To cite this chapter:

Loossens, T., Verdonck, S., & Tuerlinckx, F. (2020). Synergized bootstrapping: The whole is faster than the sum of its parts. In M. Wiberg, D. Molenaar, J. González, U. Bockenholt, & J. S. Kim (Eds.), *Quantitative Psychology: The 84th annual meeting of the psychometric society, Santiago, Chile, 2019* (pp. 227-245). Springer. https://doi.org/10.1007/978-3-030-43469-4_18

Synergized Bootstrapping: the Whole is Faster than the Sum of its Parts

Tim Loossens, Stijn Verdonck, and Francis Tuerlinckx

KU Leuven, 3000 Leuven, Belgium,
tim.loossens@kuleuven.be

Abstract. Re-sampling methods are popular for assessing uncertainty, testing hypotheses or for cross-validation because of their simplicity. They all rely on a similar scheme: generating replicated datasets by sampling data points from an original dataset, fitting a model or conducting a statistical test on each of these, and aggregating the results. However, when fitting the model or conducting the statistical test becomes time-consuming, re-sampling methods become impractical because of the many replications. Many methods have been proposed to alleviate the computational burden, but they generally do not incorporate two key features of re-sampled datasets. One, re-sampled datasets all stem form the same origin and therefore have similar characteristics. Two, there is a large class of cost functions for which the cost of a parameter set given data can be computed by summing its costs across the individual data points. As a consequence, once the costs of the individual data points are known, the parameter set's cost can be obtained for any of the cost functions related to one of the replicated datasets. The synergized bootstrap method put forward in this paper exploits these two features to accelerate the optimization procedures for re-sampling methods. It is applied to the non-parametric bootstrapping of the parameters of a univariate mixture model, of which the min-log-likelihood function can be shown to have multiple local minima, using the Differential Evolution heuristic as global optimizer. It is demonstrated that the synergized method can lead to incredible accelerations (up to 100-500 times faster) while being more accurate than the standard DE method.

Keywords: differential evolution, finite mixture models, maximum-likelihood optimization, non-parametric bootstrapping, re-sampling methods

1 Introduction

Re-sampling methods can be used for a wide variety of statistical inference tasks such as uncertainty assessment (e.g., estimating confidence intervals), hypothesis testing (estimating tail-area probabilities) or cross-validation (estimating prediction errors to perform model selection). These methods prove their usefulness even or specifically in the absence of analytic results or when specific conditions of the analytic method, such as normality, are unjustified (Adzhubei et al., 2010; Cox & Mann, 2008; Persson, Lindn, Unoson, & Elf, 2013; Ramaswamy,

Ross, Lander, & Golub, 2003; Turnbaugh et al., 2009). The flexibility of re-sampling methods has made them a popular tool. Most well-known methods are the bootstrap (Efron & Tibshirani, 1994), the permutation test (Good, 2000) and different kinds of cross-validation methods (Hastie, Tibshirani, & Friedman, 2016).

Re-sampling methods typically follow the same scheme. First, multiple datasets are generated by re-sampling from an original dataset. Then, a model is fitted to or a statistical test is conducted on all the re-sampled datasets. Ultimately, the results are combined and used, for instance, to assess the uncertainty of an estimator or to assess the robustness of a statistical test.

Notwithstanding the present-day computing power, a main disadvantage of re-sampling methods is still their computational burden (Mestdagh, Verdonck, Duisters, & Tuerlinckx, 2015). This burden is conspicuously present when iterative methods are required to maximize a likelihood or minimize a cost function for every re-sampled dataset. Typical variance and bias calculations require between 50 and 200 bootstrap replications. Confidence limits are more costly; they may require between 1,000 and 2,000 replications (Efron, 1987). Optimizers, especially global optimizers, go through multiple cost function evaluations during each iteration to update. Whenever the evaluation of the cost function becomes too costly, re-sampling methods become impractical.

Continued effort has been invested in alleviating the computational burden of re-sampling methods. The majority of these efforts focus on optimizing the process of fitting the model. This is achieved by, for example, reducing the required number of iterations for a single optimization (Andrews, 1999; Bringmann et al., 2013; Cawley & Talbot, 2008; Crainiceanu & Ruppert, 2004; Davidson & MacKinnon, 1999; Efron, 1990; Halekoh & Hjsgaard, 2014; Lippert et al., 2011; Maho et al., 2014; Samuh, Grilli, Rampichini, Salmaso, & Lunardon, 2012; Shaw et al., 2006; Zhou & Stephens, 2012, 2014), or splitting up the cost function into smaller, simpler parts (Hu & Kalbfleisch, 2000). Another approach that was proposed was aimed at improving the post-processing of bootstrap estimates which in favorable situations allows a significant reduction of the necessary number of bootstrap replications (Efron, 1990). Other improvements are custom-made for specific problems (Kleiner, Talwalkar, Sarkar, & Jordan, 2011; Stamatakis, Hoover, & Rougemont, 2008; Zeng & Lin, 2008). In general, these methods are not generic.

A special feature of re-sampling methods that can be more universally exploited is the large degree of overlap between re-sampled datasets. The re-sampled datasets all stem from the same origin and therefore have more or less the same characteristics. Therefore, when fitting the model to the re-sampled datasets, a similar task has to be executed over and over again. The fingerprint re-sampling method (Mestdagh et al., 2015) exploits this feature by learning the relation between data characteristics and the corresponding model estimates. Once this relation has been established, it can be used to suggest more appropriate starting points for the iterative estimation process. After a few estimations, the fingerprint method can lead to a significant acceleration of the estimation

procedures for the remaining re-sampled datasets. When the relation becomes sufficiently accurate, it may replace the estimation processes altogether.

The fingerprint method uses limited information to establish the relation between data characteristics and model parameters. Data characteristics are mapped to the final result of an optimization procedure, any information regarding the cost function obtained during the estimation is lost. This makes the fingerprint method susceptible to local minima. If the mapping between the data space and the model space is non-smooth or has discontinuities, the applicability of the fingerprint method can be called into question. The method has in fact only been tested for smooth, convex min-log-likelihood functions using local optimizers (Mestdagh et al., 2015). For such min-log-likelihood functions, the estimates of the re-sampled datasets typically lie closely together in the parameter space and the relation between data characteristics and estimates can nicely be established. Under these conditions, the bootstrap distributions are unimodal. If a cost function has multiple local minima, it is possible that two datasets with similar features (they lie close together in the data space) map to very different parameter configurations (they are far apart in the model space). This results in multimodalities in the bootstrap distributions and can significantly obscure the relation between data characteristics and model parameters. Local optimizers are not adequate for such problems, and the way to build the relation between the data characteristics and the model estimates becomes unclear.

The prepaid method (Mestdagh, Verdonck, Meers, Loossens, & Tuerlinckx, 2018) is another recent method that has been proposed for speeding up optimization procedures that also exploits the similarity of fitting procedures, but in a much broader context. The key idea of the prepaid method is that when the same model is used multiple times for different datasets, similar computations are required to arrive at estimations. The prepaid method capitalizes on this by creating a prepaid database in which for a regular grid across the parameter space, the model is extensively simulated. Once this database has been constructed and stored, any estimation problem can be solved on the spot using advanced interpolation techniques. The main advantage of the prepaid method over the fingerprint re-sampling method is the humongous amount of available information in the database to construct the relation between the data space and the model space, making it less susceptible to local minima. Unfortunately, setting up the prepaid grid is computationally too burdensome for a one-time re-sampling task.

In this paper, the key ideas of the fingerprint and prepaid method are exploited for re-sampling methods in a slightly different fashion. For a large class of cost functions, the cost of a specific set of model parameters can be computed for all cost functions in the bootstrap without the need for extra computations (e.g., because all cost functions consist of additive contributions of the same data points). By evaluating the costs of the original data points, any cost of a re-sampled dataset can be obtained. Hence, when a specific cost function in the bootstrap is being optimized, we can keep track of the different locations in the parameter space that have been visited and save those that are of interest

for the optimization of the other cost functions. In other words, we can construct prepaid databases on the fly which will help us with the initialization of the other optimization problems. In addition, when multiple cost functions in the bootstrap are being optimized at the same time, all visited locations in the parameter space and their corresponding costs for each of the functions can be made collectively available. As such, information can be exchanged between the different optimization procedures. The main advantage of this synergy is that due to the availability of collective information, the different cost functions can be scanned more rapidly and less cost function evaluations are required to reach the different global optima. Unlike the fingerprint method, the proposed method uses the prepaid databases that are constructed on the fly and that are optimized for each cost function independently. Hence, the method can be applied to cost functions with multiple local minima. As an example, the synergized bootstrap method described in this paper will be applied to the non-parametric bootstrapping of the parameter estimates of finite mixture models. There is no closed form expression for the optima of the min-log-likelihood function of a finite mixture model and the function can have multiple minima (see e.g., McLachlan & Peel, 2000). Finite mixture models therefore provide an ideal test case for the method. The Differential Evolution (DE) heuristic (Storn & Price, 1997) will be used as global optimizer to fit the mixture models.

2 Non-Parametric Bootstrapping

When fitting a model to data, a non-parametric bootstrap can be used to evaluate the uncertainty of the parameter estimates. The idea is to generate a number of new datasets (replications) based on the original and fit the model to each of them. That way, distributions of parameter estimates are obtained. These can then be used to compute standard errors or confidence bounds.

2.1 Re-sampling: Generating New Datasets

For a non-parametric bootstrap, a replicated dataset is created by sampling data points from the original dataset $\mathbf{y} = \{y_i\}^{i=1,...,n}$ with replacement until the sample size n is reached. Note that we will focus in this paper on i.i.d. data (i.e., $y_i \sim F$ for all i and all y_is are independent from one another). The replicated datasets will be denoted as $\mathbf{y}^j$, $j = 1, ..., B$, where B refers to the number of replications.

The constituents of the replicated datasets are the same as those of the original dataset, namely the data points y_i. The frequency of occurrence of the constituents, however, differs across replicated datasets. A replicated dataset $\mathbf{y}^j$ is fully defined in terms of the original data points y_i and the frequencies f_i^j with which they appear. The notation f_i^j is used to refer to the frequency of occurrence of the data point y_i in the replicated dataset $\mathbf{y}^j$.

2.2 Optimization: Finding the Estimates

To find the parameter estimates of a model given data $\mathbf{y}$, a cost function (like a least-squares function or a min-log-likelihood function) $\mathscr{F} : \theta \mapsto \mathscr{F}(\theta \,|\, \mathbf{y})$ has to be optimized. A cost function $\mathscr{F}$ relates a cost $\mathscr{F}(\theta \,|\, \mathbf{y})$ to every parameter set $\boldsymbol{\theta}$ given the data $\mathbf{y}$. The parameter estimates $\hat{\boldsymbol{\theta}}$ correspond to the parameter set $\boldsymbol{\theta}$ for which the cost is optimal. This set can be found at the global extremum of the cost function. For least-squares and min-log-likelihood functions, this extremum corresponds to a global minimum.

The functional form of the cost function depends on the particular model. In this paper, we are primarily concerned with cost functions that have multiple local extrema (minima or maxima, depending on the context). For such functions, local optimization algorithms are inadequate and instead global optimization heuristics are required.

We will rely on the Differential Evolution (DE) heuristic for global optimization because it is a simple, reliable and all-round optimizer (Storn & Price, 1997) that has got the required properties for the synergized method proposed in this paper to work. Furthermore, it has been used for fitting finite mixture models (Boonthiem, Boonta, & Klongdee, 2017; Kwedlo, 2014). The ideas underlying the DE method are nevertheless more broadly applicable to similar kinds of optimizers. Here, we briefly explain the ideas behind DE. For a more elaborate description, see (Storn & Price, 1997).

Differential Evolution. Differential Evolution is a parallel, stochastic, direct search method for solving continuous optimization problems (Storn & Price, 1997). It relies on a population $\mathscr{P}(g)$ of NP model parameter vectors or 'agents' $\boldsymbol{\chi}_a(g)$, $a = 1, ..., NP$. These agents are updated over generations g. The population size NP does not change during the optimization.

A DE optimization starts by sampling an initial population $\mathscr{P}(0)$ of NP agents $\boldsymbol{\chi}_a(g)$ from a prior distribution on the parameter space. This prior distribution should cover the entire search space (the part of the parameter space that is of interest). The cost $\mathscr{F}(\boldsymbol{\chi}_a \,|\, \mathbf{y})$ of each agent is also computed.

Subsequently, the populations are iteratively updated. Updating a population happens in different stages. First, NP mutant vectors are constructed by adding the weighted difference of two agents to a third. All of the triplets that constitute a mutant are unique. In total, there are as many mutant vectors as there are agents in the population. Second, a crossing-over between agents and mutant vectors takes place. During this procedure, the parameters of an agent are intermixed with the parameters of one of the mutant vectors. The resulting vectors make up the offspring and will be referred to as children. Third, the cost of each child is compared to that of the agent to which it is kin (the parent; i.e., the agent of which he inherited part of its parameters). If the child's cost is more optimal, it replaces its parent in the population. Otherwise, the parent lives on.

The total cost of the population $\mathscr{P}(g)$ can only improve from one generation to the next. Agents that are more optimal will attract other agents. In doing so, the population increasingly focuses on regions in the parameter space that

are more interesting for finding a global optimum. The spread of the population naturally shrinks across generations and so do the weighted differences between the agents. As such, the optimization scheme naturally adapts from that of a global search into that of a local search.

Algorithm 1: standard DE

```
input  : data and cost function
output: estimates, cost
agents = random_initialization(NP);
agents_costs = cost_function(data, agents);
for g = 1 to n_iter do
    children = mutate_and_crossover(agents);
    children_costs = cost_function(data, children);
    agents = compare_and_eliminate(agents, children, agents_costs,
      children_costs);
    agents_costs = update_costs(agents_costs, children_costs);
end
best_agent_position = find_optimal(agents_costs);
estimates = get_agent(agents, best_agent_position);
cost = get_cost(costs, best_agent_position);
```

Multiple Cost Functions. In the context of non-parametric bootstrapping, there is a cost function $\mathscr{F}^j : \theta \mapsto \mathscr{F}(\theta \,|\, \mathbf{y}^j)$ associated to every replicated dataset $\mathbf{y}^j$. These cost functions are entirely independent from one another. To find the optimum of each function, B different optimization procedures have to be run – exactly as many optimizations as there are replications.

Given an agent $\boldsymbol{\chi}_a^j$ of the population $\mathscr{P}^j$. In a traditional DE setup, the only relevant cost to know is $\mathscr{F}(\boldsymbol{\chi}_a^j \,|\, \mathbf{y}^j)$. When offspring is created, it is this cost that is used during the selection stage of the updating step. The costs $\mathscr{F}(\boldsymbol{\chi}_a^j \,|\, \mathbf{y}^k)$ with $j \neq k$ are never computed. Yet, such 'external' costs can hold relevant information.

For one, if both the usual (internal) costs and the external costs are available to all populations, the amount of information on the different cost functions increases by a factor B. Instead of only NP function values (costs), $B \times NP$ function values are known per cost function. This significantly augments the sheer search power, making it easier to uncover the relevant regions for the different cost functions.

An additional benefit in the context of non-parametric bootstrapping is that the cost functions all look alike. The functional form of the cost functions is the same for all replicated datasets, since it is determined by the model. They only differ in the dataset they are associated to; the function $\mathscr{F}^j$ is associated to the replicated dataset $\mathbf{y}^j$. Replicated datasets show a lot of overlap since they are all derived from the same origin $\mathbf{y}$; hence the similarity between cost functions.

As a consequence, the different populations will be attracted towards the same regions in the parameter space and agents from a specific population can

therefore learn a lot from external agents, especially during the global search stage of the optimization. Only when the populations start homing in on their specific global minimum will the gains that can be made by considering external agents reduce.

Yet, computing the agents' costs is for many practical applications the most time-consuming task during the optimization. It is therefore generally not a good strategy to compute costs from obtained less promising external agents.

2.3 Collective Information

For a specific class of cost functions, however, external costs can be obtained for free given that the re-sampled datasets are constructed from the same original data points. There is a large class of cost functions with the property that the cost of a parameter set $\boldsymbol{\theta}$ for an entire dataset $\mathbf{y}$ is equal to the sum of the costs of the individual data points that make up the dataset (here we rely on the i.i.d. assumption made above):

$$\mathscr{F}(\boldsymbol{\theta} \,|\, \mathbf{y}) = \sum_{i=1}^{n} \mathscr{F}(\boldsymbol{\theta} \,|\, y_i).$$

By extension, the cost $\mathscr{F}(\boldsymbol{\theta} \,|\, \mathbf{y}^j)$ can straightforwardly be constructed from the costs $\mathscr{F}(\boldsymbol{\theta} \,|\, y_i)$ of the individual data points as well:

$$\mathscr{F}(\boldsymbol{\theta} \,|\, \mathbf{y}^j) = \sum_{i=1}^{n} f_i^j \, \mathscr{F}(\boldsymbol{\theta} \,|\, y_i).$$

The cost of the parameter set $\boldsymbol{\theta}$ for a re-sampled dataset $\mathbf{y}^j$ is simply a weighted sum of the costs of the individual data points that make up the original dataset, where the weights are determined by the frequency of occurrence of the data points.

In order to update all DE populations of an entire non-parametric bootstrap, $n \times B \times NP$ individual costs have to be computed – n individual costs per agent. From these, the costs $\mathscr{F}(\boldsymbol{\chi}_a^j \,|\, \mathbf{y}^j)$ which are necessary for the selection procedure can be constructed. The external costs $\mathscr{F}(\boldsymbol{\chi}_a^k \,|\, \mathbf{y}^j)$ with $j \neq k$ can also be constructed from these, without the need for extra computations. In a traditional DE optimization, this external information is ignored but in the next section we will show how this information can be put to good use.

3 The Synergized Bootstrap

Instead of initializing a different population $\mathscr{P}^j$ for every cost function $\mathscr{F}^j$ (as would be the standard procedure for the traditional DE), only one population $\mathscr{P}$ of NP agents is initialized. The costs of these agents are computed for all the cost functions $\mathscr{F}^j$, resulting in $B \times NP$ costs. Note that only $n \times NP$ computations are required for this (not $n \times B \times NP$, as for the traditional DE).

The agents of the population $\mathscr{P}$ serve as initial generation for each optimization: $\forall\, j \in \{1, ..., B\} : \mathscr{P}^j(0) = \mathscr{P}$.

The agents of one of the populations, say $\mathscr{P}^1$, start updating like they would normally do. However, for each generation $\mathscr{P}^1(g)$, the children $\boldsymbol{\upsilon}_a^1$ are evaluated in all the cost functions (again, this does not require extra computations). If the cost $\mathscr{F}(\boldsymbol{\nu}_a^1 \,|\, \boldsymbol{y}^j)$ of the child $\boldsymbol{\nu}_a^1$ is better than the cost $\mathscr{F}(\boldsymbol{\chi}_a^j \,|\, \boldsymbol{y}^j)$ of the current agent $\boldsymbol{\chi}_a^j$ in population $\mathscr{P}^j$, the agent is replaced (the agent $\boldsymbol{\chi}_a^j$ and the child $\boldsymbol{\nu}_a^1$ have the same sub-index a – they have the same position within their own respective populations – but they are not parent and child because they come from different populations). So, although only NP agents are actively evolving, all agents of all populations are being updated and it is possible that some agents appear multiple times across populations.

This initial updating scheme is sufficient for establishing the regions in the parameter space that are of interest for the different cost functions. The populations $\mathscr{P}^j$ only retain agents that optimize their current cost. The agents that are being retained can vary across populations, which still enables the populations to focus on their particular problem. Essentially, a prepaid database is being constructed for each cost function, each consisting of its own NP parameter sets and their related costs. The initial global search of the DE optimizations, which is quasi-random anyways, is shared across populations and thus the computational cost of this 'burn-in' is significantly reduced.

At some point, there will be populations that no longer adopt children from the population $\mathscr{P}^1$ because its agents are drifting away from their particular regions of interest. Typically, the populations whose cost functions differ most from $\mathscr{F}^1$ will diverge earlier. The first time a population, say $\mathscr{P}^2$, is unable to adopt children for several generations in a row, its agents are also made to actively evolve. From then onward, there are two DE populations independently breeding children. All these children are still evaluated in all cost functions (because this does not require any extra computations). The supply of children which the populations can adopt to improve their prepaid database doubles. The idea is that populations corresponding to cost functions that have more in common with $\mathscr{F}^2$ might still be able to learn from the agents of the population $\mathscr{P}^2$.

Once there is another population that can no longer learn from the two populations, its agents are also made to evolve. One after another, the populations start their own individual evolution. The rate at which populations launch their own evolutionary updating strongly depends on the similarity between cost functions. In the worst case, agents cannot learn much from external information and the populations all start their own search almost immediately. In that case, B different DE optimizations are being done. Hence, the computation time of the synergized bootstrap is at worst equal to that of a traditional bootstrap where all B optimizations are done independently.

It can happen that multiple populations fail to update for several generations in a row. This implies that these populations require information that is different from the information provided by the currently active populations. It is however possible that several of these populations require similar information. Hence, it

would suffice to just activate a selection of them. The other populations that were not activated can then learn from those that have been activated. As such, the rate at which the computational burden increases can be limited.

Populations that have started evolving can also still adopt children from other populations. This is an additional advantage over the traditional DE. Populations that have converged to a local minimum can still be corrected through the adoption of external children.

Algorithm 2: synergized DE

input : data, bootstrap frequencies and cost function
output: estimates, cost

```
agents = random_initialization(NP);
agents = replicate_agents(B);
individual_costs = cost_function(data, agents);
agents_costs = weighted_sum(individual_costs, frequencies);
n_active_populations = 1;
for i = 1 to B do
    failed_counter(i) = 0;
end
for g = 1 to n_iter do
    for j = 1 to B do
        updated(j) = false;
    end
    for j = 1 to n_active_populations do
        active_agents = get_population(agents, j);
        children = mutate_and_crossover(active_agents);
        children_individual_costs = cost_function(data, children);
        for k = 1 to B do
            local_frequencies = get_frequencies(frequencies, k);
            children_local_costs = weighted_sum(children_individual_costs,
              local_frequencies);
            local_agents2 = get_population(agents, k);
            local_agents_costs = get_costs(agents_costs, k);
            local_agents = compare_and_eliminate(local_agents2, children,
              local_agents_costs, children_local_costs);
            local_agents_costs = update_costs(local_agents_costs,
              children_local_costs);
            if not equals(local_agents, local_agents2) then
                updated(k) = true;
            end
            agents = insert_population(local_agents, agents, k);
            agents_costs = insert_costs(local_agents_costs, agents_costs, k);
        end
    end
    n_activated_populations = 0;
    for j = n_active_populations + 1 to B do
        if updated(j) == false then
            failed_counter(j) = failed_counter(j) + 1;
        else
            failed_counter(j) = 0;
        end
        if failed_counter(j) > max_fails and n_activated_populations <
          max_activations then
            n_active_populations = n_active_populations + 1;
            agents = swap_populations(agents, n_active_populations, j);
            agents_costs = swap_costs(agents_costs, n_active_populations, j);
            n_activated_populations = n_activated_populations + 1;
        end
    end
end
```

Algorithm 2: synergized DE (continued)

```
for i = 1 to B do
    local_agents = get_population(agents, i);
    local_agents_costs = get_costs(agents_costs, i);
    best_agent_position = find_optimal(current_agents_costs);
    local_best = get_agent(local_agents, best_agent_position);
    estimates = concatenate(estimates, local_best);
    cost(k) = get_cost(costs, best_agent_position);
end
```

4 Application: Finite Mixtures

The ideas described above were applied to the non-parametric bootstrapping of the parameters of univariate finite mixture models. The probability density function $p(x)$ of a univariate finite mixture is defined as

$$p(x) = \sum_{m=1}^{N} \omega_m \, \phi_m(x),$$

where $\phi_m(x)$ denote univariate normal density functions (the components),

$$\phi_m(x) = \frac{1}{\sqrt{2\pi\sigma_m^2}} \, e^{-\frac{(x-\mu_m)^2}{2\sigma_m^2}},$$

and $\omega_m \geq 0$ denote weights which sum up to one. The min-log-likelihood function of the mixture model is given by

$$\mathscr{L}(\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\omega} \,|\, \mathbf{y}) = -\sum_{i=1}^{n} \ln\left(\sum_{m=1}^{N} \frac{\omega_m}{\sqrt{2\pi\sigma_m^2}} \, e^{-\frac{(y_i-\mu_m)^2}{2\sigma_m^2}} \right). \tag{1}$$

By minimizing this function, the parameter estimates of the finite mixture can be found for a given dataset $\mathbf{y} = \{y_i\}^{i=1,\ldots,n}$.

The min-log-likelihood function (1) can have multiple local minima. In Figure 1, an example of such a local minimum is shown. In Panel (a), the two-component mixture distribution $p(x)$ is drawn from which 50 data points were simulated. A histogram of the data points is shown in Panel (b). The distribution corresponding to the global minimum is indicated as well. In Panel (c), the histogram is drawn again, but this time in combination with a distribution corresponding to a local minimum of the min-log-likelihood function. This distribution was obtained using a local minimizer.

Because of the occurrence of local minima and the low computational cost for optimization, the univariate finite mixture model provides a suitable application to demonstrate the synergized bootstrap method.

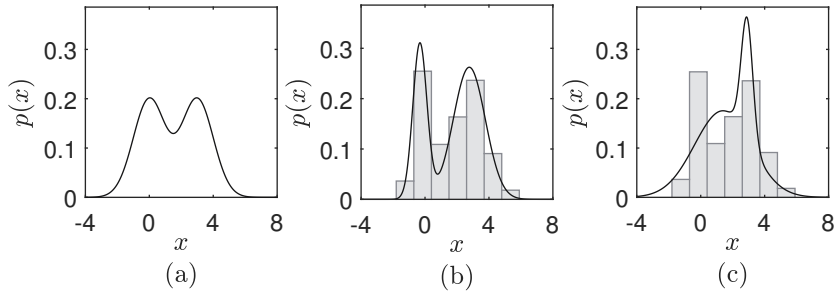


Fig. 1. Example of a local minimum. Panel (a) depicts the probability density function $p(x)$ of the finite mixture model that was used to simulate data. Panel (b) depicts the histogram of the 50 simulated data points $\{y_i\}^{i=1,...,50}$. The probability density function corresponding to the global optimum of the min-log-likelihood function is drawn as well. Panel (c) again depicts the histogram of the 50 simulated data points. Here, the probability density function corresponding to a local optimum of the min-log-likelihood function is shown.

4.1 Simulating Data

For our application, we simulated 100 different datasets $\mathbf{y}^j$ $(j = 1, ..., 100)$ from 100 distinct two-component finite mixture distributions $p^j(x)$. Each of the datasets consisted of $n^j = 50$ data points.

Generating Model Parameters. The first component of each finite mixture distribution corresponded to the standard normal distribution ($\mu_1^j = 0$, $\sigma_1^j = 1$). The weights ω_1^j were sampled from a uniform distribution

$$\omega_1^j \sim U(.25, .75).$$

The means μ_2^j and the standard deviations σ_2^j of the second components were also sampled from uniform distributions:

$$\mu_2^j \sim U(2, 4),$$
$$\sigma_2^j \sim U(.5, 1.5).$$

To ensure that the functions $p^j(x)$ were proper probability density functions, the weights ω_2^j of the second components were constructed as follows:

$$\omega_2^j = 1 - \omega_1^j.$$

Generating Data. To generate 50 data points from the two-components finite mixture distribution $p^j(x)$, first of all 50 numbers r_i^j (i = 1,...,50) were randomly drawn from a uniform distribution $U(0, 1)$. For each of these r_i^j values, a normally distributed random number y_i^j was sampled either from the first component of

the finite mixture $p^j(x)$ or from the second component. If $r_i^j < \omega_1^j$, then y_i^j was sampled from the first component, otherwise it was sampled from the second component.

Doing this for all 100 sampled parameters led to 100 datasets $\mathbf{y}^j$ of which the data points y_i^j were appropriately distributed according to their corresponding two-component finite mixture distribution $p^j(x)$.

Algorithm 3: generating data

input : number of required data points and model parameters (means, standard deviations, weights)
output: data y distributed according to mixture distribution

```
for i = 1 to n_data_points do
    r = uniform_random_number(0,1);
    if r < weights(1) then
        y(i) = mean(1) + standard_deviation(1) * normal_random_number();
    else
        y(i) = mean(2) + standard_deviation(2) * normal_random_number();
    end
end
```

4.2 Simulation Study

Non-parametric bootstrapping. For each of the 100 datasets $\mathbf{y}^j$, 1,000 bootstrap replicates $\mathbf{y}^{jk}$ $(k = 1, ..., 1000)$ were created by drawing 50 data points (exactly as many as there are in the original datasets) with replacement from the original datasets $\mathbf{y}^j$. In doing so, 100,000 datasets were constructed in total.

Algorithm 4: non-parametric bootstrapping

input : data
output: re-sampled dataset y

```
for i = 1 to n_data_points do
    r = uniform_random_integer(1,n_data_points);
    y(i) = data(r);
end
```

Optimization. Associated to every replicated dataset $\mathbf{y}^{jk}$ there is a min-log-likelihood function $\mathscr{L}(\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\omega} \,|\, \mathbf{y}^{jk})$. The parameter estimates that are obtained by minimizing the 1,000 min-log-likelihood functions $\mathscr{L}^{jk}$ corresponding to a specific original dataset $\mathbf{y}^j$ can be used to construct bootstrap confidence intervals around the parameter estimates $(\hat{\boldsymbol{\mu}}^j, \hat{\boldsymbol{\sigma}}^j, \hat{\boldsymbol{\omega}}^j)$ obtained by minimizing the min-log-likelihood function $\mathscr{L}(\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\omega} \,|\, \mathbf{y}^j)$. For our simulation study, the 1,000 min-log-likelihood functions $\mathscr{L}^{jk}$ corresponding to a specific original dataset $\mathbf{y}^j$ were minimized in parallel. In what follows, an optimization (procedure) will refer to the minimization of the 1,000 min-log-likelihood functions corresponding to a single original dataset $\mathbf{y}^j$.

All optimizations were done using the standard DE optimizer as well as the synergized DE method. For both, the crossover rate was set equal to .3. Moreover, they were run using the traditional *DE/rand/2/bin* configuration (see Storn & Price, 1997). In other words, we opted for the random DE mutation strategy and the binomial crossover scheme. The 2 refers to the number of agents that made up the weighted differences of the mutants. The weighing factors of the weighted differences for the mutations were randomly sampled from a uniform distribution $U(0, 2)$, as was done by Mohamed, Sabry, and Khorshid (2012).

Every optimization procedure was done with $NP = 100$ agents per population which is amply sufficient according to DE standards (see Storn & Price, 1997). To study the consequence of significantly reducing the population size, each simulation procedure was also repeated with $NP = 10$ agents per population. Hence, in total 400 different optimization procedures were run, four for each original dataset $\mathbf{y}^j$.

The synergized DE required some additional specifications. Populations were activated if they failed to update for three consecutive iterations. Furthermore, only five populations could simultaneously be activated.

Accuracy and Benchmarks. There are 1,000 DE populations at play during a single optimization procedure. The accuracy of a DE population is defined as the difference between its best agent's min-log-likelihood (the smallest value in the population) and the min-log-likelihood of the global minimum of the function that is being minimized by the agents of this population. The bootstrap accuracy is the average of all 1,000 accuracies in the optimization. Both the accuracy and the bootstrap accuracy are functions of the number of DE iterations; they become smaller the more the populations evolve towards their global minimum.

In order to be able to determine the accuracies, the global minima of the min-log-likelihood functions have to be known prior to doing the optimizations. We determined these by taking the best of three independent standard DE optimizations for each of the 100,000 min-log-likelihood functions in the simulation study. The setup of these DE optimizers was the same as described above. The number of agents NP was chosen to be 100 and the maximum number of iterations was set equal to 3,000.

Comparing Methods. Every optimization procedure was repeated four times: the standard DE algorithm was run with $NP = 100$ and $NP = 10$ agents per population and so was the synergized DE algorithm. For each of them, the bootstrap accuracy was tracked as a function of the number of cost function evaluations that were required to attain this specific accuracy. At every instance in the evolution, the accuracy of the different procedures could be compared. In all cases, the optimizations ran until a satisfactory accuracy was reached.

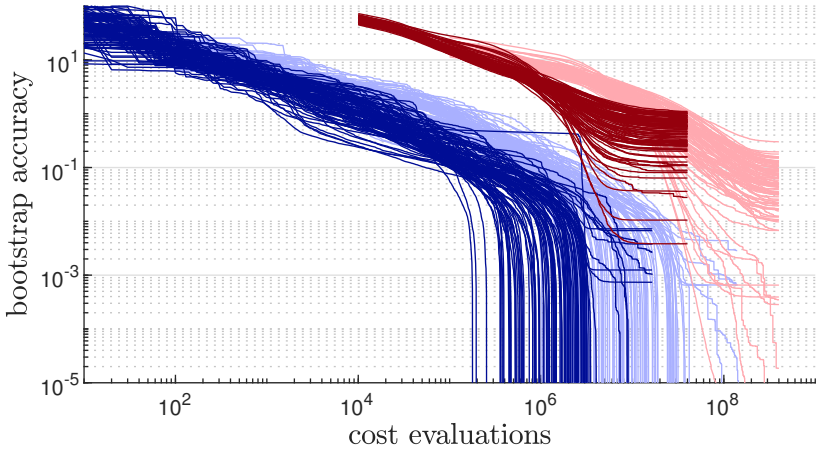


Fig. 2. Comparison of the standard DE bootstrap and the synergized bootstrap. The bootstrap accuracy is depicted in function of the number of cost function evaluations for the standard DE method (red) and the synergized method (blue). The bootstrap accuracy refers to the average difference between the min-log-likelihoods of the best agents of the populations and those of the corresponding global minima. Lighter lines correspond to populations of $NP = 100$ agents and darker lines to populations of $NP = 10$ agents.

5 Results

The results of the simulation study are summarized in Figure 2 which depicts the bootstrap accuracies as a function of the number of executed cost function evaluations. Red lines correspond to the optimizations with the standard DE method and blue lines correspond to the optimizations with the synergized method. Lighter and darker lines correspond to populations with $NP = 100$ and $NP = 10$ agents respectively. The standard DE bootstraps ran for 4,000 iterations, the synergized bootstraps ran for 2,000 iterations.

Looking first at the red lines, we can see that reducing NP is not favorable for the standard DE method for this particular problem. The cost of updating an entire population is smaller for the darker lines, and initially the darker lines converge faster, but they typically do not reach the same level of accuracy as the lighter lines. It is known that smaller populations can evolve more rapidly, but for non-convex cost functions with many local minima this increases the risk of getting stuck in local minima because the spread of the agents in the population shrinks too quickly. In the remainder of this discussion, we will only compare to the standard DE with $NP = 100$ (light red).

The synergized method converges a lot faster than the standard DE method, regardless of the population size. The lines corresponding to synergized optimizations start earlier, because less cost function evaluations are required for initialization and also during the global search stage of the optimization process (not yet all populations are actively evolving). For the red lines, $B \times NP = 10^5$

evaluations are needed to initialize the optimization procedure, which determines the starting points of the lines. Moreover, for the standard DE every update of a generation requires 10^5 evaluations. The light blue lines ($NP = 100$) reach an accuracy of about 10^{-1} approximately 100 times faster than the red lines. The dark blue lines ($NP = 10$) reach this accuracy approximately 500 times faster.

Not only do the synergized bootstraps converge faster, they are more accurate than the standard method. The bootstrap accuracy of the DE method only drops to about 10^{-1} to $2 \cdot 10^{-2}$, indicating that for each simulation several of the bootstrap populations get stuck in local minima. For the synergized method, the accuracy generally drops below 10^{-5}. Reducing the number of agents per population does not affect this accuracy for the synergized method (unlike for the standard DE). The fact that populations that got stuck in local minima can still be corrected by others seems to have an advantageous effect on the global performance of the optimizer.

In summary, the synergized method is both faster and more accurate than the standard DE method. It also requires less agents per population to work, which reduces the computation time even more.

6 Discussion

In this paper, we introduced a new method for speeding up the optimization procedures in the context of re-sampling methods and we applied the method to the non-parametric bootstrapping of the parameters of a univariate two-component mixture model. Just like the fingerprint method (Mestdagh et al., 2015), the proposed synergized bootstrap method exploits the fact that the characteristics of the re-sampled datasets are very similar, and so are the related optimization problems. Moreover, it relies on the fact that, for a large class of cost functions, the costs of a parameter set for the different cost functions can be obtained at a very low computing cost. This allows combining resources across the different optimization procedures during run time without extra burden. By letting the different optimization processes communicate with one another, the computation time can significantly be reduced since the optimization problems are very similar.

For the application on the mixture models, the results indicated that the synergized Differential Evolution (DE) can be approximately 100 to 500 times faster than the standard DE. Furthermore, because the synergized DE can correct populations that got stuck in local minima by relying on information from the other populations, it can generally achieve a better accuracy.

The synergized bootstrap method is more widely applicable than the fingerprint method because it is not limited to smooth, convex cost functions. A limitation of the synergized method is, however, the need for cost functions of which the costs of the different data points can be computed independently. This class of functions is nonetheless very large and these functions are frequently encountered in practical applications because independence of data points is a common assumption. Although this requirement is sufficient for the procedure

to work, it is not strictly necessary. Another case where the synergized method could be used is for optimization routines where there is separability of the initial model calculations and the final comparison to data in the objective function. If, for instance, the entire probability density function is calculated for an agent (e.g., as a sufficiently fine discrete distribution) independent of the data, it can be recycled for use with other (re-sampled) datasets (and hence for other cost functions) for practically no extra cost – here the bottleneck is the preparatory calculation of the probability density function, not its evaluation. Some examples of algorithms that first calculate/simulate the entire probability density function and then in a second, separable step compare this with the data are D*M optimization for diffusion models of choice response time (Verdonck & Tuerlinckx, 2016) and simulated likelihood approaches (Verdonck & Tuerlinckx, 2014).

References

Adzhubei, I. A., Schmidt, S., Peshkin, L., Ramensky, V. E., Gerasimova, A., Bork, P., ... Sunyaev, S. R. (2010). A method and server for predicting damaging missense mutations. *Nature Methods*, *7*(4), 248–249. doi: 10.1038/nmeth0410-248

Andrews, D. W. K. (1999). *Higher-Order Improvements of a Computationally Attractive-Step Bootstrap for Extremum Estimators* (Tech. Rep. No. 1230). Cowles Foundation for Research in Economics, Yale University.

Boonthiem, S., Boonta, S., & Klongdee, W. (2017). A Differential Evolution Algorithm with Adaptive Controlling Weighted Parameter for Finite Mixture Model of Some Fire Insurance Data in Thailand. *SNRU Journal of Science and Technology*, *9*, 491–501.

Bringmann, L. F., Vissers, N., Wichers, M., Geschwind, N., Kuppens, P., Peeters, F., ... Tuerlinckx, F. (2013). A Network Approach to Psychopathology: New Insights into Clinical Longitudinal Data. *PLOS ONE*, *8*(4), e60188. doi: 10.1371/journal.pone.0060188

Cawley, G. C., & Talbot, N. L. C. (2008). Efficient approximate leave-one-out cross-validation for kernel logistic regression. *Machine Learning*, *71*(2-3), 243–264. doi: 10.1007/s10994-008-5055-9

Cox, J., & Mann, M. (2008). MaxQuant enables high peptide identification rates, individualized p.p.b.-range mass accuracies and proteome-wide protein quantification. *Nature Biotechnology*, *26*(12), 1367–1372. doi: 10.1038/nbt.1511

Crainiceanu, C. M., & Ruppert, D. (2004). Likelihood ratio tests in linear mixed models with one variance component. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, *66*(1), 165–185. doi: 10.1111/j.1467-9868.2004.00438.x

Davidson, R., & MacKinnon, J. G. (1999). Bootstrap Testing in Nonlinear Models. *International Economic Review*, *40*(2), 487–508.

Efron, B. (1987). Better Bootstrap Confidence Intervals. *Journal of the American Statistical Association*, *82*(397), 171–185. doi: 10.1080/01621459.1987.10478410

Efron, B. (1990). More Efficient Bootstrap Computations. *Journal of the American Statistical Association*, *85*(409), 79–89. doi: 10.1080/01621459.1990.10475309

Efron, B., & Tibshirani, R. J. (1994). *An Introduction to the Bootstrap*. CRC Press. (Google-Books-ID: gLlpIUxRntoC)

Good, P. I. (2000). *Permutation Tests: A Practical Guide to Resampling Methods for Testing Hypotheses*. Springer.

Halekoh, U., & Hjsgaard, S. (2014). A Kenward-Roger approximation and parametric bootstrap methods for tests in linear mixed models: The R Package pbkrtest. *Journal of Statistical Software*, *59*(9), 1–32. doi: 10.18637/jss.v059.i09

Hastie, T., Tibshirani, R., & Friedman, J. (2016). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition* (2nd edition ed.). New York, NY: Springer.

Hu, F., & Kalbfleisch, J. D. (2000). The estimating function bootstrap. *Canadian Journal of Statistics*, *28*(3), 449–499.

Kleiner, A., Talwalkar, A., Sarkar, P., & Jordan, M. I. (2011). A Scalable Bootstrap for Massive Data. *arXiv:1112.5016 [stat]*. (arXiv: 1112.5016)

Kwedlo, W. (2014). Etimation of Parameters of Gaussian Mixture Models by a Hybrid Method Combining a Self-Adaptive Differential Evolution with the EM Algorithm. *Advances in Computer Science Research*, *11*, 109–123.

Lippert, C., Listgarten, J., Liu, Y., Kadie, C. M., Davidson, R. I., & Heckerman, D. (2011). FaST linear mixed models for genome-wide association studies. *Nature Methods*, *8*(10), 833. doi: 10.1038/nmeth.1681

Maho, Y. L., Whittington, J. D., Hanuise, N., Pereira, L., Boureau, M., Brucker, M., ... Bohec, C. L. (2014). Rovers minimize human disturbance in research on wild animals. *Nature Methods*, *11*(12), 1242. doi: 10.1038/nmeth.3173

McLachlan, G., & Peel, D. (2000). *Finite Mixture Models* (1edition ed.). New York: Wiley-Interscience.

Mestdagh, M., Verdonck, S., Duisters, K., & Tuerlinckx, F. (2015). Fingerprint resampling: A generic method for efficient resampling. *Scientific Reports*, *5*, 16970. doi: 10.1038/srep16970

Mestdagh, M., Verdonck, S., Meers, K., Loossens, T., & Tuerlinckx, F. (2018). Prepaid parameter estimation without likelihoods. *arXiv:1812.09799 [stat]*. (arXiv: 1812.09799)

Mohamed, A. W., Sabry, H. Z., & Khorshid, M. (2012). An alternative differential evolution algorithm for global optimization. *Journal of Advanced Research*, *3*(2), 149–165. doi: 10.1016/j.jare.2011.06.004

Persson, F., Lindn, M., Unoson, C., & Elf, J. (2013). Extracting intracellular diffusive states and transition rates from single-molecule tracking data. *Nature Methods*, *10*(3), 265. doi: 10.1038/nmeth.2367

Ramaswamy, S., Ross, K. N., Lander, E. S., & Golub, T. R. (2003). A molecular signature of metastasis in primary solid tumors. *Nature Genetics*, *33*(1), 49–54. doi: 10.1038/ng1060

Samuh, M. H., Grilli, L., Rampichini, C., Salmaso, L., & Lunardon, N. (2012). The Use of Permutation Tests for Variance Components in Linear Mixed Models. *Communications in Statistics - Theory and Methods*, *41*(16-17), 3020–3029. doi: 10.1080/03610926.2011.587933

Shaw, P., Greenstein, D., Lerch, J., Clasen, L., Lenroot, R., Gogtay, N., ... Giedd, J. (2006). Intellectual ability and cortical development in children and adolescents. *Nature*, *440*(7084), 676. doi: 10.1038/nature04513

Stamatakis, A., Hoover, P., & Rougemont, J. (2008). A rapid bootstrap algorithm for the RAxML Web servers. *Systematic Biology*, *57*(5), 758–771. doi: 10.1080/10635150802429642

Storn, R., & Price, K. (1997). Differential Evolution A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *Journal of Global Optimization*, *11*(4), 341–359. doi: 10.1023/A:1008202821328

Turnbaugh, P. J., Hamady, M., Yatsunenko, T., Cantarel, B. L., Duncan, A., Ley, R. E., ... Gordon, J. I. (2009). A core gut microbiome in obese and lean twins. *Nature*, *457*(7228), 480. doi: 10.1038/nature07540

Verdonck, S., & Tuerlinckx, F. (2014). The Ising Decision Maker: A binary stochastic network for choice response time. *Psychological Review*, *121*(3), 422–462. doi: 10.1037/a0037012

Verdonck, S., & Tuerlinckx, F. (2016). Factoring out nondecision time in choice reaction time data: Theory and implications. *Psychological Review*, *123*(2), 208–218. doi: 10.1037/rev0000019

Zeng, D., & Lin, D. Y. (2008). Efficient resampling methods for nonsmooth estimating functions. *Biostatistics (Oxford, England)*, *9*(2), 355–363. doi: 10.1093/biostatistics/kxm034

Zhou, X., & Stephens, M. (2012). Genome-wide efficient mixed-model analysis for association studies. *Nature Genetics*, *44*(7), 821–824. doi: 10.1038/ng.2310

Zhou, X., & Stephens, M. (2014). Efficient multivariate linear mixed model algorithms for genome-wide association studies. *Nature Methods*, *11*(4), 407. doi: 10.1038/nmeth.2848