A Python Code For Maximum Likelihood Estimation Of The Location And Scale Parameters Of The Truncated Normal Distribution

Melih Yılmaz ÖGÜTCEN
Informatics Institute
Istanbul Technical University
Istanbul, Turkey
ogutcen20@itu.edu.tr

Mehmet KOCATÜRK
Department of Biomedical Engineering
Istanbul Medipol University
Istanbul, Turkey
mkocaturk@medipol.edu.tr

Murat OKATAN
Informatics Institute
Istanbul Technical University
Istanbul, Turkey
okatan@itu.edu.tr

***Abstract*— Extracellular neural recordings obtained from chronically implanted microelectrode arrays are widely used in behavioral neurophysiology and invasive brain-machine interfaces. After the raw recordings are band-pass filtered within a frequency band suitable for spike detection, spikes are often detected by amplitude thresholding. Developing principled methods for computing amplitude thresholds is an active research area. 'Truncation thresholds' are a pair of amplitude thresholds that are computed using a recently proposed algorithm. As part of an effort that aims to integrate this algorithm into a real-time data acquisition and spike detection system, here we present a Python code for maximum likelihood estimation of the location and scale parameters of the truncated Normal distribution, which is one of the steps involved in the computation of truncation thresholds.**

Keywords*— *truncated Normal distribution; maximum likelihood estimation; python code; truncation thresholds

I. INTRODUCTION

Extracellular neural recordings provide important information about the functioning of the nervous system at high spatio-temporal resolution [1,2]. These recordings contain action potentials (i.e. spikes) fired by individual neurons that are located in the vicinity of the tip of the recording microelectrode. The raw recording is band-pass filtered within a frequency range suitable for spike detection [3]. Figure 1 illustrates a snippet of recording from rat primary motor cortex which was sampled at 40 kHz and band-pass filtered between 400 Hz and 8 kHz using a 4th order Butterworth filter [4].

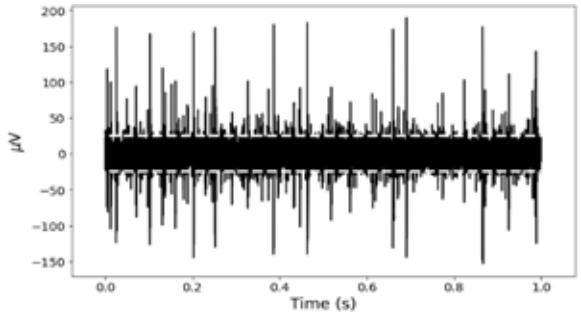

Fig. 1. Extracellular recording from the primary motor cortex of a rat.

In Figure 1, action potentials fired by nearby neurons are visible as spikes, the amplitudes of which depend inversely on the distance of the neuron from the electrode. The very low amplitude spikes of thousands of distant neurons superimpose to yield a band of background activity that has noise characteristics. Usually an amplitude threshold is used to separate spikes from the background activity [5,6,7,8,9]. It has been shown that even small amplitude spikes carry useful information for Brain-Machine Interface (BMI) operations, as long as they are above threshold [9]. This has raised the problem of computing optimal threshold values, which is an active research area [10,11]. Recently, Okatan and Kocatürk proposed an algorithm for automatically determining a pair of thresholds that separate spikes from the background activity in an entirely data-driven manner [4]. The resulting thresholds are called 'truncation thresholds'. The thresholds computed for the data in Figure 1 are shown as white dashed horizontal lines. The MATLAB source code and standalone executable files of the algorithm that computes the truncation thresholds are available at SciCrunch.org under the name "Truncation Thresholds Software" (RRID:SCR_014637). These files are suited for offline analysis. Yet there is a real need for computing spike detection thresholds online during data acquisition using principled methods.

As part of a plan to integrate the computation of truncation thresholds into a data acquisition system written in C [12], we explored the possibility of converting the available MATLAB source code into C using MATLAB Coder (MATLAB Version: 9.8.0.1323502 (R2020a). Natick, Massachusetts, USA: The MathWorks Inc.). However, MATLAB Coder could not convert some of the key functions required [13]. Because there are tools available to run C and Python codes in an integrated way [14], the required code could also be written in Python. Our literature search failed to reveal any study where the required codes were developed in Python or C and compared in performance to MATLAB. Therefore, here we present a Python code for maximum likelihood estimation of the location and scale parameters of the truncated Normal distribution, which is

This work was supported by Research Fund of the Istanbul Technical University. Project Number: MAB-2020-42808.

one of the steps involved in the computation of truncation thresholds, and compare its performance to MATLAB's mle.m function (MATLAB Version: 9.7.0.1261785 (R2019b) Update 3. Natick, Massachusetts, USA: The MathWorks Inc.).

II. METHOD

A. Maximum Likelihood Fitting of the Truncated Normal Distribution

The truncated Normal distribution naturally arises in problems where a Normally distributed data set is thresholded or screened [15]. Maximum likelihood fitting of truncated Normal distribution has been shown to yield more accurate results in applications such as speech power estimation [16] and noise standard deviation estimation [4]. The computation of the truncation thresholds involves an iterative search for a pair of voltage values intercalating the median of the amplitude distribution [4]. At each iteration, the amplitude distribution is truncated at the current threshold candidates and a truncated Normal probability density function (Eq. 1) is fitted by maximum likelihood to the truncated data. Namely, the maximum likelihood estimates of the parameters μ and σ are obtained given the data and the truncation points a and b, with $a<b$.

$$f(x|a,b,\mu,\sigma) = \frac{\frac{1}{\sigma}\phi\left(\frac{x-\mu}{\sigma}\right)}{\Phi\left(\frac{b-\mu}{\sigma}\right) - \Phi\left(\frac{a-\mu}{\sigma}\right)} \quad (1)$$

In Eq. 1, $\phi(\cdot)$ is the standard Normal probability density function and $\Phi(\cdot)$ is the standard Normal cumulative distribution function.

The goodness-of-fit is assessed using the Kolmogorov-Smirnov (KS) test at level 0.05. The widest interval $[a, b]$ that passes the KS test is searched by halving the cardinality of the search set at each iteration [17]. The resulting a and b values are called truncation thresholds [4].

Figure 2 shows the truncation thresholds obtained for the data in Figure 1.

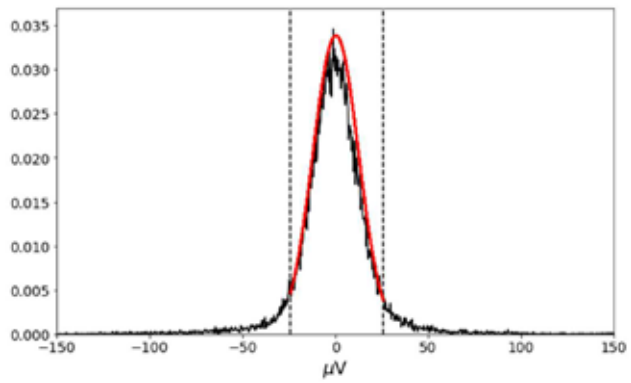

Fig. 2. Normalized histogram of the extracellular activity (black) with the fitted truncated Normal probability density function (red). The vertical dashed lines indicate the truncation thresholds.

Our Python code computes the log-likelihood function (Eq. 2) of the parameter vector using the SciPy Stats [18] Python library and then minimizes the negative log-likelihood using the SciPy Optimize [19] Python library.

$$l(\mu,\sigma|a,b,x_{1:N}) = \sum_{i=1}^{N} log\left(f(x_i|a,b,\mu,\sigma)\right) \quad (2)$$

Here, $a \le x_i \le b$ is the i^{th} sample in the data set truncated at a and b, and N is the number of samples in the truncated data set.

The steps of the algorithm are given as follows.

1. Truncate the data according to lower and upper thresholds
2. Define the function that calculates the probability density of truncated data and the negative log-likelihood function
3. Minimize the negative log-likelihood function with Nelder-Mead optimization method
4. Get estimated location and scale parameters that have minimum negative log-likelihood

In MATLAB, the mle.m function uses fminsearch.m function by default to find the minimum point of a function [20]. This function uses the Nelder-Mead simplex search method [21,22]. Therefore, we used the Nelder-Mead optimizer in SciPy minimizer function to make an objective comparison between MATLAB and Python using both real and simulated data sets. Our code is available at:

https://web.itu.edu.tr/okatan/SOFTWARE/TruncatedNormalMLFitPython/

B. Comparison of Computation Times and Complexity

We performed 10 independent measurements of the total time taken by our Python and MATLAB codes to process our real and simulated data sets. We then compared these measurements using the Kruskal-Wallis test. We also compared the number of function evaluations performed during the fitting of each truncation threshold pair. All computations were performed on a system with Intel(R) Core (TM) i7-10750H CPU @ 2.60GHz 12-core processor with 12 GB RAM under Ubuntu 20.10 Linux operating system.

III. RESULTS

A. Parameter Estimation for Neural Recording Data

Truncation Thresholds Software tests 15 different threshold pairs while iteratively calculating the truncation thresholds for the data in Figure 1. The truncated Normal distribution parameters estimated at each tested threshold pair were also estimated by the code we proposed, and the obtained results are compared with the results of the original algorithm in Figure 3.

The results in Figure 3 clearly show that the Python implementation of the algorithm gives almost the same results as the MATLAB implementation for the extracellular neural recording data. Also, all estimated parameters are within the confidence interval of MATLAB's estimates.

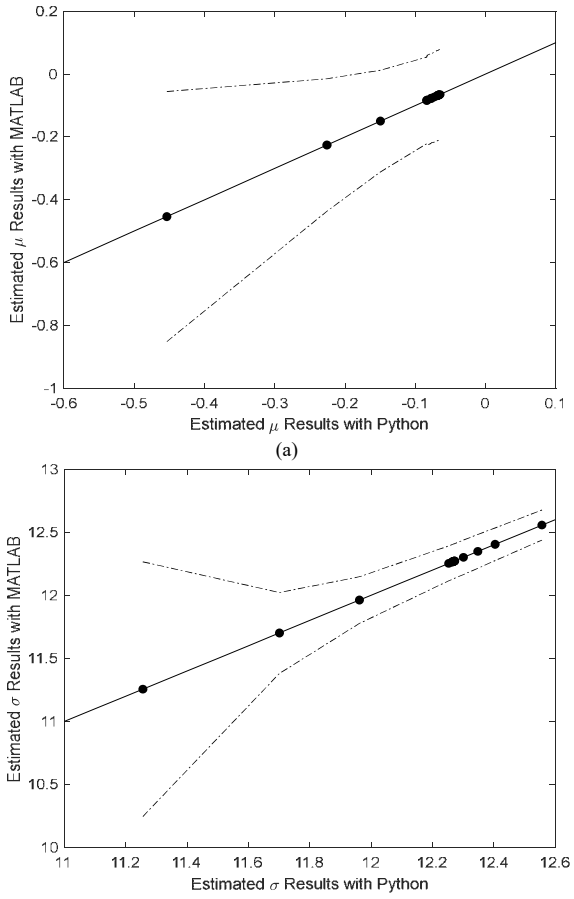

Fig. 3. Scatter plots of the results. The dashed curves show the 95% confidence interval of the estimates computed using MATLAB. The solid line shows the 45-degree line. (a) estimated μ parameters (b) estimated σ parameters.

Table 1 shows the maximum, minimum and mean values of the difference between the estimates obtained using the two approaches.

TABLE I. Maximum, minimum and mean values of the difference between the estimates

	Max	Mean	Min
μ_{Python}- μ_{MATLAB} (μV)	7.1032×10^{-5}	-1.381×10^{-6}	-6.3702×10^{-5}
σ_{Python}- σ_{MATLAB} (μV)	5.0402×10^{-5}	1.3656×10^{-5}	-2.8872×10^{-5}
Δμ (%)	0.0886%	-0.0002%	-0.0758%
Δσ (%)	0.0004%	0.0001%	-0.0002%

In Table 1, Δ represents the difference between the estimates as a percentage of the estimates obtained using the Python code. As can be seen in Table 1, the difference between the parameters estimated using Python and MATLAB is quite small.

B. Parameter Estimation for Standard Normal Distribution

In this section, we generate 10^6 independent samples from the standard Normal distribution. We then truncate these data at points [a, b], where a is any one of the 2.5th, 5th, 10th, 20th, 30th, and 40th percentiles of the standard Normal distribution and b is any one of the 97.5th, 95th, 90th, 80th, 70th, and 60th percentiles of the same distribution, as shown Fig. 4, yielding 36 distinct pairs of truncation points. We estimate the parameters of the truncated Normal distribution for these 36 data sets and compare the results of the two approaches in Figure 5.

The estimated parameters for the simulated data are almost the same for MATLAB and Python. Here, again, all Python estimates and true parameter values are within the confidence interval of the MATLAB estimates. Table 2 shows the Mean Squared Error (MSE) values between the true parameter values and the estimated parameters.

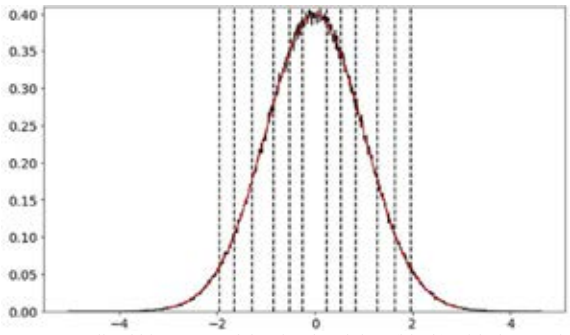

Fig. 4. Normalized histogram of the simulated data (black) with the standard Normal probability density function (red) overlaid. Vertical lines indicate the truncation points considered in the analysis.

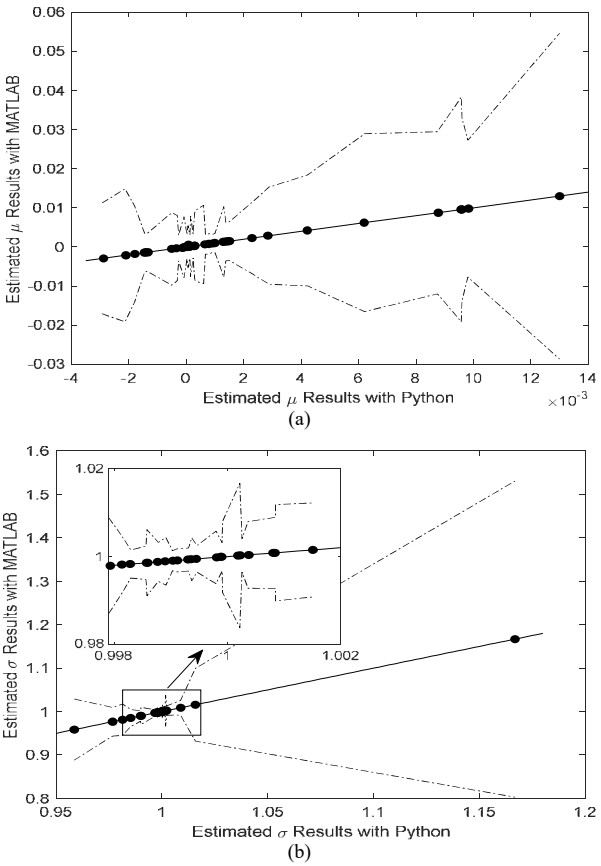

Fig. 5. Scatter plots of the results. The dashed curves show the 95% confidence interval of the estimates computed using MATLAB. The solid line shows the 45-degree line. (a) estimated μ parameters (b) estimated σ parameters.

TABLE II. MSE between true and estimated parameter values for the 36 pairs of truncation points considered.

	MATLAB	Python
MSE μ	1.7497×10^{-5}	1.7494×10^{-5}
MSE σ	0.00086846	0.00086830

MSE between the true and estimated parameter values for the 36 different pairs of truncation points is quite low. Also, MSE values in MATLAB and Python are very close to each other.

C. Comparison of Computation Times and Complexity

The number of function evaluations of the Nelder-Mead optimizer was used to compare the computation cost between MATLAB and Python as in [23]. For neural recording data, the evaluation numbers of MATLAB and Python have a roughly constant difference in all cases, with MATLAB taking 31.4±2.87 extra evaluations on the average (mean±std) to minimize the function (P=0.022; Kruskal-Wallis test). Default parameters of the Nelder-Mead optimizer were used in both MATLAB and Python. For standard Normal distribution, the number of function evaluations of the Nelder-Mead optimizer was systematically higher for MATLAB by 29.33±12.32 evaluations on the average (mean±std) as with parameter estimation for neural recording data ($P=6.6\times10^{-9}$, Kruskal-Wallis test).

Python takes longer than MATLAB to fit the truncated Normal distribution to the real or simulated data sets (P=0.0002, Kruskal-Wallis test). The difference is 0.90±0.09 s and 25.69±3.19 s (mean±std) for the real (4×10^4 samples) and simulated (10^6 samples) data sets, respectively.

IV. CONCLUSION

In this study, we developed a Python code for maximum likelihood estimation of the location and scale parameters of the truncated Normal distribution given the truncation points. The estimation results and the computational cost were compared to the same operation performed using MATLAB's mle.m function. All Python estimates were found to be within the 95% confidence interval of the MATLAB estimates, and the estimates obtained using the two methods were virtually identical. The computational cost of the Python algorithm was found to be less than that of the MATLAB algorithm. However, MATLAB was faster than Python. In future studies we plan to explore whether this computation can be accelerated in Python. The results show that the Python code presented here can be used as a viable alternative in platforms where MATLAB is unavailable.

REFERENCES

[1] G. Buzsáki, "Large-scale recording of neuronal ensembles," *Natural Neuroscience*, vol. 7, pp. 446-451, 2004.
[2] M.E.J. Obien, K. Deligkaris, T. Bullmann, D.J. Bakkum and U. Frey, "Revealing neuronal function through microelectrode array recordings," *Front Neurosci*, vol 8, pp. 423, 2015.
[3] M.S. Lewicki, "A review of methods for spike sorting: the detection and classification of neural action potentials." *Network Computation in*
[4] M. Okatan and M. Kocatürk, "Truncation thresholds: a pair of spike detection thresholds computed using truncated probability distributions," *Turkish Journal of Electrical Engineering & Computer Sciences*, vol 25, pp. 1436-1447, 2017.
[5] R.Q. Quiroga, Z. Nadasdy and Y. Ben-Shaul, "Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering," *Neural Comput* ,vol.16, pp. 1661-1687, 2004.
[6] C. Vargas-Irwin and J.P. Donoghue, "Automated spike sorting using density grid contour clustering and subtractive waveform decomposition," *Journal of neuroscience methods*, vol. 164, pp. 1-18, 2007.
[7] T. Takekawa, Y. Isomura and T. Fukai, "Accurate spike sorting for multi-unit recordings," *European Journal of Neuroscience,* vol. 31, pp. 263-272, 2010.
[8] D. Jäckel, U. Frey, M. Fiscella, F. Franke and A. Hierlemann "Applicability of independent component analysis on high-density microelectrode array recordings," *Journal of Neurophysiology*, vol 108, pp. 334-348, 2012.
[9] S. Todorova, P. Sadtler, A. Batista, S. Chase and V. Ventura, "To sort or not to sort: the impact of spike-sorting on neural decoding performance," *Journal of neural engineering*, vol. 11, 2014.
[10] B.P. Christie et al., "Comparison of spike sorting and thresholding of voltage waveforms for intracortical brain–machine interface performance," *Journal of neural engineering*, vol 12, 2015.
[11] E.R. Oby, et al., "Extracellular voltage threshold settings can be tuned for optimal encoding of movement and stimulus parameters," *Journal of neural engineering*, vol. 13, 2016.
[12] M. Kocaturk, H.O. Gulcur and R. Canbeyli, "Toward building hybrid biological/in silico neural networks for motor neuroprosthetic control. " *Frontiers in Neurorobotics*, vol. 9, pp. 8, 2015.
[13] Functions and Objects Supported for C/C++ Code Generation MATLAB R2020a, https://www.mathworks.com/help/coder/ug/functions-and-objects-supported-for-cc-code-generation.html (online available).
[14] C-Extensions for Python, https://cython.org/ (online available).
[15] J. Pender, "The truncated normal distribution: Applications to queues with impatient customers", Operations Research Letters, vol. 43(1), pp. 40-45, 2015.
[16] C. Lu, "Speech power estimation with a truncated normal distribution," ICASSP '87. IEEE International Conference on Acoustics, Speech, and Signal Processing, Dallas, TX, USA, 1987, pp. 1736-1739.
[17] W.H. Press, S.A. Teukolsky, B.P. Flannery and W.T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing,* 2nd ed., Cambridge, UK: Cambridge University Press, 1992.
[18] SciPy Statistical Functions (scipy.stats) documentation, https://docs.scipy.org/doc/scipy/reference/stats.html (online available)
[19] SciPy Optimization Package (scipy.optimize) documentation, https://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html (online available)
[20] Maximum likelihoods estimates (mle.m) documentation, MATLAB R2020b, https://www.mathworks.com/help/stats/mle.html (online available)
[21] Find minimum of unconstrained multivariable function using derivative free-method (fminsearch.m) documentation, MATLAB R2020b,https://www.mathworks.com/help/MATLAB/ref/fminsearch.html (online available)
[22] J.C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright. "Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions." *SIAM Journal of Optimization*. Vol. 9, pp. 112–147, 1998.
[23] L. S. Matott, K. Leung and J. Sim, "Application of MATLAB and Python optimizers to two case studies involving groundwater flow and contaminant transport modeling," *Computers & Geosciences,* vol. 37, pp. 1894-1899, 2011.