A Galton board on a rocking ship

Erik Andersson, Sebastian Holmin

December 2020

In this report we will investigate the use of the Approximate Bayesian omputation (ABC) algorithm, supported by a neural network (NN), to reverse engineer the parameters for a toy model of a Galton board on a rocking ship.

A Galton board (bean machine) is a device that produces a normal distribution by utilizing the law of big numbers. It consists of rows of pegs where balls can roll a step to the left or the right at each row. Our toy model has 31 rows, giving 32 possible end positions for each ball, and two parameters α and s. The parameter α describes a simplified moment of inertia, i.e. the tendency for a ball to continue rolling in the same direction again for the next peg, while s describes the incline of the rocking ship that the Galton board is situated on. The probability of a ball rolling to the right is then given by

$$P_+ = 0.5 + (\alpha M + s)$$

where $\alpha \in [0, 0.5]$ and $s \in [-0.25, 0.25]$ and M is -0.5 if the ball previously rolled to the left and 0.5 if it rolled to the right.

We are faced with the task of determining an unknown (but constant) α from a 'black box' function that simulates the final positions of 1000 balls. This is done for an unknown, randomly chosen, latent variable s. To help us with this task we will implement our own simulator where we can control the parameters and analyse the behaviour.

Phrased in a Bayesian language, given a (set of) simulated distributions y_m, find the posterior distribution

$$p(\alpha|y_m) = \int p(\alpha, s|y_m)\mathrm{d}s = \int \frac{p(y_m|\alpha, s)\pi(\alpha)\pi(s)}{p(y_m)}\mathrm{d}s \tag{1}$$

where we have first used the law of total probability to write the likelihood as marginalized over the latent variable s and then Bayes theorem to write the posterior in terms of the likelihood. As this is a toy model, we will assume that s is drawn uniformly in its allowed interval for every run of the 'black box' simulation, and that the 'true' α was chosen randomly from uniform distribution. That is, we will assume that the priors $\pi(\alpha)$ and $\pi(s)$ are uniform.

The accuracy of the posterior can be increased using the results from several experiment outcomes. Given a set y_m^i, for $i = 1, ..., N$, the total posterior is given by

$$p(\alpha|y_m^0, ..., y_m^N) \propto \pi(\alpha)\prod_{i=1}^{N} p(y_m^i|\alpha) \propto \prod_{i=1}^{N} p(\alpha|y_m) \tag{2}$$

where we have use the fact that $\pi(\alpha)$ is uniform to insert the posterior of eq. (1).

1 The ABC algorithm

To tackle the problem of evaluating the posterior of eq. (1), we will resort to sampling it using the ABC algorithm. This will allow us to avoid the challenging analytical calculation of the likelihood and to ignore the constant evidence $p(y_m)$ in the denominator. We will make use of our simulator to avoid explicit evaluation of the likelihood and a NN trained to solve the reverse problem (estimate the posterior), to speed up the process. To explain how this works we first need to go through some preliminaries.

1.1 Likelihood free rejection sampling

The standard rejection sampling algorithm effectively allows you to sample a target distribution $f(\theta)$ using only samples drawn from a proposal distribution $g(\theta)$. It works as follows, given a sample $\theta \sim g(\theta)$, accept it with probability $f(\theta)/[Mg(\theta)]$ where $M \geq \max_\theta f(\theta)/g(\theta)$. The algorithm will take on average M attempts to sample from $f(\theta)$, hence its efficiency benefits from a proposal function that accurately predicts the target distribution.

The rejection sampling algorithm can be applied to the problem of sampling from a posterior by setting $f(\theta) = p(y_m|\theta)\pi(\theta) \propto p(\theta|y_m)$. This however requires to ability to numerically evaluate the likelihood, which is not feasible for our problem. The core realisation of the ABC algorithm is that we can avoid evaluating the likelihood if we have a way to simulate (or perform) the experiment for any choice of parameters θ. Since a simulator basically is a machine that samples $y \sim p(y|\theta)$, if we run the simulation to produce y given θ, then outcomes where $y = y_m$ will happen with probability $p(y_m|\theta)$. This allows us to sample $\theta \sim p(\theta|y_m)$ by, generating $\theta \sim g(\theta)$, simulating y from this θ, then accepting θ with probability $\pi(\theta)/[Mg(\theta)]$ when $y = y_m$, where $M \geq \max_\theta \pi(\theta)/g(\theta)$.

1.2 Kernel functions and summary statistics

The second important aspect of ABC comes from the 'Approximate' part of the name. Measured data y_m often (certainly in our case) has a very large number of possible outcomes and is highly multidimensional. Thus the probability that $y = y_m$ is far too low for practical use. ABC has two ways of assessing this issue. First, one can reduce recondition $y = y_m$ to $\mathfrak{s} = \mathfrak{s}_m$[1], where $\mathfrak{s} = S(y)$ is a so called summary statistic. It captures the important properties of y with far fewer dimensions. An example of summary statistic might be the mean of y.

The second trick is to replace $\mathfrak{s} = \mathfrak{s}_m$ with the more lenient condition of accepting imperfect matches with probability $K_h(\|\mathfrak{s} - \mathfrak{s}_m\|)$, where $K_h(u) = \frac{1}{h}K\left(\frac{u}{h}\right)$ for some kernel function $K(u)$. Kernels are symmetric and satisfy $K(u) \geq 0, \int K(u)\mathrm{d}u = 1, \int uK(u)\mathrm{d}u = 0$ and $\int u^2K(u)\mathrm{d}u > \infty$, and h is a scale parameter which determines how strict the similarity must be. It can be shown[1, p.128, c. 5.1.2] that with a sufficient $\mathfrak{s}$ (which can still be lower dimensional than y) and $h \to 0$ the approximation approaches zero.

The Kernel function can be combined into the acceptance step of the rejection algorithm, so that the sample $\theta \sim g(\theta)$ is accepted with probability $K_h(\|\mathfrak{s} - \mathfrak{s}_m\|)\,\pi(\theta)\,/\,[Mg(\theta)]$ where $M \geq K_h(0)\max_\theta \pi(\theta)/g(\theta)$.

[1]Note here that we use the fraktur versions of $\mathfrak{s}$, provided by the *amsfonts* package, to differentiate between the summary statistic $\mathfrak{s}$ and the parameter s.

1.3 Importance sampling version of ABC

We have chosen not to implement the standard ABC algorithm described above for the following reason. For the standard rejection sampling algorithm, the best performing proposal distribution $g(\theta)$ is one as close to the target distribution $f(\theta)$ as possible, which will allow for $M \geq \max_\theta f(\theta)/g(\theta)$ close to unity. However, for the ABC algorithm we get an optimal performance using $g(\theta) = \pi(\theta)$. This is due to the fact that the normalizing constant becomes

$$M \geq \frac{\max_\theta K(\|\mathfrak{s} - \mathfrak{s}_m\|)\pi(\theta)}{g(\theta)} = K_h(0) \max_\theta \pi(\theta)/g(\theta),$$

since $\mathfrak{s} \sim p(\mathfrak{s}|\theta)$ always gives a non-zero probability of generating $\mathfrak{s} - \mathfrak{s}_m$ for any θ. The total probability of adding a certain sample to your posterior distribution will be

$$g(\theta)K_h\left(\|\mathfrak{s} - \mathfrak{s}_m\|\right)\pi\left(\theta\right) / \left[Mg\left(\theta\right)\right] = K_h\left(\|\mathfrak{s} - \mathfrak{s}_m\|\right)\pi\left(\theta\right)/M$$

which is maximized by $M = K_h(0)$, i.e. $g(\theta) = \pi(\theta)$. Since we have chosen a uniform prior, such a $g(\theta)$ would overwhelmingly sample outside the important regions of the posterior, causing the factor $K(\|\mathfrak{s} - \mathfrak{s}_m\|)$ to be very close to zero in most cases and only a small amount of samples will be accepted. To remedy this we have implemented instead an importance sampling version of ABC [1, p. 93]. Such an algorithm replaces the probability of rejecting a sample with instead associating it with a weight $\tilde{w} = K_h\left(\|\mathfrak{s} - \mathfrak{s}_m\|\right)\pi\left(\theta\right)/g\left(\theta\right)$, quantifying how much the sample should be counted for. This removes the need for the normalizing constant $M \geq \max_\theta \pi(\theta)/g(\theta)$ as the weights (as opposed to acceptance probabilities) are allowed to exceed one. Thus, we are free to chose $g(\theta)$ as we see fit to most accurately predict the posterior.

1.4 Elimination of latent variables

Our problem has the complication of including a latent variable. As stated earlier in eq. (1), this can be dealt with by simply sampling the posterior for the set of both variables (α, s), and marginalizing over s. Marginalization is especially easy with sampled parameters, as it simply entails ignoring the s. The full ABC algorithm we will use is thus

NN supported ABC importance sampling algorithm with latent variables.
Task: Sample $(\alpha^{(i)}, \tilde{w}^{(i)}) \sim p(\alpha|y_m)$ from eq. (1), for $i = 1, ..., N$.
Input:

- A NN prediction α_{guess} and s_{guess} from y_m.
- A proposal distribution $g(\alpha, s, \alpha_{\text{guess}}, s_{\text{guess}}) > 0$ for all α and s where $p(\alpha, s|y_m) > 0$.
- A kernel $K_h(u)$.
- Summary statistic $S(y)$.

Algorithm: (Repeat for $i = 1, ..., N$)

1. Generate $(\alpha^{(i)}, s^{(i)}) \sim g(\alpha, s, \alpha_{\text{guess}}, s_{\text{guess}})$
2. Generate $y^{(i)}$ from simulation using $\alpha^{(i)}, s^{(i)}$.
3. Compute $\mathfrak{s}^{(i)} = S(y^{(i)})$
4. Compute weight $\tilde{w}^{(i)} = K_h\left(\left\|\mathfrak{s}^{(i)} - \mathfrak{s}_m\right\|\right) \pi\left(\alpha^{(i)}, s^{(i)}\right) / g\left(\alpha^{(i)}, s^{(i)}\right)$.

Note in step 5 that we can ignore the priors as we have set them to be independent and uniform.

1.4.1 A note on the algorithm from the lecture

In the lecture on ABC, a version of NN supported ABC (using rejection sampling) which automatically eliminates latent variables was proposed as follows:

1. Use the NN to predict $y_m \to s_{\text{guess}}$
2. Generate $s^{(i)} \sim g(s, s_{\text{guess}})$
3. Generate $\alpha^{(i)} \sim \pi(\theta)$
4. Generate $y^{(i)}$ from simulation using $\alpha^{(i)}, s^{(i)}$.
5. Accept $\alpha^{(i)}$ with probability $K_h\left(\left\|y^{(i)} - y_m\right\|\right)$.
6. Repeat from step 2

We believe that this algorithm in principle is faulty, as it samples from the distribution

$$\int \frac{p(y_m|\alpha, s)\pi(\alpha)g(s)}{p(y_m)} \mathrm{d}s, \tag{3}$$

which is only equal to the marginalized posterior $p(\alpha|y_m)$ if $g(s) = \pi(s)$, or if the likelihoods dependance on α and s are uncorrelated, i.e. $p(y_m|\alpha, s) \propto p(y_m|\alpha)$. If α and s are correlated (picture the likelihood $p(y_m|\alpha, s)$, as a function of α and s, being skewed diagonally), then any choice of $g(s)$ more specific than the uniform prior (which is every $g(s)$), will cause the marginalization of

eq. (3) to be more narrow than what it should be, and if the modes of $g(s)$ and $p(y_m|\alpha, s)$ do not coincide, then the mode of the marginalized distribution might be slightly shifted.

In our samplings of the likelihood $p(y_m|\alpha, s)$, we found that α and s are correlated for y_m generated using a non-zero s. In practice, we might expect a NN to be fairly accurate in its predictions of the mode, so the main consequence of using this algorithm would be that your posterior distribution eq. (3) would be slightly too narrow unless you discard y_m generated with large s.

2 Implementation

In this chapter we describe in detail the implementation of the theory stated in the previous chapter. We begin by describing the neural network used to propose samples for the ABC algorithm. We then explain and motivate our choice of kernel function and summary statistic and lastly we explain our choice of proposal functions.

2.1 Designing a suitable neural network

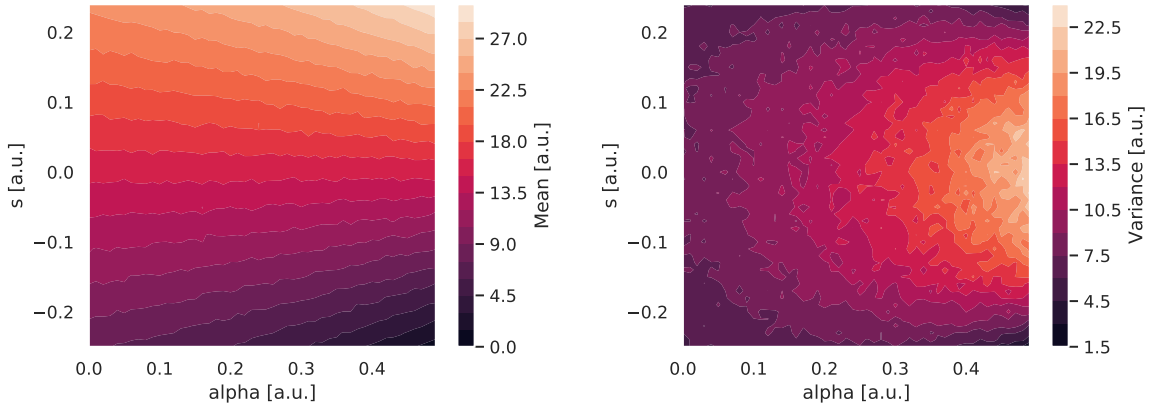
To solve the inverse problem we implement a neural network with 32 input neurons (one for each bin of the experiment output) and two output neurons (one for each parameter). We also implemented two hidden layers with 16 and 8 neurons respectively. This choice was mostly made through trial and error, trying to achieve a good test score with an as simple network as possible. For this reason we used the quite standard tanh activation function. And set the convergence tolerance to 10^{-6} with an initial learning rate of 0.1. This was mostly done by trial and error to get as good results with as fast a training time as possible. To train the network, a set of 30000 input parameters were generated randomly, through $\alpha \sim \mathcal{U}(0, 0.5)$ and $s \sim \mathcal{U}(-0.25, 0.25)$, and used as input to the simulator. The output of the simulator were given as input to the network and the input were given as targets. The training consisted of using 75% for the training and the remaining 25% for testing. The scoring given was $1 - u/v$, where $u = \sum(y_{pred} - y_{true})^2$ and $v = \sum(y_{true} - y_{true,mean})^2$. The best score possible is thus 1 and the difference from the best score shown how much the prediction varies from the true value normalized by the variation of the targets. Thus, if the predictions are off by the same amount as the variation of the targets the the score is 0 and if the predictions are exact the score is 1. After training the network achieved a test score of 0.988 which we deem sufficient for our purposes. The Mean Squared Error (MSE) for the test set was $5.20 \cdot 10^{-4}$ for α and $3.32 \cdot 10^{-5}$ for s.

2.2 Choice of Kernel and summary statistics

In choosing the kernel function we need to take into account the error of the approximation. This can be stated as [1, p. 18]

$$p_{ABC}(y_{obs}|\theta) - p(y_{obs}|\theta) = \frac{1}{2}h^2 p''(y_{obs}|\theta) \int u^2 K(u) du + \mathcal{O}(h^4).$$

We see that the variance of the kernel function determines the error. With a standard Gaussian K (i.e. K_h is Gaussian with variance h^2) we see that the error is proportional to h^2. The choice of a Gaussian is particularly good in importance sampling since it automatically weighs $s^{(i)}$ higher the closer it is to $\mathfrak{s}_m$. We implemented h as a vector valued scaling factor that can weigh the



(a) (b)

Figure 1: The mean and variance generated by the controlled experiment.

components of the summary statistic independently. We then set it to 70% of the standard deviation of the components of $\mathfrak{s}$ generated with the predicted parameters. This heuristic approach roughly corresponds to the intention that $\mathfrak{s}$ generated from the same parameters should be identified as equal. We decided on the factor 0.7 as it seemed to provide decent robustness without unreasonable computational cost.

In choosing $\mathfrak{s} = S(y)$ to describe the distribution y_m, the natural choice of statistics is the mean and the variance. To investigate how suitable this choice would be we evaluated these properties for trial runs with a controlled experiment method in a grid of α and s values. The result is shown in fig. 1. We can see that, since the isolines of the mean and variance are orthogonal, these statistics should be able to uniquely identify both α and s, except for in the extreme case of both parameters being very large. For larger magnitude s the lines are instead more or less parallel meaning that a kernel based on mean and variance will induce a correlation between α and s in the posterior distribution. Since the real experiment has a randomly varying s this correlation should average out, but may lead to a larger standard deviation than a Kernel which does not have this feature. We also investigated using higher order moments of the distribution such as skewness and kustosis, and also the mode. However these were all highly correlated with either the mean or the variance so we thought it unlikely to improve the analysis.

We could possibly remedy the correlation problem by simply rejecting experiment results with too extreme predicted s values. In this case it could be even more efficient since running an experiment is relatively cheap (about the same cost as running a simulation). In a real life application, where running experiments may be very expensive this may not be a very good solution.

2.3 Proposal function $g(\alpha, s)$ using neural network

In the ABC algorithm, the choice of proposal function does not affect the resulting distribution, we are free to choose whatever we like. The speed of convergence is however increased the closer we can predict the posterior. For simplicity, we chose to model this as two uncorrelated normal distributions around the predictions α_{guess} and s_{guess} made by the NN, with variance given by the

MSE of the NN on its test set, i.e. $g_\alpha(\alpha, \alpha_m) = \mathcal{N}(\alpha_m, \sigma_\alpha^2)$ and $g_s(s, s_m) = \mathcal{N}(s_m, \sigma_s^2)$. It turns out that there is some correlation between α and s for large magnitude s, but we choose to disregard this.

3 Results

Each run of the ABC algorithm of section 1.4 will produce the posterior given one simulated data set y_m. We terminated the algorithm when the total weight of the samples exceeded 10, which corresponds to roughly 60000 iterations. The value of the weight limit is of course arbitrary, and should in principle be normalized. It can be thought of as an upper bound on how many samples a rejection version of the algorithm would have produced.

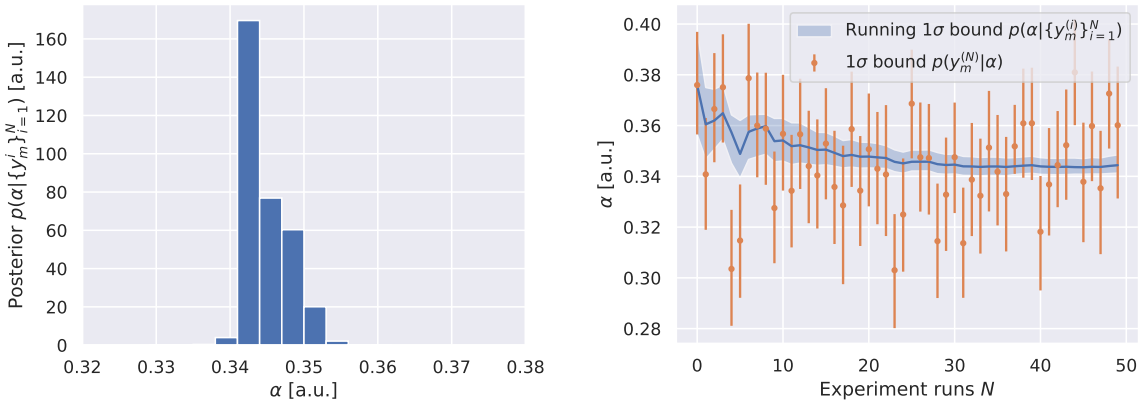
We ran the algorithm and analysed the resulting posterior distribution for each new $y_m^{(i)}$ added to eq. (2). The convergence can be seen in fig. 2b, where the running mean is plotted together with a 1σ bound, and the final posterior distribution in fig. 2a. The mean converges to 0.344 and we calculated a 95% credible interval of $0.340 < \alpha < 0.353$. This was done by calculating the center-most region of the posterior corresponding to 95% probability. Note however that this credible interval assumes a perfect posterior as it does not take approximations resulting from a non-zero h and a finite number of samples N for each likelihood. Thus, the actual confidence interval for 'true' value of α should probably be somewhat larger.

3.1 Control test

To verify that the accuracy of the confidence interval we have given is reasonable we also tested the algorithm on our own simulation with $\alpha = 0.35$, and a uniformly random s to mimic the experiment set up. The posterior seems to have converged after roughly 40 analysed $y_m^{(i)}$, and can be seen in fig. 3a. The mean of the total posterior is $\alpha = 0.346$, with a 95% credible interval of $0.341 < \alpha < 0.351$. We note that the results are almost identical to the unknown experiment. Whether this is a coincidence or that they both in fact underestimate α is not possible to say without more controlled experiments. It might also be that it is simply more likely to get a distribution that looks (as regards the mean and variance) like it was generated with a smaller α than with a larger one. As an example we can note that 20 of the 50 samples are above the true value and also that the deviation from the true value is larger for the runs with mean below the true value than for those with mean above it.

4 Discussion

The ABC algorithm we have used, in the limit of $h \to 0$ and number of samples $N \to \infty$, should converge to the 'true' posterior $p(\alpha|y_m)$. This however, would not gives us an exact prediction of α as the experiment setup has an inherent randomness, i.e. $p(\alpha|y_m)$ has a non-zero variance. To further improve the prediction, several simulated y_m are needed. As one particular y_m could conceivably have come from a slightly different α, but the probability of an incorrect α being predicted several times decreases for every new y_m analysed. It seems however as if the total posterior might converge with a non-zero variance, i.e. that a perfect prediction is not possible even for an infinite amount of simulated data sets y_m.

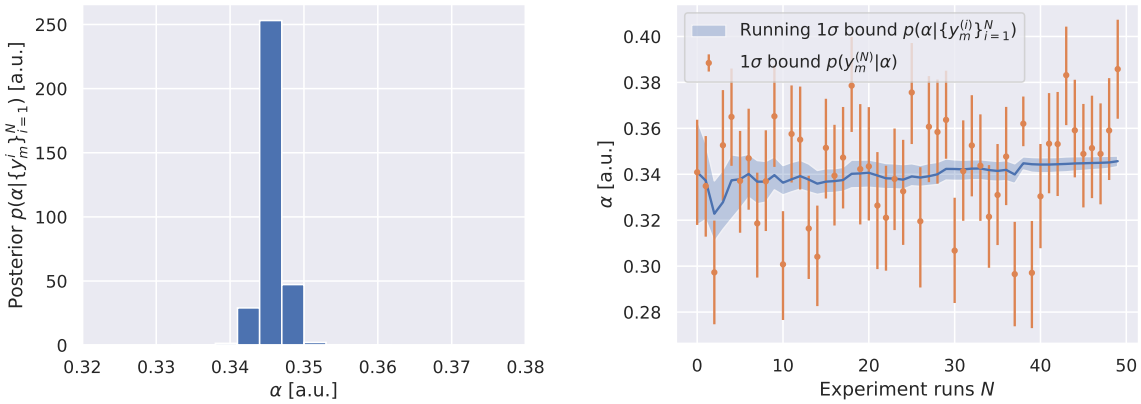


(a) (b)

Figure 2: (a) The resulting total posterior for 50 runs of the unknown experiment. (b) Convergence of the posterior $p(\alpha|\{y_m^{(i)}\}_{i=1}^N)$ prediction shown as a running 1σ confidence bound for increasing number of simulated likelihoods $p(y_m|\alpha)$ incorporated into eq. (2). The component likelihoods $p(y_m^{(N)}|\alpha)$ are also shown in orange.

We can probably assume that increasing the resolution of our distributions by increasing the number of bins as well as running longer simulations should increase the accuracy of the method. The theory presented above should produce the correct results and it is most probably numerical issues that causes the slight deviation of the most probable value. However we have not had time to run these longer simulations.

The computational cost scales roughly as the inverse square of h, as our summary statistic is two-dimensional. Because of this, decreasing h for better accuracy is very expensive. We encountered a fair bit of noise, and sometimes large 'spikes' in our likelihoods, that we could not entirely eliminate by lowering h. Potentially a smoothing filter could be applied as an alternative to increase the robustness of the method, as we expect the true distributions to be smooth. How this would effect the statistical analysis is however difficult to say.



(a) (b)

Figure 3: (a) The resulting total posterior for 50 runs of the controlled experiment. (b) Convergence of the posterior $p(\alpha|\{y_m^{(i)}\}_{i=1}^N)$ prediction shown as a running 1σ confidence bound for increasing number of simulated likelihoods $p(y_m|\alpha)$ incorporated into eq. (2). The component likelihoods $p(y_m^{(N)}|\alpha)$ are also shown in orange.

References

[1] Scott A. Sisson, Yanan Fan, and Mark A. Beaumont. *Handbook of approximate Bayesian computation.* Chapman Hall/CRC handbooks of modern statistical methods. CRC Press, 2019.