpha} + \frac{\partial^2 \mu_i}{\partial \theta_\gamma \partial \theta_\alpha} \frac{\partial \mu_i}{\partial \theta_\beta} \\ & - (y_i - \mu_i) \frac{\partial^3 \mu_i}{\partial \theta_\alpha \partial \theta_\beta \partial \theta_\gamma} \Bigg). \end{aligned} \tag{26}$$

Since we are dealing with power series, we expect that sometimes the second order bias correction (12) may yield a very large value that completely spoils the result. However, it is easy to check if the correction is too large by comparing it to the covariance matrix of $\hat{\boldsymbol{\theta}}(\mathbf{y})$ as estimated by the standard formula

$$\mathrm{Cov}[\hat{\theta}_\alpha(\mathbf{y}), \hat{\theta}_\beta(\mathbf{y})] \approx (H(\hat{\boldsymbol{\theta}}(\mathbf{y}))^{-1})_{\alpha\beta}, \tag{27}$$

$$\text{where } H_{\alpha\beta}(\boldsymbol{\theta}) \equiv \sum_i \frac{\partial \mu_i}{\partial \theta_\alpha}(\boldsymbol{\theta}) \frac{\partial \mu_i}{\partial \theta_\beta}(\boldsymbol{\theta}). \tag{28}$$

(If $\boldsymbol{\mu}(\boldsymbol{\theta})$ is linear, H^{-1} is the exact covariance matrix of $\hat{\boldsymbol{\theta}}(\mathbf{y})$.) In particular, the following quantity shall be checked to be less than a fixed threshold:

$$\mathbf{B}(\mathbf{y})^\top H(\hat{\boldsymbol{\theta}}(\mathbf{y})) \mathbf{B}(\mathbf{y}). \tag{29}$$

The reason for using $H(\hat{\boldsymbol{\theta}}(\mathbf{y}))$ is that it is stable. What we need here is a measure of scale with low variance, not an unbiased estimate of the actual covariance matrix of our bias-corrected estimator $\hat{\boldsymbol{\theta}}_2$. Formula (29) is quadratical so the threshold should be thought of as "the square of a number of standard deviations".

When it happens that (29) is larger than the chosen threshold, the only option is to not apply the correction. This will implicitly change the estimator we are using, because the choice is based on the observed data, but the effect should be small if we are prudent in the threshold we choose (e.g. at least $3^2 = 9$).

Another caveat is that the variance of $\hat{\boldsymbol{\theta}}_2$ will be larger than the variance of $\hat{\boldsymbol{\theta}}$. Computing $E[\hat{\theta}_{2\alpha}\hat{\theta}_{2\beta}]$ expanding f and its derivatives to second order like in section 2, it can be shown that

$$\begin{aligned} V_{\alpha\beta}(\mathbf{y}) \equiv\ & \partial_i\hat{\theta}_\alpha\partial_i\hat{\theta}_\beta \\ & + \frac{1}{2}(\partial_i\hat{\theta}_\alpha\partial_{jk}\hat{\theta}_\beta + \partial_i\hat{\theta}_\beta\partial_{jk}\hat{\theta}_\alpha)V_{ijk} \\ & + \frac{1}{4}\partial_{ij}\hat{\theta}_\alpha\partial_{kl}\hat{\theta}_\beta(V_{ijkl} - V_{ij}V_{kl} - 4V_{ik}V_{jl}), \end{aligned} \tag{30}$$

where the derivatives of $\hat{\theta}$ are evaluated at $\mathbf{y}$, the latin indices are summed over,

$$V_{ijk} \equiv E[(y_i - \mu_i)(y_j - \mu_j)(y_k - \mu_k)] \tag{31}$$

$$\text{and } V_{ijkl} \equiv E[(y_i - \mu_i)(y_j - \mu_j)(y_k - \mu_k)(y_k - \mu_l)], \tag{32}$$

is an unbiased (up to second order of $\hat{\boldsymbol{\theta}}$) estimator of $\mathrm{Cov}[\hat{\theta}_{2\alpha}, \hat{\theta}_{2\beta}]$. To use formula (30) in practice it is necessary to make assumptions about V_{ijk} and V_{ijkl}. A simpler formula, but still different from the H^{-1} in (27), is obtained with first order error propagation, i.e. dropping all the terms with second derivatives in (30):

$$V^{(1)}_{\alpha\beta}(\mathbf{y}) \equiv \sum_i \frac{\partial\hat{\theta}_\alpha}{\partial y_i}\frac{\partial\hat{\theta}_\beta}{\partial y_i}, \tag{33}$$

which is readily available since $\partial_i\hat{\boldsymbol{\theta}}$ has already been computed.

If we remove the terms with $(y_i - \mu_i)$ in (22) and (26), i.e. if $\mathbf{y} = \boldsymbol{\mu}(\hat{\boldsymbol{\theta}}(\mathbf{y}))$, the complete formula for $\mathbf{B}(\mathbf{y})$ simplifies a bit and can be written concisely as

$$\mathbf{B}(\mathbf{y}) \underset{\text{if } \mathbf{y}=\boldsymbol{\mu}(\hat{\boldsymbol{\theta}}(\mathbf{y}))}{=} -\frac{1}{2}\sum_{i\beta\gamma\delta}(H^{-1})_{\alpha\beta}\frac{\partial\mu_i}{\partial\theta_\beta}\frac{\partial^2\mu_i}{\partial\theta_\gamma\partial\theta_\delta}(H^{-1})_{\gamma\delta}, \tag{34}$$

where H is defined in (28). This is exactly the same formula obtained by [2], equation 2.20. You can convince yourself that $\mathbf{B}(\mathbf{y})$ in (12) is not in general expressed by (34) by noting that in (26) there is a third derivative of $\boldsymbol{\mu}(\boldsymbol{\theta})$, while in (34) there are only up to second derivatives of $\boldsymbol{\mu}$. Effectively, [2] derives its result computing the bias and then replacing the true value $\boldsymbol{\theta}$ with $\hat{\boldsymbol{\theta}}(\mathbf{y})$, so the terms $y_i - \mu_i$ disappear in taking expectations.

4 Example

We apply the method to an example that we know to suffer from a significative bias, which is fitting an oscillating curve with errors on the "explanatory variable".

We have measurements $\mathbf{x} \in \mathbb{R}^M$, $\mathbf{t} \in \mathbb{R}^M$ which are unbiased estimators of the true values $\bar{\mathbf{x}}$, $\bar{\mathbf{t}}$, constrained by the relation $\bar{x}_i = p_0 \cos(\bar{t}_i/p_1)$. Formulating everything with our precedent notations, we have

$$\mathbf{y} = (t_1, \dots, t_M, x_1, \dots, x_M), \tag{35}$$

$$\boldsymbol{\theta} = (p_0, p_1, \bar{t}_1, \dots, \bar{t}_M), \tag{36}$$

$$\boldsymbol{\mu}(\boldsymbol{\theta}) = (\bar{t}_1, \dots, \bar{t}_M, p_0 \cos(\bar{t}_1/p_1), \dots, p_0 \cos(\bar{t}_M/p_1)). \tag{37}$$

For simplicity we set $\mathrm{Cov}[y_i, y_j] = \delta_{ij}$.

We have written an inefficient but quick implementation, reported in Appendix A, that checks the properties of the estimator using Monte Carlo. The results are reported in detail in Figure 1. We also check for comparison the bias correction from [2] which is given by using (34) even if $\mathbf{y} \neq \boldsymbol{\mu}(\hat{\boldsymbol{\theta}}(\mathbf{y}))$.

Comment on Figure 1: the two corrections have, within Monte Carlo sample uncertainty, the same mean (id est they correct the bias the same), but the correction obtained by approximating the estimator as a paraboloid (12) has higher variance than (34). The bias is significative, about half the standard deviation, and gets reduced to about one tenth by the corrections. Formula (33) for the variance of $\hat{\boldsymbol{\theta}}_2$ has surely higher mean, but for parameter p_1 it has a quite long tail, so the standard formula (27) is probably better in general.

5 Conclusion

We have shown in detail how to correct the bias of the least squares estimator by using a parabolic approximation of the estimator. Let us outline the complete algorithm:

1. reformulate the problem using the transformation given at the end of section 1;
2. mimimize the sum of squares by any standard algorithm;
3. use equations 22 to 26 to compute the terms in (17) to (19) and solve the linear systems (20) and (21);
4. compute the bias correction using (12);
5. check that (29) is less than a fixed threshold, if it is not, do not use the correction.

Since in the example (Figure 1) the correction performs slightly worse than the simpler to compute (34) already known in the literature, we actually advise to use (34). Nevertheless, let us comment for completeness on our algorithm. Up to the third derivatives of $\boldsymbol{\mu}(\boldsymbol{\theta})$

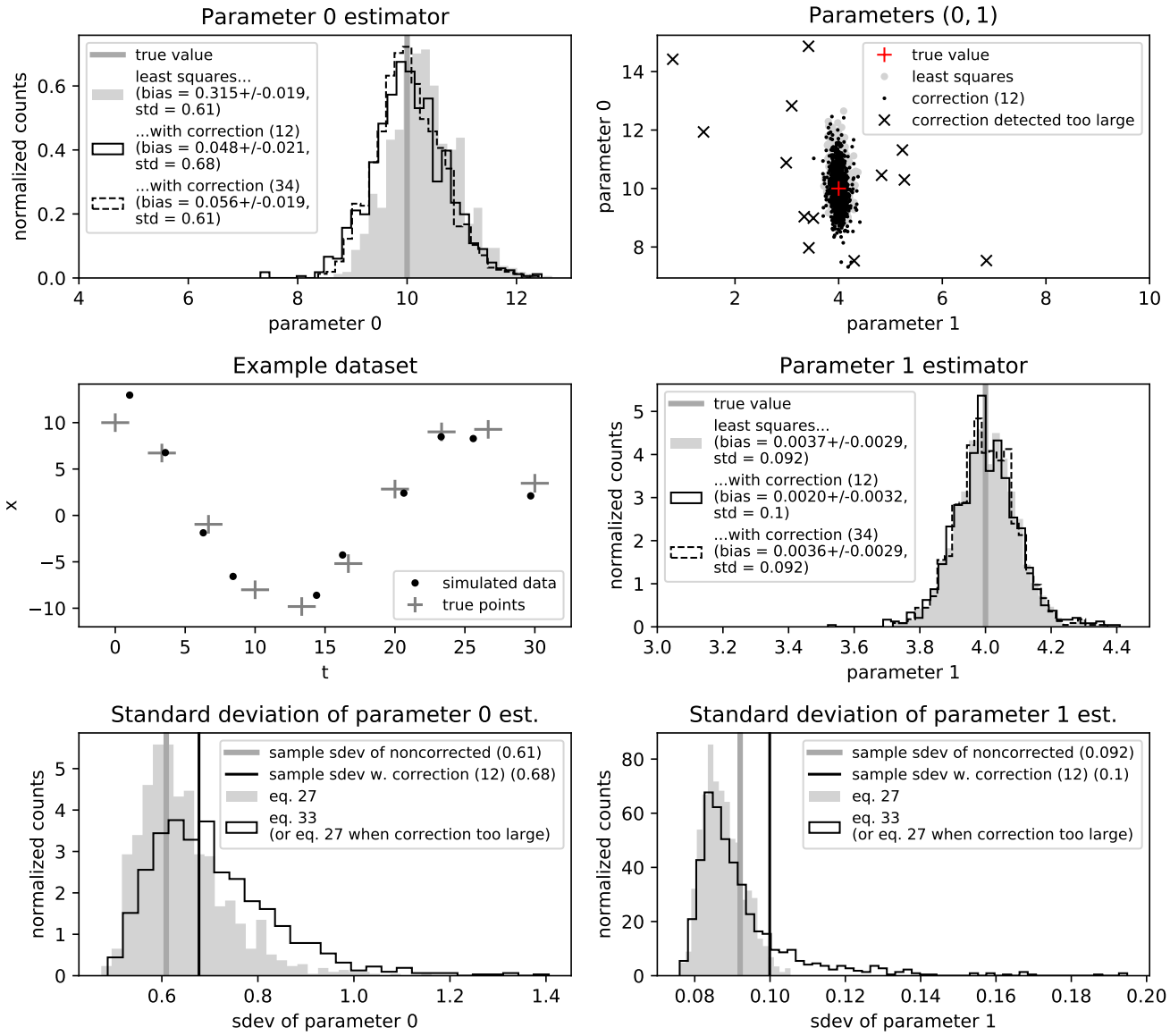

Figure 1: Monte Carlo study of the bias correction to the case defined in section 4, i.e. fitting $x = p_0 \cos(t/p_1)$ with errors on both x and t. The data is simulated 1000 times. The true values of the parameters are fixed to $p_0 = 10$, $p_1 = 4$. The plots with histograms on the first two rows show the distributions of the least squares estimator (filled gray), of the estimator corrected with (12) (solid line), and of the estimator corrected with (34) (dashed line). The correction (12) is applied only when (29) is less than 25. In the scatter plot, differently from the histograms, the correction is always applied and marked with a "$\times$" when it is not applied in the histograms.

are required, but with modern computational tools the derivatives can be evaluated automatically. The computational overhead should be comparable to just minimizing the sum of squares. If the number of parameters K is large, computing the third derivatives of $\boldsymbol{\mu}$, which are $NK^3/6 + O(K^2)$, may become too expensive, but tipically when the number of parameters is large there is some sparsity in the problem. The method can be in principle extended to successive orders, but at order λ would require to evaluate $O(K^{1+\lambda})$ derivatives of $\boldsymbol{\mu}$, and it would require knowledge of higher moments of $\mathbf{y}$.

If one does not have the covariance matrix V of $\mathbf{y}$ but an estimate $\hat{V}$ of it, $\hat{V}$ must be unbiased, as is noted at the end of section 2. This requirement is new compared to linear least squares, where if V is whatever random matrix with the only constraint of being independent from $\mathbf{y}$, the least squares estimator is still unbiased (but not minimum variance).

The correction, although long known, to the knowledge of the author is not implemented in commonly used least squares fitting programs. We have provided an example where the bias is half the standard deviation, and can be made higher by increasing the uncertainties. Since the casual user may not have the time (or statistical knowledge) to implement herself a bias correction, we think that fitting programs should provide the implementation and the option to use it, along with a brief explanation of what is the bias.

Finally, we consider a further exploitation. Having computed the second derivatives of the estimator with respect to the data, second order error propagation can be used to get a wealth of information about the shape of the distribution of the estimator, and correlation with the data. For example, if two fits are performed on overlapping datasets, the overlap would be taken into account transparently and correlation of the estimates would be known. This has already been excellently implemented to first order in [4], but using second order would allow to compute higher order correlation functions and draw a nontrivial distribution for any multidimensional estimate using e.g. a maximum entropy estimate, which can feasibly be implemented up to four dimensions [3] (and, of course, to correct the bias). The author is currently writing a program with this final goal in mind [5].

References

[1] Aitken, A. C. (1934). *On Least-squares and Linear Combinations of Observations*. Proceedings of the Royal Society of Edinburgh. 55: 42–48. https://doi.org/10.1017/S0370164600014346.

[2] Box, M. J. (1971). *Bias in Nonlinear Estimation.* Journal of the Royal Statistical Society. Series B (Methodological), Vol. 33, No. 2, pp. 171–201. `https://www.jstor.org/stable/2985002`.

[3] Abramov, R. V. (2009) *The Multidimensional Maximum Entropy Moment Problem: a Review on Numerical Methods.* Commun. Math. Sci. Vol. 8, No. 2, pp. 377–392. `https://projecteuclid.org/euclid.cms/1274816887`.

[4] Lepage, G. P. (2008–2019). *`lsqfit`: Utilities for nonlinear least-squares fits.* 10.5281/zenodo.2613676. `https://github.com/gplepage/lsqfit`.

[5] Petrillo, G. (2019). *`uncertainties-cpp`: C++ header library for first- and second-order uncertainty propagation.* `https://github.com/Gattocrucco/uncertainties-cpp`.

A Computer program

The following Python scripts were tested with Python 3.6.4, `numpy` 1.16.4, `scipy` 1.2.1, `matplotlib` 3.0.2, `autograd` 1.2, `progressbar2` 3.38.0, `uncertainties` 3.0.3. They implement the test described in section 4, but adapting them to a different model should be quick. Note that this implementation, due to the use of `autograd` from the ground up, is so inefficient that running a Monte Carlo without bias correction would be faster than a single corrected fit. However, it has the advantage of not having many chances of error.

```
1  # file fit.py
2
3  from scipy import optimize, linalg
4  import autograd
5  from autograd import numpy as np
6  from numpy.lib import format as nplf
7  import progressbar
8
9  M = 1000 # number of monte carlo
10 N = 2 # number of parameters
11 def mu(x, p):
12     return p[0] * np.cos(x / p[1])
13 true_x = np.linspace(0, 30, 10)
14 true_par = np.array([10, 4])
15
16 ##################################
17
```

```
18 table = nplf.open_memmap('fit.npy', mode='w+', shape=(M,), dtype=[
19     ('success', bool),
20     ('estimate', float, N),
21     ('bias', float, N),
22     ('standard_bias', float, N),
23     ('cov', float, (N, N)),
24     ('standard_cov', float, (N, N)),
25     ('data_y', float, len(true_x)),
26     ('data_x', float, len(true_x)),
27     ('complete_estimate', float, N + len(true_x)),
28     ('complete_bias', float, N + len(true_x)),
29     ('complete_cov', float, (N + len(true_x), N + len(true_x)))
30 ])
31 table['success'] = False
32
33 def res(p, data):
34     par = p[:N]
35     x = p[N:]
36     data_x = data[:len(data) // 2]
37     data_y = data[len(data) // 2:]
38     return np.concatenate([data_y - mu(x, par), data_x - x])
39
40 jac = autograd.jacobian(res, 0)
41
42 def Q(p, data):
43     r = res(p, data)
44     return np.sum(r ** 2)
45
46 f = autograd.jacobian(Q, 0)
47 dfdy = autograd.jacobian(f, 1)
48 dfdp = autograd.jacobian(f, 0)
49 dfdpdy = autograd.jacobian(dfdp, 1)
50 dfdpdp = autograd.jacobian(dfdp, 0)
51
52 true_y = mu(true_x, true_par)
53
54 np.savez(
55     'fit-info.npz',
56     true_x=true_x,
57     true_par=true_par,
58     true_y=true_y
59 )
60
```

```
61  def least_squares_cov(result):
62      _, s, VT = linalg.svd(result.jac, full_matrices=False)
63      threshold = np.finfo(float).eps * max(result.jac.shape) * s[0]
64      s = s[s > threshold]
65      VT = VT[:s.size]
66      return np.dot(VT.T / s**2, VT)
67
68  def compute_bias_cov(result, data):
69      dfdy_ = dfdy(result.x, data)
70      dfdp_ = dfdp(result.x, data)
71      dfdpdy_ = dfdpdy(result.x, data)
72      dfdpdp_ = dfdpdp(result.x, data)
73
74      grad = np.linalg.solve(dfdp_, -dfdy_)
75      assert(grad.shape == (N + len(true_x), 2 * len(true_x)))
76
77      cov = np.einsum('ai,bi->ab', grad, grad)
78      assert(np.allclose(cov, cov.T))
79
80      B = (
81          - np.einsum('abi,bj->aij', dfdpdy_, grad)
82          - np.einsum('abi,bj->aji', dfdpdy_, grad)
83          - np.einsum('abg,bi,gj->aij', dfdpdp_, grad, grad)
84      )
85      assert(B.shape == (N + len(true_x), 2 * len(true_x), 2 * len(true_x)))
86      B_ = B.reshape(N + len(true_x), 4 * len(true_x) * len(true_x))
87
88      hess = np.linalg.solve(dfdp_, B_)
89      hess = hess.reshape(N + len(true_x), 2 * len(true_x), 2 * len(true_x))
90      assert(np.allclose(hess, np.einsum('aji', hess)))
91
92      return 1/2 * np.einsum('aii', hess), cov
93
94  for i in progressbar.progressbar(range(M)):
95      data_x = true_x + np.random.randn(len(true_x))
96      data_y = true_y + np.random.randn(len(true_x))
97      data = np.concatenate([data_x, data_y])
98
99      p0 = np.concatenate([true_par, true_x])
100     result = optimize.least_squares(res, p0, jac=jac, args=(data,))
101     if not result.success:
102         print(f'minimization␣failed␣for␣i␣=␣{i}')
103         continue
```

```
104
105         bias, cov = compute_bias_cov(result, data)
106         zero_residual_data = np.concatenate([
107             result.x[N:],
108             mu(result.x[N:], result.x[:N])
109         ])
110         standard_bias, _ = compute_bias_cov(result, zero_residual_data)
111
112         table[i]['estimate'] = result.x[:N]
113         table[i]['bias'] = bias[:N]
114         table[i]['standard_bias'] = standard_bias[:N]
115         table[i]['cov'] = cov[:N, :N]
116         table[i]['data_y'] = data_y
117         table[i]['data_x'] = data_x
118         table[i]['complete_estimate'] = result.x
119         table[i]['complete_bias'] = bias
120         table[i]['complete_cov'] = cov
121         table[i]['standard_cov'] = least_squares_cov(result)[:N, :N]
122         table[i]['success'] = result.success
123
124         table.flush()
125
126 del table
```

```
1  # file fit-plot.py
2  # you can run this script while fit.py is running to see the partial results
3
4  from matplotlib import pyplot as plt
5  import uncertainties
6  import numpy as np
7
8  SIGMA_FACTOR = 5
9
10 table = np.load('fit.npy', mmap_mode='r')
11 table = table[table['success']]
12 N = len(table[0]['estimate'])
13
14 info = np.load('fit-info.npz')
15 true_par = info['true_par']
16
17 fig = plt.figure('fit')
18 fig.clf()
19 axs = fig.subplots(N + 1, N)
20
```

```
21  def ms(s):
22      return uncertainties.ufloat(np.mean(s), np.std(s) / np.sqrt(len(s)))
23
24  estimates = table['estimate'].T
25  biases = table['bias'].T
26  sbias = table['standard_bias'].T
27  covs = table['cov'].T
28  scovs = table['standard_cov'].T
29  sigmas = np.sqrt(np.einsum('iim->im', covs))
30  ssigmas = np.sqrt(np.einsum('iim->im', scovs))
31
32  ok = np.einsum(
33      'ua,uab,ub->u',
34      table['bias'], np.linalg.inv(table['standard_cov']), table['bias']
35  ) < SIGMA_FACTOR ** 2
36
37  for i in range(N):
38      ax = axs[i][i]
39
40      noncorr = estimates[i]
41      corr = estimates[i] - np.where(ok, biases[i], 0)
42      stdcorr = estimates[i] - sbias[i]
43      ax.hist(
44          noncorr, bins='auto', histtype='stepfilled',
45          label='least␣squares...\n(bias␣=␣{},\nstd␣=␣{:.2g})'.format(
46              ms(noncorr - true_par[i]),
47              np.std(noncorr)
48          ),
49          color='lightgray', zorder=0, density=True
50      )
51      ax.hist(
52          corr, bins='auto', histtype='step',
53          label='...with␣correction␣(12)\n(bias␣=␣{},\nstd␣=␣{:.2g})'.format(
54              ms(corr - true_par[i]),
55              np.std(corr)
56          ),
57          color='black', zorder=2, density=True
58      )
59      ax.hist(
60          stdcorr, bins='auto', histtype='step',
61          label='...with␣correction␣(34)\n(bias␣=␣{},\nstd␣=␣{:.2g})'.format(
62              ms(stdcorr - true_par[i]),
63              np.std(stdcorr)
```

```
64          ),
65          linestyle='--', color='black', zorder=2, density=True
66      )
67      ax.plot(
68          2 * [true_par[i]], ax.get_ylim(),
69          scaley=False, label='true value', color='darkgray', linewidth=3,
70          zorder=0.5
71      )
72      ax.legend(loc='best', fontsize='small')
73      ax.set_title(f'Parameter {i} estimator')
74      ax.set_xlabel(f'parameter {i}')
75      ax.set_ylabel('normalized counts')
76
77      # now we plot histogram of estimated standard deviation
78      ax = axs[N][i]
79
80      sstd = np.std(corr)
81      sstdnc = np.std(noncorr)
82
83      ax.hist(
84          ssigmas[i],
85          bins='auto', histtype='stepfilled', density=True,
86          color='lightgray', zorder=0,
87          label='eq. 27'
88      )
89      ax.hist(
90          np.where(ok, sigmas[i], ssigmas[i]),
91          bins='auto', histtype='step', density=True,
92          color='black', linestyle='-', zorder=1,
93          label='eq. 33\n(or eq. 27 when correction too large)'
94      )
95      ax.plot(
96          2 * [sstdnc], ax.get_ylim(),
97          scaley=False, color='darkgray', linewidth=3, zorder=0.5,
98          label=f'sample sdev of noncorrected ({sstdnc:.2g})'
99      )
100     ax.plot(
101         2 * [sstd], ax.get_ylim(),
102         scaley=False, color='black', zorder=1.5, linestyle='-',
103         label=f'sample sdev w. correction (12) ({sstd:.2g})'
104     )
105     ax.legend(loc='best', fontsize='small')
106     ax.set_title(f'Standard deviation of parameter {i} est.')
```

```
107         ax.set_xlabel(f'sdev␣of␣parameter␣{i}')
108         ax.set_ylabel('normalized␣counts')
109
110     for i in range(N):
111         for j in range(i + 1, N):
112             ax = axs[i][j]
113
114             ax.plot(
115                 true_par[j], true_par[i],
116                 marker='+', linestyle='', color='red', markersize=8,
117                 zorder=5, label='true␣value'
118             )
119             ax.plot(
120                 estimates[j], estimates[i],
121                 marker='.', markersize=6, color='lightgray',
122                 linestyle='', label='least␣squares'
123             )
124             ax.plot(
125                 (estimates[j] − biases[j])[ok], (estimates[i] − biases[i])[ok],
126                 marker='.', markersize=2, color='black',
127                 linestyle='', label='correction␣(12)'
128             )
129             ax.plot(
130                 (estimates[j] − biases[j])[~ok], (estimates[i] − biases[i])[~ok],
131                 marker='x', markersize=6, color='black',
132                 linestyle='', label='correction␣detected␣too␣large'
133             )
134             ax.legend(loc='best', fontsize='small')
135             ax.set_title(f'Parameters␣$(0,␣1)$')
136             ax.set_xlabel(f'parameter␣{j}')
137             ax.set_ylabel(f'parameter␣{i}')
138
139     for i in range(N):
140         for j in range(i):
141             ax = axs[i][j]
142
143             k = j ∗ N + i
144             ax.errorbar(
145                 info['true_x'], info['true_y'],
146                 xerr=1, yerr=1,
147                 linestyle='', marker='', color='gray',
148                 label='true␣points'
149             )
```

```
150             ax.plot(
151                 table[k]['data_x'], table[k]['data_y'],
152                 linestyle='', marker='.', color='black', label='simulated␣data'
153             )
154             ax.legend(loc='best', fontsize='small')
155             ax.set_title('Example␣dataset')
156             ax.set_xlabel('t')
157             ax.set_ylabel('x')
158
159 fig.tight_layout()
160 fig.show()
```