HAMABS: Estimation of Discrete Choice Models with Hybrid Stochastic Adaptive Batch Size Algorithms

Gael Lederrey [*] Virginie Lurkin [†] Tim Hillel [*]
Michel Bierlaire [*]

December 5, 2019

Report TRANSP-OR 191213
Transport and Mobility Laboratory
School of Architecture, Civil and Environmental Engineering
Ecole Polytechnique Fédérale de Lausanne
http://transp-or.epfl.ch

[*]École Polytechnique Fédérale de Lausanne (EPFL), School of Architecture, Civil and Environmental Engineering (ENAC), Transport and Mobility Laboratory, Switzerland, {gael.lederrey,tim.hillel,michel.bierlaire}@epfl.ch

[†]Eindhoven University of Technology, Department of Industrial Engineering & Innovation Science, The Netherlands, v.j.c.lurkin@tue.nl

Abstract

The emergence of Big Data opened research to new perspectives for the discrete choice community. While the Machine Learning (ML) community has been thriving in finding ways to deal with such amount of data, the choice modeling community has not yet fully grasped the potential of Big Data. In this article, we provide new ways of dealing with extensive datasets in the context of Discrete Choice Models (DCMs). We achieved this by creating new efficient stochastic optimization algorithms. We develop these algorithms based on three major contributions: the use of a stochastic Hessian, the modification of the batch size, and a change of optimization algorithm depending on the batch size. We propose the HAMABS algorithm, a hybrid adaptive batch size stochastic method. We compare this algorithm with fourteen other ones, on ten Multinomial logit models. This algorithm speeds up the optimization time by a factor of 23 on the largest model. Therefore, by replacing the standard optimization methods in state-of-the-art DCMs software, researchers and practitioners can significantly reduce the time required for utility specification.

Keywords Optimization, Discrete Choice Models, Stochasticity, Adaptive Batch Size, Hybridization

List of acronyms

AMABS Adaptive Moving Average Batch Size. 13–16, 20–23, 32

BFGS Broyden-Fletcher-Goldfarb-Shanno. 4, 10, 11, 15, 16, 20, 23, 30

CNN Convolutional Neural Network. 3

DCM Discrete Choice Model. 3, 5, 6, 10, 15–17, 32, 33

FFNN Feed-Forward Neural Network. 4

LPMC London Passenger Mode Choice. 15–17, 24, 25, 27–32

ML Machine Learning. 2–6

MNL MultiNomial Logit. 16

MTMC Mobility and Transport MicroCensus. 16, 17, 24, 33

SGD Stochastic Gradient Descent. 6, 8, 12

VoT Value of Time. 3

WMA Window Moving Average. 12

1 Introduction

The availability of more and more data for choice analysis is both a blessing and a curse. On the one hand, this data provides analysts with great wealth of behavioral information. On the other hand, processing the increasingly large and complex datasets to estimate choice models presents new computational challenges. The Machine Learning (ML) community has been thriving in dealing with vast amounts of data. It therefore seems natural to investigate ML estimation algorithms to estimate choice models.

The central ingredient of these algorithms is the *stochastic gradient*. It consists in approximating the gradient of the log likelihood function (or any goodness of fit measure) using only a small subset of the data set. The gradient is said to be stochastic because the subset of data used to calculate it is drawn randomly from the full data set. A version of the steepest descent algorithm using this stochastic gradient is then applied to maximize the log likelihood function. Many variants have been proposed around this primary principle.

To illustrate the stochastic gradient, we consider applying stochastic gradient descent to a choice model with J alternatives, and a data set of N observations, each of them containing the vector of exploratory variables x_n and the observed choice i_n. The choice model

$$P_n(i|x_n;\theta) \tag{1}$$

provides the probability that individual n chooses alternative i in the context specified by x_n, where $\theta \in \mathbb{R}^K$ is a vector of K unknown parameters, to be estimated from data. Typically, this is done using maximum likelihood estimation, where the log likelihood function $\mathcal{L}(\theta)$ is maximized:

$$\max_{\theta\in\mathbb{R}^K} \mathcal{L}(\theta) = \max_{\theta\in\mathbb{R}^K} \sum_{n=1}^{N} \ln P_n(i|x_n;\theta). \tag{2}$$

In the optimization literature, it is custom to define the algorithms for minimization problems. We follow the same convention, and consider the equivalent minimization problem:

$$\min_{\theta\in\mathbb{R}^K} -\mathcal{L}(\theta) \tag{3}$$

The gradient of the log likelihood function is

$$\nabla\mathcal{L}(\theta) = \sum_{n=1}^{N} \nabla \ln P_n(i|x_n;\theta). \tag{4}$$

To obtain its stochastic version, we draw randomly, without replacement, a subset of N' observations from the data that we call a *batch*, and calculate:

$$\nabla_{N'}\mathcal{L}(\theta) = \sum_{n\in N'} \nabla \ln P_n(i|x_n;\theta). \tag{5}$$

As shown in the above analysis, the variants of the stochastic gradient methods that are successful in ML could be used as such to estimate the parameters of choice models. This could be used to decrease the time and computational cost of estimating choice models on large datasets. However, there are three key differences between the two contexts

which must be considered. Firstly, the parameters in choice models are an important output, and are used to estimate behavioural indicators such as Values of Time (VoT) and elasticities for the population. Conversely, parameters in ML models typically have no behavioural interpretation, and only the model predictions are treated as a modelling output. It is therefore typical to allow choice model parameter estimates to fully converge during estimation, so as to obtain the highest accuracy and precision of each individual parameter estimate. Conversely, in ML, it is typical to restrict the model from converging fully to prevent overfitting, for example by restricting the number of training epochs.

Secondly, choice models tend to have far fewer parameters than used in ML. Complex choice models have hundreds of parameters, while the neural networks used in deep learning can involve millions of unknown parameters. There has been recent efforts to reduce the number of parameters in ML models. For example, Wu (2019) investigates simplifying Convolutional Neural Networks (CNNs). Using matrix decompositions, the author is able to reduce the number of parameters from three million to just above three thousand, with only a small loss of precision. Nonetheless, these models are still complex, and exceed the typical number of parameters used in choice models.

Finally, choice data is typically collected using specific sampling strategies and designs of experiments. The objective is to obtain a representative sample whilst avoiding redundancies. Furthermore, the analyst usually has a model in mind when designing the data collection. In contrast, ML techniques are often applied to datasets collected automatically or that were originally collected for other purposes (e.g. trip records from contactless payment cards), in order to detect patterns which have not previously been considered. It is therefore common to have a great deal of redundancy in the data.

In this paper, we introduce a new algorithmic framework for optimisation of choice models which addresses the key differences between the choice modelling and ML contexts. Our framework includes three primary contributions:

- the use of a stochastic Hessian (that is, second derivative matrix), which is possible thanks to the relatively low number of unknown parameters in discrete choice models,

- the possible modification of the batch size from iteration to iteration, which is used to allow the high accuracy and precision in individual parameter estimates required in choice models,

- a change of optimization algorithm depending on the size of the batch, which aims at finding the best trade-off between accuracy and efficiency.

The rest of the paper is laid out as follows. In the next section, we describe optimization algorithms used both for the estimation of Discrete Choice Models (DCMs) and in ML. Then, in Section 3, we present in detail the three ideas mentioned above and propose a catalog of variants of optimization algorithms for the estimation of choice models. In Section 4, we evaluate the performance of a series of algorithms on various choice models with large data sets. Finally, in Section 5, we conclude this article and mention some further ideas that will be investigated in the future.

2 Literature Review

To the best of our knowledge, the maximum likelihood estimation of the parameters of choice models exclusively relies on deterministic algorithms that are variants of the line search and trust-region methods. In particular, the main estimation packages written in Python (Pandas Biogeme (Bierlaire, 2003, 2018), PyLogit (Brathwaite et al., 2017), and Larch (Newman et al., 2018)) all use the `minimize` function from the package Scipy (Jones et al., 2014). This makes use of the quasi-Newton Broyden-Fletcher--Goldfarb-Shanno (BFGS) algorithm. While this performs well for estimating small-to-medium-sized models, it struggles with larger, more complex models with more parameters. With the availability of larger and larger data sets, the performance of these methods is completely dominated by the time it takes to calculate the log-likelihood function, its gradient, and its possible second derivatives matrix. For example, Hillel (2019) shows that fitting a Feed-Forward Neural Network (FFNN) in the Tensorflow Python library is up to 200 times faster than estimating a Nested Logit model in Pandas Biogeme on the same data set containing 81'086 observations, despite the former having far more parameters.

To understand how we may be able to estimate choice models more quickly on large datasets, we can look for inspiration from ML. Datasets used in ML (and in particular in Computer Vision) can contain millions of observations. For example, ImageNet, a collection of images labeled by hand, contains around fifteen million images. The sheer size of the full dataset ($\approx$150GB) prevents it from being stored in memory. As such, analysing the full dataset is a significant computational challenge. As the full dataset cannot be stored in memory, the data must be batch processed. This explains the importance of the stochastic approach, where the gradient is calculated on only a batch of the data each time, for ML.

In order to better understand the existing approaches used to estimate both choice models and ML, we conduct a literature review of existing algorithms 2.1. Then, in Section 2.2, we identify and discuss the gaps found in the literature for the optimisation of choice models. We also link them to the three primary contributions presented in Section 1.

2.1 Overview of existing studies

In this section, we give a descriptive overview of the literature. We identify and review twenty-six studies which propose stochastic optimisation algorithms. We focus specifically on quasi-Newton and second-order approaches, whilst including key examples of first-order algorithms. The selected studies, whilst not exhaustive, are believed by the authors to represent a broad overview of the existing optimisation algorithms. For each of these algorithms, we analyze five aspects:

- the mathematical order, *i.e.* if it uses a gradient-based (first-order), a quasi-Newton-based (1.5-th order), or a Newton-based step (second-order),
- if it uses an adaptive batch size technique or if the batch size is constant,
- if the theoretical convergence of the algorithm is analyzed,
- if a quantitative assessment of the algorithm is conducted,

- a summary of the numerical applications of the algorithm.

The results of the review are displayed in tables 1 and 2. Table 1 shows the details of the first-order stochastic algorithms, whilst Table 2 shows the quasi-Newton (1.5th order) and second-order stochastic algorithms. Each table summarises the five considered features for each algorithm, alongside the reference and algorithm name. The following discussion evaluates the results in these tables.

Of the twenty-six algorithms considered in this review, fourteen are first-order, six are quasi-Newton (1.5th order), and six are second order. We refer the reader to the article of Ruder (2016) for a precise overview of the first-order methods. Whilst the review was focused on quasi-Newton and second-order algorithms, we find that the literature focuses predominantly on first-order approaches. We believe this focus is due to the speed of first order algorithms for optimising large, complex neural networks.

Among the six second-order algorithms, three of them use the Hessian-Free (truncated Newton) optimization technique. This technique consists of approximating the problem using the second Taylor expansion and then solving it using a conjugate-gradient. The remaining three second-order stochastic methods use a subsampling method of the Hessian to avoid its heavy computation at each step.

The majority of the algorithms use a fixed batch size, with only three out of the twenty-six algorithms using an adaptive batch size technique. All three articles which make use an adaptive batch size are recent (2016-2018). None of the second-order methods, and only one of the quasi-Newton algorithms, make use of an adaptive batch size.

In terms of the analysis within the study, fifteen out of the twenty-six articles conduct theoretical analysis on the convergence of their algorithms. No theoretical convergence rates are calculated for any of the Hessian-Free techniques, nor those using an adaptive batch size. The algorithm RMSProp (Tieleman and Hinton, 2012) does not provide any theoretical nor numerical results since it was only presented in a lecture. Conversely, twenty-two of the 26 references introducing new algorithms include a numerical application of the algorithm. The four references which do not include a numerical application are the oldest considered (1951-1992). This shows the importance of presenting numerical results.

Finally, we can see that the algorithms in the study have been applied in multiple domains, but none explicitly for choice modelling. The earliest algorithms (4/26) were first applied to classical optimization problems. The remainder of the algorithms are applied to ML problems, with the majority (13/26) being applied to neural networks. This shows the predominant focus on optimising neural networks in the literature.

2.2 Gaps in knowledge

As shown by the results of the literature review, the predominant focus of existing optimisation research has been the optimisation of neural networks, with none of the algorithms explicitly designed for the optimisation of choice models. As discussed, there are substantial differences between the optimisation of neural networks and DCMs. In this section, we therefore assess the limitations of the existing algorithms in terms of optimising DCMs. Furthermore, we identify three gaps in knowledge in the existing research which may enable higher performing optimisation algorithms for DCMs.

The following sections identify specific gaps in knowledge for stochastic optimisation of discrete choice models. The limitations are linked to three themes; stochastic second order approaches, adaptive batch size, and combined first order and second order approaches.

2.2.1 Stochastic second order approaches

ML researchers have predominantly focused on stochastic first-order algorithms due to their speed when estimating parameters in large models. Existing research by Lederrey et al. (2018a) shows that first-order stochastic methods are not able to achieve convergence on a Multinomial Logit Models with ten parameters. The authors compare a gradient descent algorithm, a mini-batch Stochastic Gradient Descent (SGD), Adagrad (Duchi et al., 2011), and SAGA (Defazio et al., 2014). They note that using a normalization on the parameters leads to more accurate results. However, these algorithms are still not able to converge to the optimal values. This failure is mainly due to the required precision for the convergence. Indeed, the parameter values are a key output of DCMs. These parameters are used to calculate behavioural indicators such as the value-of-time (VOT) and elasticities. Therefore, it is critical to achieve convergence with high precision.

Of the six second-order stochastic algorithms reviewed, none calculate the exact Hessian, with all using an approximation instead. However, the smaller number of parameters used in choice models compared to ML models means computing the use full Hessian for a batch of data is less computationally complex. There is therefore a need to investigate stochastic second order approaches which compute the exact Hessian. The computation of the full Hessian could be used for choice models to obtain parameter estimates with the necessary precision for convergence.

2.2.2 Adaptive batch size

None of the second-order methods found in the literature use an adaptive batch size. Furthermore, among the three algorithms proposing adaptive batch size methods, all of them couple the batch size with the *learning rate* (a parameter specified in ML estimation). While Goyal et al. (2018) shows that these two parameters are related, this coupling is specifically targeting ML models, where the learning rate is often set to a fixed value or is slightly decreasing at each iteration.

The learning rate in machine learning is similar to the *step size* used in classical optimization problems. Typically, in classical optimization, the step size is computed using more advanced techniques such as Line Search methods or Trust-Region methods. Since a high degree of precision is required for choice models, especially at the later stage of the optimization process, the step size should be separated from the batch size. Indeed, close to the optimum value, the optimization algorithm should both use a small step and as many data points as possible to achieve the highest precision.

While Line Search and Trust-Region methods have already been applied to ML, as demonstrated by Rafati et al. (2018), they have not yet been used in combination with an adaptive batch size. Also, Lederrey et al. (2018a,b) have demonstrated that the use of a full batch data is required at the end of the optimisation process to achieve the appropriate precision for DCMs. Therefore, it is required to develop an adaptive batch size technique with a second order approach, that does not interfere with the step size and that will

eventually reach the full dataset, as needed to achieve the required precision for choice models.

2.2.3 Combined first and second order approaches

All the algorithms presented in the review use the same algorithm throughout the optimization process. This can hinder the performance of the optimization since the requirements change during the process. Indeed, an optimization can be faster and less precise at the beginning of the optimization and then become more precise (and slower) close to the optimal solution. This can be partially achieved by using an adaptive batch size technique. However, this could be further addressed by switching the optimization algorithm at the right time to cope with the increased complexity due to larger batch size. There is therefore a need to investigate hybrid algorithms, which switch between first and second order approaches at the appropriate point in the optimization process.

Reference	Name	Order	ABS	Theoretical	Numerical	Application
Robbins and Monro (1951)	SGD	1st		✓		Regression functions
Polyak (1964)	Momentum	1st		✓		Classical optimization problems
Nesterov (1983)	NAG	1st		✓		Convex optimization problems
Polyak and Juditsky (1992)	Averaging	1st		✓		Classical optimization problems
Duchi et al. (2011)	Adagrad	1st		✓	✓	Image, text, and handwritten digit classification
Zeiler (2012)	Adadelta	1st			✓	Handwritten digit classification
Tieleman and Hinton (2012)	RMSProp	1st				None, first shown in a lecture
Schmidt et al. (2013)	SAG	1st		✓	✓	Binary classification
Defazio et al. (2014)	SAGA	1st		✓	✓	Handwritten digit, binary, and multivariate classification
Kingma and Ba (2014)	Adam & AdaMax	1st		✓	✓	Logistic Regression, Neural Networks, and Convolutional Neural Networks
Dozat (2016)	Nadam	1st			✓	Handwritten digit classification
Balles et al. (2016)	CABS	1st	✓		✓	Convolutional Neural Networks
Devarakonda et al. (2017)	AdaBatch	1st	✓		✓	Multiple Neural Networks architecture
Reddi et al. (2018)	AMSGrad & AdamNc	1st		✓	✓	Logistic Regression and Neural Networks

Table 1: Analysis of first-order stochastic optimization algorithms included in the review. The term "ABS" stands for Adaptive Batch Size, "Theoretical" for the analysis of theoretical convergence, and "Numerical" for the quantitative assessments.

Reference	Name	Order	ABS	Theoretical	Numerical	Application
Bordes et al. (2010)	SGDQN	1.5th		✓	✓	Dense and sparse large datasets (PASCAL Large Scale challenge)
Martens (2010)	HF	2nd			✓	Image classification via Deep Learning models
Kiros (2013)	Stochastic HF	2nd			✓	Classification and deep autoencoder tasks
Wang et al. (2014)	SQN & RSQN	1.5th		✓	✓	Convex and non-convex optimization problems
Mokhtari and Ribeiro (2014)	RES	1.5th		✓	✓	Well- and ill-conditioned problems with large scale datasets
You and Xu (2014)	SHF	2nd			✓	Speech recognition
Keskar and Berahas (2016)	adaQN	1.5th			✓	Recurrent Neural Networks
Agarwal et al. (2016)	LiSSA	2nd		✓	✓	Handwritten digit and multivariate classification
Mutny (2016)	ISSA	2nd		✓	✓	Least square estimators
Ye and Zhang (2017)	AccRegSN	2nd		✓	✓	Least square regressions
Gower et al. (2018)	hBFGS	1.5th			✓	Matrix Inversion problems
Bollapragada et al. (2018)	PBQN	1.5th	✓		✓	Logistic Regressions and Neural Networks

Table 2: Analysis of 1.5th- and second-order stochastic optimization algorithms included in the review. The term "ABS" stands for Adaptive Batch Size, "Theoretical" for the analysis of theoretical convergence, and "Numerical" for the quantitative assessments.

2.3 Summary

In previous section, we identify gaps in knowledge in the literature across three themes; stochastic second order approaches, adaptive batch size, and combined first and second order approaches. Researchers have already investigated solutions for some of these limitations individually. However, we could not find any research investigating their combination in a systematic approach. We thus aim at combining the three primary contributions stated in the introduction, Section 1, in a new algorithm for the optimization of choice models. Thus, this algorithm should incorporate the following concepts:

- the use of a stochastic second-order approach which calculates the true Hessian for each batch,
- the use of adaptive batch size for second order approaches which do not couple the batch size to the learning rate,
- the use of a hybrid approach which combines first and second order algorithms and switches between them at the appropriate time.

3 Methodology

This section introduces our novel algorithmic framework for estimating DCMs. Section 3.1 provides a reminder of the two standard optimization techniques: *line search* and *trust regions*. Section 3.2 shows how we construct the new hybrid stochastic algorithms with adaptive batch size.

3.1 Line Search and Trust-Region Methods

3.1.1 Line search methods

Line search optimisation methods combine a descent direction with a line search. A descent direction is obtained by preconditioning the gradient, that is

$$p_k = -D_k \nabla \mathcal{L}(\theta_k) \tag{6}$$

Where D_k is a positive definite matrix.

There are typically three ways to select D_k:

- **Steepest descent** methods assume that D_k is the identity matrix. In that case, the descent direction is the opposite of the gradient.
- **Newton's method** assumes that D_k is the inverse of the second derivative matrix (possibly perturbed to make it definite positive).
- Quasi-Newton methods assume that D_k is a secant approximation of the inverse of the second derivative matrix, updated at each iteration. Among the many secant methods, we consider the BFGS algorithm for which, according to Fletcher (1987), the approximation is given by:

$$B_{k+1} = B_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k s_k^T B_k^T}{s_k^T B_k, s_k} \tag{7}$$

with $s_k = \theta_{k+1} - \theta_k$ and $y_k = \nabla\mathcal{L}(\theta_{k+1}) - \nabla\mathcal{L}(\theta_k)$.

A slightly different version of BFGS consists in approximating the inverse of the Hessian. In that case, the name **BFGS**$^{-1}$ is used and the approximation is given by:

$$B_{k+1}^{-1} = B_k^{-1} + \frac{\left(s_k^T y_k + y_k^T B_k^{-1} y_k\right)\left(s_k s_k^T\right)}{\left(s_k^T y_k\right)^2} - \frac{B_k^{-1} y_k s_k^T + s_k y_k^T B_k^{-1}}{s_k^T y_k} \tag{8}$$

The step is calculated with an inexact line search method, based on the two Wolfe conditions (Wolfe, 1969, 1971). The first condition guarantees that the step gives sufficient decrease in the objective function, while the second one makes sure that unacceptably small steps are ruled out.

3.1.2 Trust-region methods

Trust-region methods define a region around the current search point, where a quadratic approximation of the function value is "trusted" to be correct and steps are chosen to optimize the function within this region. There exist multiple ways to compute the quadratic approximation of the function value. The most common choice is a quadratic function of the type:

$$m_k(\theta_k + z) = \mathcal{L}(\theta_k) + \nabla\mathcal{L}(\theta_k)^T z + \frac{1}{2} z^T B_k z \tag{9}$$

where B_k is either the Hessian $\nabla^2\mathcal{L}(\theta_k)$ or an approximation of it. If the Hessian is chosen, the name *Trust-Region* algorithm is used. If an approximation of the Hessian is used it is referred to as a *Quasi-Newton Trust-Region*.

A central element of trust-region methods is the size of the region. Typically, the size of the region is modified during the search, based on how well the quadratic model agrees with the actual objective function value. Following Conn et al. (2000), the region is modified conditional to the ratio ρ_k of the actual function reduction to the reduction predicted by the quadratic model:

$$\rho_k = \frac{\mathcal{L}(\theta_k) - \mathcal{L}(\theta_k + z_k)}{m_k(\theta_k) - m_k(\theta_k + z_k)}. \tag{10}$$

Given the ratio ρ_k, the decision to change the trust region is based on the following rules:

$$\Delta_k = \begin{cases} \gamma_1 \Delta_k & \text{if } \rho_k \geq \eta_2, \\ \Delta_k & \text{if } \rho_k \in [\eta_1, \eta_2), \\ \gamma_2 \Delta_k & \text{if } \rho_k < \eta_1, \end{cases} \tag{11}$$

where $\gamma_1, \gamma_2, \eta_1$ and η_2 are all *a priori* defined parameters.

Intuitively, these rules imply that when the quadratic model is a good predictor of the function value (ratio close to 1), the region will not be modified. In contrast, when the ratio is too small or too large the quadratic model will no longer be a good predictor, and so the region will be modified.

Stopping Criterion Both line search and trust-region methods require a stopping criterion to determine when to stop iterating. A relative gradient based stopping criterion ensures that the algorithm stops when the norm of the relative gradient is below some threshold, ε:

$$\|\nabla_{rel}\mathcal{L}(\theta)\| \leq \varepsilon. \tag{12}$$

A sufficiently small value is chosen for ε, typically in the range $[10^{-6}, 10^{-8}]$. A more detailed discussion of the stopping criterion can be found in Dennis and Schnabel (1996) (see Chapter 7.2 page 159).

3.2 Hybrid Stochastic Algorithms with Adaptive Batch Size

We present our new algorithmic framework for optimisation of choice models by discussing its three main contributions:

- the use of a stochastic hessian,
- the possible modification of the batch size from iteration to iteration,
- the change of optimization algorithm depending on the size of the batch.

3.2.1 Stochastic Hessian

Inspired by the technique of stochastic gradient, our algorithmic framework includes a stochastic Hessian, defined as:

$$\nabla^2_{N'}\mathcal{L}(\theta) = \sum_{n\in N'} \nabla^2 \ln P_n(i|x_n;\theta), \tag{13}$$

where N' is a subset of observations, drawn randomly, without replacement, from the full dataset.

A stochastic Hessian can be included in SGD algorithms to create a **Stochastic Newton Method** (SNM), as in Lederrey et al. (2018b), or a **Stochastic Trust-Region** (STR) algorithm.

3.2.2 Adaptive Batch Size

We propose to modify the size of the batch as the algorithm proceed with a Window Moving Average (WMA) technique. WMA is a simple smoother, consisting of averaging the values of a time series across a window of W consecutive observations, thereby generating a series of averages. To better capture change in the data, more importance is usually given to the most recent iterations. The WMA at the m-th iteration is given by:

$$\mathrm{WMA}_{m,W} = \frac{W\mathcal{L}(\theta_m) + (W-1)\mathcal{L}(\theta_{m-1}) + \cdots + 2\mathcal{L}(\theta_{m-W+2}) + \mathcal{L}(\theta_{m-W+1})}{W + (W-1) + \cdots + 2 + 1} \tag{14}$$

Note that for all first W iterations, the size of the window is reduced to the iteration number, *i.e.* $W = m$.

Intuitively, this technique will increase the batch size when the algorithm is not able anymore to improve the value of the objective function. This generally happen for two

reasons. Either a local optimum in the current neighborhood has been reached, or the stochastic nature of the gradient and Hessian precludes the algorithm from making further progress.

More precisely, the decision to increase the batch size is based on a successive lack of progress in the past iterations. The progress at the m-th iteration is defined as:

$$\mathcal{I}_m = \frac{\text{WMA}_{m-1,W} - \text{WMA}_{m,W}}{\text{WMA}_{m-1,W}}. \tag{15}$$

We consider that there is a lack of progress when I_m is less than a threshold $I_m < \Delta$. After C iterations with a lack of progress, the batch size is increased by a factor of τ.

We call the above algorithm Adaptive Moving Average Batch Size (AMABS) and its pseudocode is given in Algorithm 1. The AMABS technique can be used in any stochastic optimization algorithm. For example, using the principle of stochastic Hessian, an **AMABS Newton's method** or an **AMABS Trust-Region** method can be defined.

Algorithm 1 Adaptive Moving Average Batch Size (AMABS)

Inputs:

- Current iteration index: m,
- Function value at iteration m: $\mathcal{L}(\theta_m)$,
- Current batch size: N'_m,
- Size of the full dataset: N_{max},
- Size of the window: W (default: 10),
- Threshold for successfull iterations: Δ (default: 1%),
- Maximum number of unsuccessfull iterations with the same batch size: C (default: 2),
- Expansion factor for the batch size: τ (default: 2),

Output: New batch size: N'_{k+1}

function AMABS
  Compute $\text{WMA}_{m,W}$, as in Equation 14.
  if $m > 0$ **then** ▷ At least two iterations required
    Compute $\mathcal{I}_m$ as in Equation 15 using $\text{WMA}_{m,W}$ and $\text{WMA}_{m-1,W}$
    if $\mathcal{I}_M < \Delta$ **then** ▷ Count the consecutive steps under threshold.
      $c = c + 1$
    else
      $c = 0$
    if $c == C$ **then** ▷ Update the batch size
      $c = 0$
      $N'_{k+1} = \min(\tau \cdot N', N_{max})$
    else
      $N'_{k+1} = N'_k$
  return N'_{k+1}

3.2.3 Hybridization

The use of the AMABS technique naturally brings the opportunity of using different optimization algorithms based on the batch size. Indeed, at the beginning of the optimization process when small batch size are used, it makes sense to use an algorithm that leads to more precise steps. As the batch size increases, the computation of the precise step takes more and more time, and the algorithm can become too slow. It makes therefore sense to switch to a less precise but faster algorithm. Our hybrid algorithm determines the algorithm to use based on the percentage of data used in a batch. The pseudocode is showed in Algorithm 2.

Algorithm 2 Hybrid algorithm with AMABS method

Inputs:

- Log likelihood function: $f(\theta) \equiv \mathcal{L}(\theta)$,
- Gradient of the log likelihood: $g(\theta) \equiv \nabla\mathcal{L}(\theta)$,
- Hessian of the log likelihood: $H(\theta) \equiv \nabla^2\mathcal{L}(\theta)$,
- Initial parameter value: θ_0,
- Algorithm generating a candidate for the next iteration using only the gradient (first-order or quasi-Newton method): `generateCandidateFirstOrder`(θ, g),
- Algorithm generating a candidate for the next iteration using the gradient and the Hessian (second-order method): `generateCandidateSecondOrder`(θ, g, H),
- Parameters specific to the algorithm:
  - Initial batch size: N'_0 (default: 1000),
  - Size of the full dataset: N_{max},
  - Size of the window: W (default: 10),
  - Threshold for successfull iterations: Δ (default: 1%),
  - Maximum number of unsuccessfull iterations with the same batch size: C (default: 2),
  - Expansion factor for the batch size: τ (default: 2),
  - Threshold for hybridization: Δ_H (default: 30%),
  - Threshold for stopping criterion: ε (default: 10^{-6})

Output: Optimized parameters: θ^*

1: **function** ITERATION
2: **if** $N'_m/N_{max} > \Delta_H$ **then**
3: Compute $f_m = f(\theta_m)$ and $g_m = g(\theta_m)$.
4: Generate candidate $\theta_{m+1} =$ `generateCandidateFirstOrder`(θ_m, g_m)
5: **else**
6: Compute $f_m = f(\theta_m)$, $g_m = g(\theta_m)$, and $H_m = H(\theta_m)$.
7: Generate candidate $\theta_{m+1} =$ `generateCandidateSecondOrder`(θ_m, g_m)
8: $N'_{m+1} = AMABS(m, f_m, N'_m, N_{max}, W, \Delta, C, \tau)$
9: Stop the optimization if $\nabla_{rel}\mathcal{L}(\theta_{m+1}) < \varepsilon$.

The functions `generateCandidateFirstOrder` (line 4) and `generateCan-`

didateSecondOrder (line 7) return the parameter values for the next iteration. For example, Newton's Method or Trust-Region method can be used as the function generateCandidateSecondOrder. The usual initial parameter value θ_0 is an array of zeros. The last two parameters introduced in Algorithm 2, Δ_H and ε, are further discussed in Section 4.5.

4 Results

This section presents the results of our experiments. Before showing the numerical results, we start by explaining our experimental design, *i.e.*, the algorithms, models, and datasets used in our experiments, as well as the implementation details.

4.1 Experimental Design

As depicted in Table 3, the new algorithmic framework presented in Section 3 allows us to compare the performance of 15 different algorithms. These algorithms can be split into three categories:

- *standard non-stochastic algorithms* (first 6 algorithms) – deterministic algorithms which are commonly used in the DCM community and are therefore considered as benchmarks. For example, the current version of Biogeme uses the Python package Scipy (Jones et al., 2014) with the BFGS^{-1} implementation. We will start with a comparison of the performance of these standard algorithms before moving to the comparison with stochastic approaches.

- *stochastic algorithms* (next 6 algorithms) – algorithms based on the AMABS method presented in Section 3.2.2. These methods are used to show the improvement in terms of estimation time over their non-stochastic counterparts. The algorithms NM-ABS and TR-ABS also make use of the stochastic Hessian, as presented in Section 3.2.1. The added value of using stochastic Hessian will therefore also be discussed.

- *hybrid stochastic methods* (last 3 algorithms) – meta-algorithms which combine two separate optimization algorithms as presented in Section 3.2.3. Three types of hybridization are investigated and discussed: (i) Newton's method and BFGS, (ii) Trust-Region method and BFGS, and (iii) Newton's method and BFGS^{-1}.

We collect empirical evidence of the behavior of these 15 algorithms by observing their performance on 10 different choice models, presented in Table 4. As shown in the column *Data*, two different sources of data are used. The first 9 choice models are estimated on data obtained from the London Passenger Mode Choice (LPMC) dataset. This dataset, collected by Hillel et al. (2018), contains mode choice on an urban multimodal transport network, from April 2012 to March 2015. In order to study the impact of the size of the dataset on the performance of the different algorithms, three subdatasets have been created:

- a small dataset (S), that contains observations from year 2012 (27'478 observations),

Name	Order	AMABS	Description
GD	1st		Generic gradient descent algorithm.
BFGS	1.5th		BFGS algorithm using Eq. 7.
$\texttt{BFGS}^{-1}$	1.5th		$\texttt{BFGS}^{-1}$ algorithm using Eq. 8.
TR-BFGS	1.5th		Trust-Region method with BFGS (Eq. 7).
NM	2nd		Generic Newton's method.
TR	2nd		Generic Trust-Region method.
GD-ABS	1st	✓	Stochastic Gradient Descent with AMABS.
BFGS-ABS	1.5th	✓	BFGS algorithm (Eq. 7) with AMABS.
$\texttt{BFGS}^{-1}$-ABS	1.5th	✓	BFGS^{-1} algorithm (Eq. 8) with AMABS.
TR-BFGS-ABS	1.5th	✓	Trust-Region with BFGS (Eq. 7) and AMABS.
NM-ABS	2nd	✓	Newton with AMABS.
TR-ABS	2nd	✓	Trust-Region with AMABS.
H-NM-ABS	Hybrid	✓	Hybridization: Newton + BFGS (Eq. 7).
H-TR-ABS	Hybrid	✓	Hybridization: Trust-Region + BFGS (Eq. 7).
HAMABS	Hybrid	✓	Hybridization: Newton + BFGS^{-1} (Eq. 8).

Table 3: Overview of all algorithms used for the optimization of DCMs. A small description of the algorithms is provided as well as their order and if it is including the adaptive batch size method.

- a medium dataset (M), that contains observations from years 2012 to 2013 (54'766 observations),
- a large dataset (L), that contains all observations, from year 2012 to 2015 (81'766 observations).

In order to analyze the impact of the number of parameters on the estimation time, we compare three MultiNomial Logit (MNL) models from Hillel (2019):

- the LPMC_DC model that contains 13 parameters to be estimated,
- the LPMC_RR model that contains 54 parameters to be estimated,
- the LPMC_Full model that contains 100 parameters to be estimated.

Finally, a tenth choice model was estimated on the Mobility and Transport MicroCensus (MTMC) dataset, a statistical survey of the travel behavior of the Swiss population (Danalet and Mathys (2018)). We use the most recent version of the survey, collected in 2015. This model provides an interesting opportunity to study the efficiency of all algorithms presented in Table 3 for estimating a rather large choice model (almost 250 parameters), on a medium size dataset (56'915 observations).

4.2 Implementation details

All models are optimized on a single node in a supercomputer (18 Cores Skylake Processor@2.30 GHz, 192GB) for each algorithm. We include a stopping criterion on the maximum number of epochs (1,000 epochs). This is done to avoid extremely long computation

Names	#Free	# Fixed	Data	#Observations
LPMC_DC_S	13	1	LPMC	27'478
LPMC_DC_M	13	1	LPMC	54'766
LPMC_DC_L	13	1	LPMC	81'086
LPMC_RR_S	54	17	LPMC	27'478
LPMC_RR_M	54	17	LPMC	54'766
LPMC_RR_L	54	17	LPMC	81'086
LPMC_Full_S	100	198	LPMC	27'478
LPMC_Full_M	100	198	LPMC	54'766
LPMC_Full_L	100	198	LPMC	81'086
MTMC	247	146	MTMC	56'915

Table 4: Summary of the models used for the performance analysis. The number of free and fixed parameters is provided as well as the dataset and the number of observations used in the model. All the models are Multinomial Logit models.

time for algorithms that would struggle in achieving convergence for certain models. In addition, since some of these algorithms are stochastic, the speed of convergence may differ on the same optimisation task. Thus, each stochastic algorithm is used to optimize each model 20 times. We impose an upper limit on the execution time for the 20 estimation process: 12 hours for the models LPMC_DC, 24 hours for the models LPMC_RR, 36 hours for the models LPMC_Full, and 48 hours for the model MTMC. Finally, since we want to be able to compare the efficiency of our algorithms with state-of-the-art DCM software, we also optimize all the models in Table 4 with Biogeme and Scipy within the same rules. All the results are presented and discussed in Section 4.3. The code implementing all the algorithms in Table 3 can be found on Github at: https://github.com/glederrey/HAMABS.

4.3 Performance analysis

In this section, we analyze the performance of the fifteen algorithms reported in Table 3. For ease of comparison, we decided to use a graphical approach named *performance profiles* to benchmark these algorithms. As stated by Beiranvand et al. (2017), performance profiles are a great tool to analyze algorithms in terms of efficiency, robustness, and probability of successfully performing a required task. The concept of performance profile is presented in Section 4.3.1, while results are analyzed in Section 4.3.3.

4.3.1 Performance profiles

Performance profiles are introduced by Dolan and Moré (2002) and are now a recurring tool used to compare performance of optimization algorithms. They are used to compare the performance of a set of optimization algorithms, $\mathcal{A}$, on a set of optimization problems, $\mathcal{P}$. For each pair $(p, a) \in \mathcal{P} \times \mathcal{A}$, they define a performance measure $t_{p,a} > 0$ whose large value indicates poor performance. Classical measures of performance are the execution time or the number of epochs.

In addition, a convergence test $\mathcal{C}_{p,a}$ states if algorithm a was able to optimize problem p. For each optimization problem p and optimization algorithm a, the performance ratio is defined as

$$r_{p,a} = \begin{cases} \frac{t_{p,a}}{\min_{a \in \mathcal{A}} t_{p,a}} & \text{if } \mathcal{C}_{p,a} \text{ passed,} \\ \infty & \text{if } \mathcal{C}_{p,a} \text{ failed.} \end{cases} \tag{16}$$

This leads to $r_{p,a} = 1$ for the best algorithm and $r_{p,a} = \infty$ for all algorithms a unable to solve problem p. The performance profile of an algorithm a is finally defined as

$$\rho_a(\pi) = \frac{|p \in \mathcal{P} : r_{p,a} \leq \pi|}{|\mathcal{P}|} \tag{17}$$

where $|\mathcal{P}|$ is the cardinality of the set $\mathcal{P}$. $\rho_a(\pi)$ represents the proportion of problems for which the performance ration $r_{p,a}$ for algorithm $a \in \mathcal{A}$ is within a factor $\pi \in \mathbb{R}$ of the best possible performance ratio.

Generally, $\pi \in \mathbb{N}^+$ is used to avoid showing too many data points. Furthermore, any upper bound on π can be used. However, if $\mathcal{R} = \max_{p \in \mathcal{P}, a \in \mathcal{A}} r_{p,a}$, $\forall r_{p,a} < \infty$, the performance profiles will remain the same for any $\pi \geq \mathcal{R}$. Therefore, $\mathcal{R}$ is used as the upper bound on π.

It is interesting to note that $\rho_a(1)$ corresponds to the percentage of problems for which algorithm a has the best performance. Also, $\rho_a(\mathcal{R})$ represents the percentage of problems that algorithm a was able to solve under the condition of the convergence test $\mathcal{C}$. Therefore, algorithms with high values of $\rho_a(\pi)$ are of interest.

4.3.2 Interpretation

A performance profile has the values of π on the x-axis ranging from 1 to $\mathcal{R}$, the maximum of all ratios. The proportion $\rho_a(\pi)$ is situated on the y-axis. To interpret these profiles, there are three specific elements to analyze:

- the proportion for each algorithm a at the value $\pi = 1$, *i.e.* at the leftmost side of the graph. Indeed, the proportion $\rho_a(1)$ indicates the percentage of problems for which an algorithm a is the best, based on the performance test $t_{p,a}$.

- the proportion for each algorithm a at the value $\pi = \mathcal{R}$, *i.e.* at the rightmost side of the graph. This proportion indicates the percentage of problems that algorithm a was able to solve within the convergence criterion $\mathcal{C}$.

- how quickly the algorithm a reaches a proportion of 100%, *i.e.* the line reaches the top of the graph. The value of π for which algorithm a reaches 100% indicates its worse relative performance compared to all other algorithms.

This thus means that the ideal performance profile start at 100% and finish at 100%. It means that this algorithm a is the best for the performance measure $t_{p,a}$. However, these kind of results are rare. Therefore, it is important to compare the algorithms by looking at the three points cited above to determine which algorithm is better.

As an example, we analyze in detail the line corresponding to the algorithm `HAMABS` in Figure 1. The performance measure $t_{p,a}$ corresponds to the execution time on this figure. We now analyze the three elements cited above for the algorithm `HAMABS`:

- it reaches a proportion of 70% at $\pi = 1$. It therefore means that this algorithm is the fastest for 7 out of the 10 optimized models. Tables 8 and 9, provided in Appendix 6.1, report the average time and the standard deviation to optimize each model with each algorithm. As seen in these tables, the HAMABS is effectively the fastest algorithm on seven out of ten models.

- it is able to solve all problems. Indeed, it reaches a proportion of 100% for $\pi = \mathcal{R}$. We can also verify that it is effectively the case in Tables 8 and 9.

- it reaches a proportion of 100% at $\pi = 5$. This thus means that this algorithm has at worst a relative performance of 5 compared to the fastest algorithm on all the models.

Based on the analysis above and by comparing the HAMABS algorithms with the other algorithms, we can conclude that this algorithm is the fastest and the most robust in general. Indeed, it is the fastest on the majority of the models. Besides, the only models on which this algorithm is not the fastest are the LPMC_DC models, the smallest models in terms of parameters. Also, we see that this algorithm is the fastest to reach 100% proportion. This thus shows that it is the most robust algorithm across all models..

4.3.3 Results

In our case, the set of problems $\mathcal{P}$ contains the ten models in Table 4 and the set of optimization algorithms $\mathcal{A}$ include the fifteen algorithms in Table 3. The convergence test $\mathcal{C}$ tells us weather the algorithm was able to converge with the required precision ε, in less than 1000 epochs. We selected two performance measures: the execution time and the number of epochs.

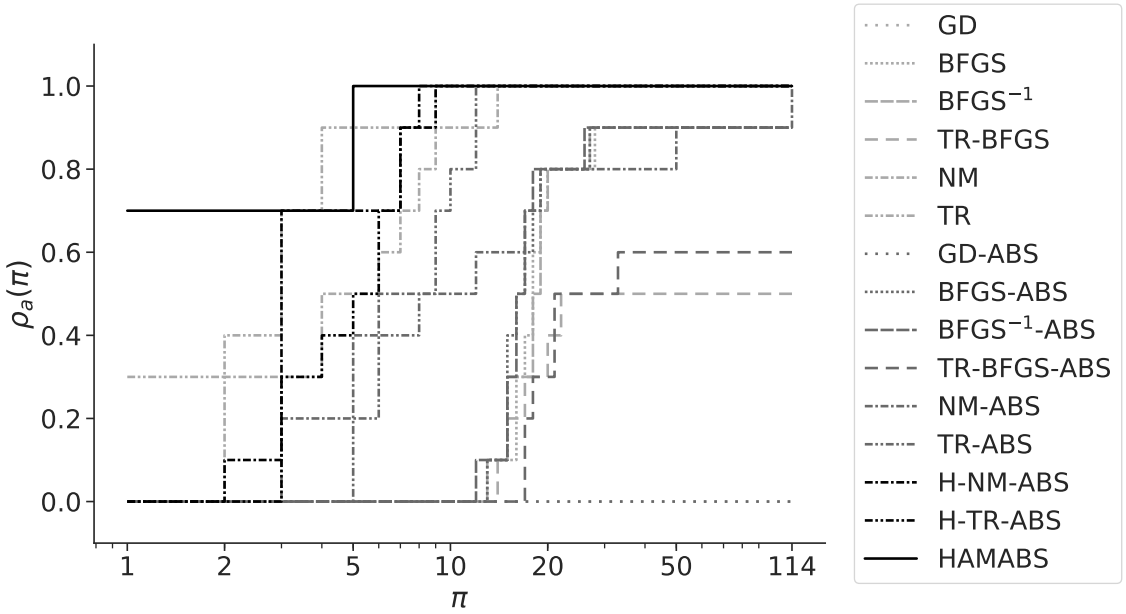


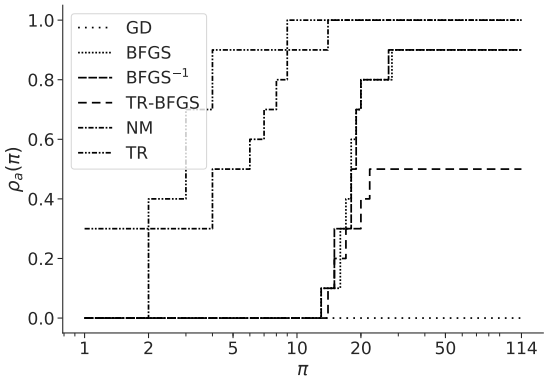
Figure 1: Performance profile on the execution time for the all models in Table 4 and all algorithms in Table 3.

Figure 1 shows the results for the execution time. The value of $\mathcal{R}$ for the execution is 114. While analyzing the lines in Figure 1, it is recommended to have a look at the reported execution times in Tables 8 and 9, provided in Appendix 6.1. We also provide the maximum ratio, $\mathcal{R} = 114$, for the execution time. In Section 4.3.2, we already analyzed the HAMABS algorithm in details. Since there are many algorithms to analyze, we discuss them further by types of algorithms.

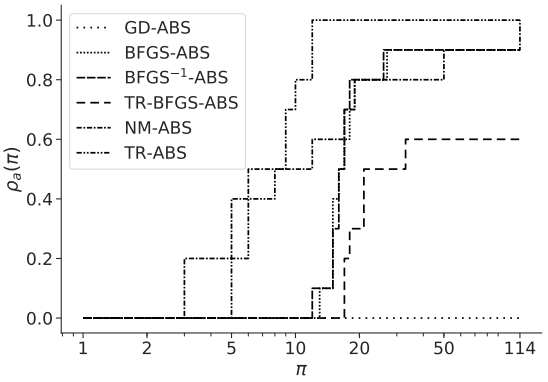
Standard non-stochastic algorithms Figure 2a shows the performance profile for all standard non-stochastic algorithms. We observe that these standard algorithms are struggling to optimize the models. Trust-Region and Newton's method are the fastest and the most robust methods amongst the standard ones and reach the 100% proportion with a relative performance of up to 15 times the fastest algorithm. The two BFGS methods are slower than the first two methods. Besides, they also fail to optimize some models. We also see that the algorithm TR−BFGS is struggling to optimize the models, indicating that the algorithm H−TR−ABS might struggle as well. The worst method is the Gradient Descent since it has never converged in less than 1000 epochs.

Stochastic algorithms Figure 2b shows the results for the algorithms using the AMABS technique. The behavior of these methods does not differ much as the slow standard methods stay slow with the AMABS method. For example, both the NM−AMABS and the TR−AMABS are the fastest and most robust algorithms. The algorithm GD−ABS is unable to optimize any model within the required number of epochs. Table 8 and 9 also show that, except for the TR−ABS, all AMABS methods are faster than the standard ones It thus shows that the AMABS algorithm is generally able to speed up the standard algorithms.

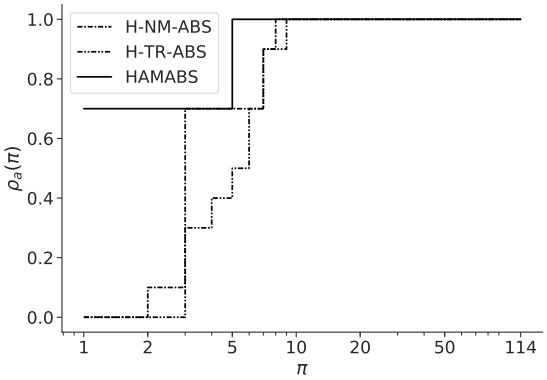
Hybrid stochastic algorithms Figure 2c shows the results for the three algorithms using the Hybridization and the AMABS method. These three methods are the fastest algorithms on larger models. While we already discussed the case of the HAMABS algorithm, the other two methods are never the fastest method for any models. However, they are slightly more robust than the other algorithms. Indeed, the H−NM−ABS is almost as good as the HAMABS algorithm. However, by looking at the times, we still see quite a difference between these two algorithms. Indeed, there is generally a 20% difference of execution time between these two algorithms. The H−TR−ABS seems to perform quite well. However, by looking at the times, we see that both the TR and the TR−ABS algorithms are faster. This is most likely due to the use of the Trust-Region method with the BFGS approximation being especially slow.



(a) Standard non-stochastic algorithms



(b) Stochastic AMABS algorithms



(c) Hybrid stochastic algorithms

Figure 2: Performance profiles on the execution time for all models in Table 4 splitted into different groups of algorithms.

As seen in Figures 1 and 2, the `HAMABS` algorithm is the fastest to optimize most of the models. While the execution time is important, the number of epochs used throughout the optimization process can also bring some important conclusions. An optimization algorithm performs one epoch as soon as it has seen all the data in the dataset once. Thus, the time to optimize a model is often correlated to the number of epochs it takes. Figure 3 shows the performance profile on the epochs for all algorithms and Figure 4 splits the previous figure based on the different groups of algorithms. We can see on Figure 3 that the `HAMABS` algorithm is more robust than most of the other algorithms, except for the `TR` algorithm.

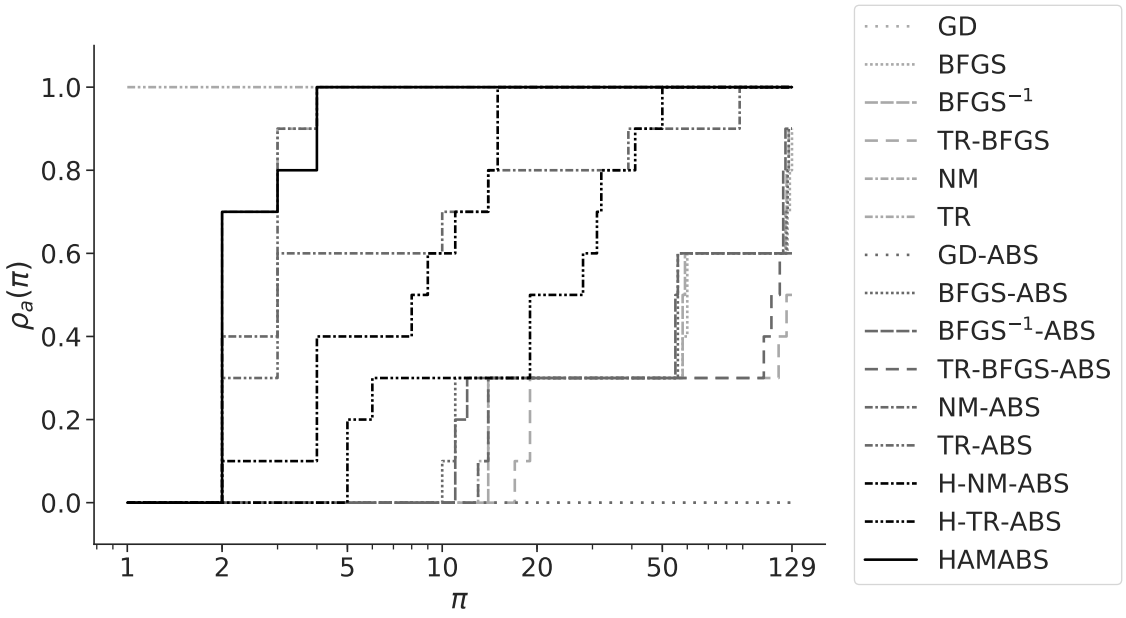
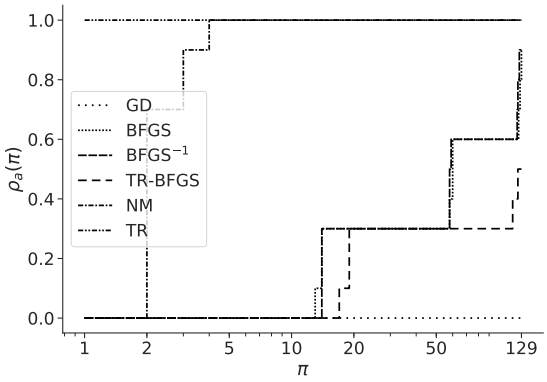
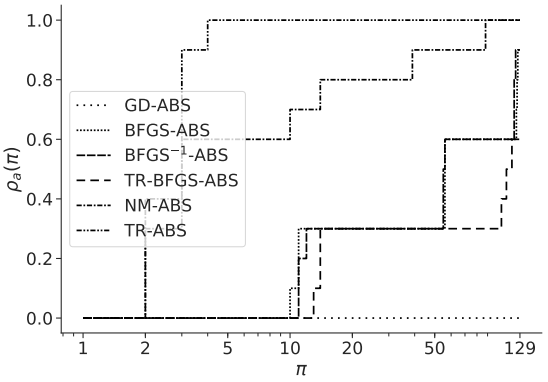


Figure 3: Performance profile on the epochs for the all models in Table 4 and all algorithms in Table 3.

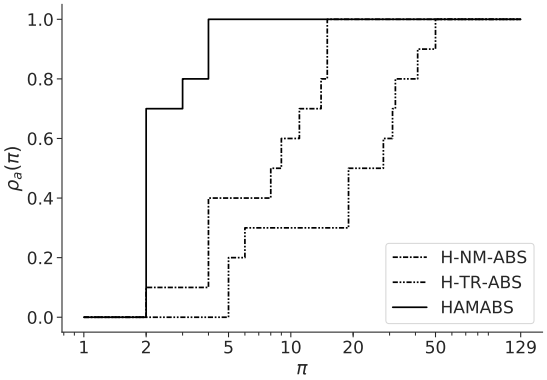
If we compare the different algorithms, we see that second-order methods tend to use fewer epochs to achieve convergence. Indeed, we can see in both Figure 4a and Figure 4b that the methods based on Newton's method (`NM`/`NM-ABS`) and Trust-Region method (`TR`/`TR-ABS`) are more robust than the other algorithms. This is expected because these methods use the information on the curvature. They thus require fewer steps to complete the estimation process. We also see that second-order methods using the full size dataset, `NM` and `TR`, are using less epochs than the stochastic methods, `NM-ABS` and `TR-ABS`. This is also an expected behaviour since the stochastic algorithms use less information per step. They thus need to perform many more steps, often leading to more epochs, to gain the same knowledge. On the other hand, they spend less time on each step, leading to a consequent speedup. It is interesting to note that the `HAMABS` algorithm is amongst the algorithms using the least number of epochs. Indeed, it reaches a proportion of 100% with a relative performance of 4. It therefore explains why this algorithm is that fast compared to the AMABS algorithms. Also, we see that the `H-NM-ABS` tends to use more epochs than the `HAMABS` algorithm. This could explain why the `HAMABS` is the fastest algorithm. Tables 10 and 11 in Appendix 6.2 show the average number of epochs with the standard deviation used by each algorithm to optimize each model.



(a) Standard non-stochastic algorithms



(b) Stochastic AMABS algorithms



(c) Hybrid stochastic algorithms

Figure 4: Performance profiles on the epochs for all models in Table 4 splitted into different groups of algorithms.

4.4 Comparison with State-of-the-Art Software

We now compare the performance of our best algorithm, the `HAMABS` algorithm, to Pandas Biogeme (Bierlaire, 2018), a state-of-the-art choice modeling software. Biogeme is using the Python package Scipy to optimize the models. The default algorithm for the minimization in this package is the BFGS^{-1}. Table 5 reports the average time to optimize each model for both Biogeme and the `HAMABS` algorithm. Besides, the last column shows the speedup gained by using the `HAMABS` algorithm instead of Scipy.

Models	Time [s]		Speedup
	HAMABS	Biogeme/Scipy	
LPMC_DC_S	1.86 ± 0.12	1.62 ± 0.01	$\times 1.15$
LPMC_DC_M	3.11 ± 0.20	2.79 ± 0.02	$\times 1.11$
LPMC_DC_L	4.59 ± 0.32	4.07 ± 0.04	$\times 1.13$
LPMC_RR_S	11.98 ± 1.23	65.17 ± 0.09	$\div 5.44$
LPMC_RR_M	18.46 ± 1.06	127.67 ± 0.30	$\div 6.91$
LPMC_RR_L	18.14 ± 1.06	177.09 ± 0.29	$\div 9.76$
LPMC_Full_S	257.02 ± 42.85	1462.51 ± 14.41	$\div 5.69$
LPMC_Full_M	405.43 ± 43.77	2480.06 ± 18.27	$\div 6.12$
LPMC_Full_L	486.31 ± 63.38	4758.28 ± 45.22	$\div 9.78$
MTMC	1243.95 ± 56.21	28008.10 ± 528.33	$\div 22.52$

Table 5: Comparison of the optimization time for all models in Table 4 between the HAMABS algorithm and Biogeme. The time are reported in seconds. The speedup corresponds to a ratio between the two compared values.

Results presented in Table 5 show that the algorithm HAMABS is generally faster than the Scipy package. On the models LPMC_DC, that have few parameters, the HAMABS algorithm is slower with a ration around 1.15. However, the HAMABS becomes faster than the Scipy package on the models LPMC_RR and LPMC_Full that include more parameters. This implies that a model that previously took minutes, or hours to converge, is now optimized in only a few seconds or minutes, respectively. The most important gain is on the optimization of the largest model, the MTMC model, with a speedup ratio exceeding 22. While Biogeme took around seven and a half hours to converge, our HAMABS algorithm is converging in less than 20 minutes.

Table 6 compares the two algorithms based on the number of epochs used for the optimization. Results show that the computational gains achieved by our HAMABS algorithm are even more important in terms of number of epochs. For Biogeme and the Scipy package, the number of epochs is directly correlated to the number of parameters. As a result, the number of epochs highly depends on the size of the model. For example, the MTMC models uses hundred times more epochs to be optimized than the LPMC_DC models. For our HAMABS algorithm, on the other hand, the number of epochs used is more stable across the different models. Indeed, between the smallest and the largest models, the average number of epochs only doubles. On the MTMC model, the speedup ratio in number of epochs between Biogeme and our HAMABS algorithm exceeds 600. The reason behind these discrepancies in the number of epochs is the stopping criterion. The Scipy package is using the standard, yet incorrect, gradient value as the stopping criterion. If the objective function and its derivatives are not correctly normalized, this criterion can either stop the algorithm too early or too late. Therefore, the use of an appropriate stopping criterion makes an important difference..

Models	Epochs		Speedup
	HAMABS	Biogeme/Scipy	
LPMC_DC_S	13.79 ± 1.70	122	$\div 8.85$
LPMC_DC_M	13.50 ± 2.05	114	$\div 8.44$
LPMC_DC_L	12.57 ± 1.93	123	$\div 9.78$
LPMC_RR_S	15.31 ± 2.53	787	$\div 51.40$
LPMC_RR_M	13.95 ± 1.67	809	$\div 57.97$
LPMC_RR_L	9.55 ± 0.56	772	$\div 80.84$
LPMC_Full_S	24.98 ± 2.16	1786	$\div 71.50$
LPMC_Full_M	20.42 ± 1.84	1531	$\div 74.96$
LPMC_Full_L	21.36 ± 2.05	1996	$\div 93.44$
MTMC	18.63 ± 1.58	11920	$\div 639.88$

Table 6: Comparison of the epochs used in the optimization process for all models in Table 4 between the HAMABS algorithm and Biogeme. The time are reported in seconds. The speedup corresponds to a ratio between the two compared values.

The similar ratios between the models LPMC_RR and LPMC_Full, can be due to the added complexity on the LPMC_Full models. Indeed, in these models, multiple parameters are computed on small populations. This leads to an increase in complexity, and the stochasticity might therefore not be that helpful. The algorithm has to perform more steps at the full size to find the parameter values on the small groups. We thus lose some time at the end of the optimization process compared to LPMC_RR models.

In definitive, our results showed that the HAMABS algorithm is not only the fastest among the 15 algorithms in Table 3, but that it is also much faster than the current implementation of the state-of-the-art choice modeling software Biogeme.

4.5 Sensitivity Analysis

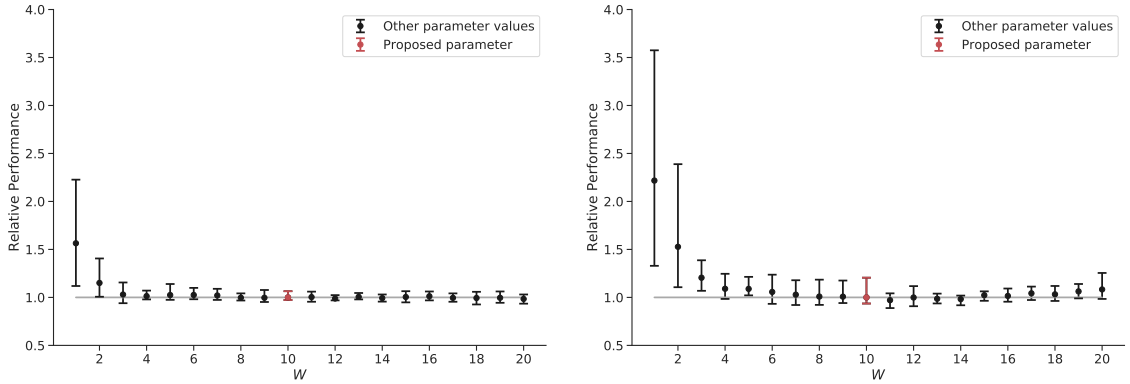
We want now to test the sensitivity of the HAMABS algorithm's parameters to make sure that the default parameters given in Algorithm 2 are sufficiently good across the different models. We selected the three following models to perform the sensitivity study: LPMC_DC_L, LPMC_RR_L, and LPMC_Full_L. The sensitivity analysis was performed on the estimation time of these models by the HAMABS algorithm. Each model was trained 20 times for all the values present in Table 7.

Parameter	Default	Test values
W	10	$[1, 2, 3, \cdots, 18, 19, 20]$
Δ	1%	[0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, 100]
C	2	$[1, 2, 3, \cdots, 13, 14, 15]$
τ	2	[1.1, 1.2, 1.3, 1.4, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 6, 7, 8, 9, 10]
Δ_H	30%	$[0, 5, 10, \cdots, 90, 95, 100]$
ε	10^{-6}	$[10^{-9}, 10^{-8}, \cdots, 10^{-1}, 10^{0}]$

Table 7: Parameter values of the HAMABS algorithm used for the sensitivity analysis. We refer the reader to Algorithm 2 for a detailed explanation of the parameters.

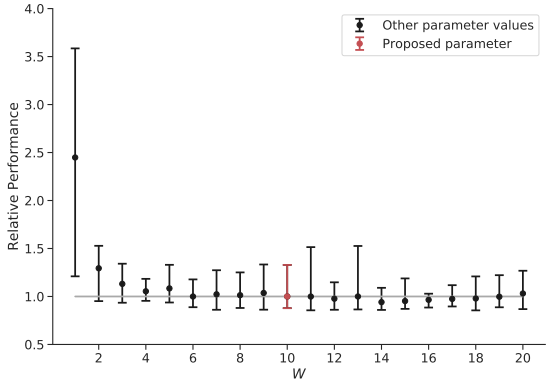
All results are reported using graphs for which the relative performance is indicated on the vertical axis. To ease the comparison of results across the models, all performance are normalized to the execution time obtained with the default parameter values (base value of 1).

Figure 5 shows the analysis for the parameter W; the size of the window. The results are similar across all models. Indeed, small values of W lead to an increase in the execution time. It can be explained by the fact that if the window is too small, the average is then noisy. Therefore, the computation of the improvement of the log likelihood is not precise. It thus increases the batch size too soon or too late. This thus leads to an increase in the total execution time. Large values for W do not affect the optimization process as much as small values. Indeed, we see a small increase in the execution time in Figure 5b when W is large. Indeed, if we use large values of W, the average is less influenced by the new data points. Thus, the AMABS reaction is slower. Therefore, a good value for W has to be in the middle. We thus propose to use $W = 10$.



(a) LPMC_DC_L

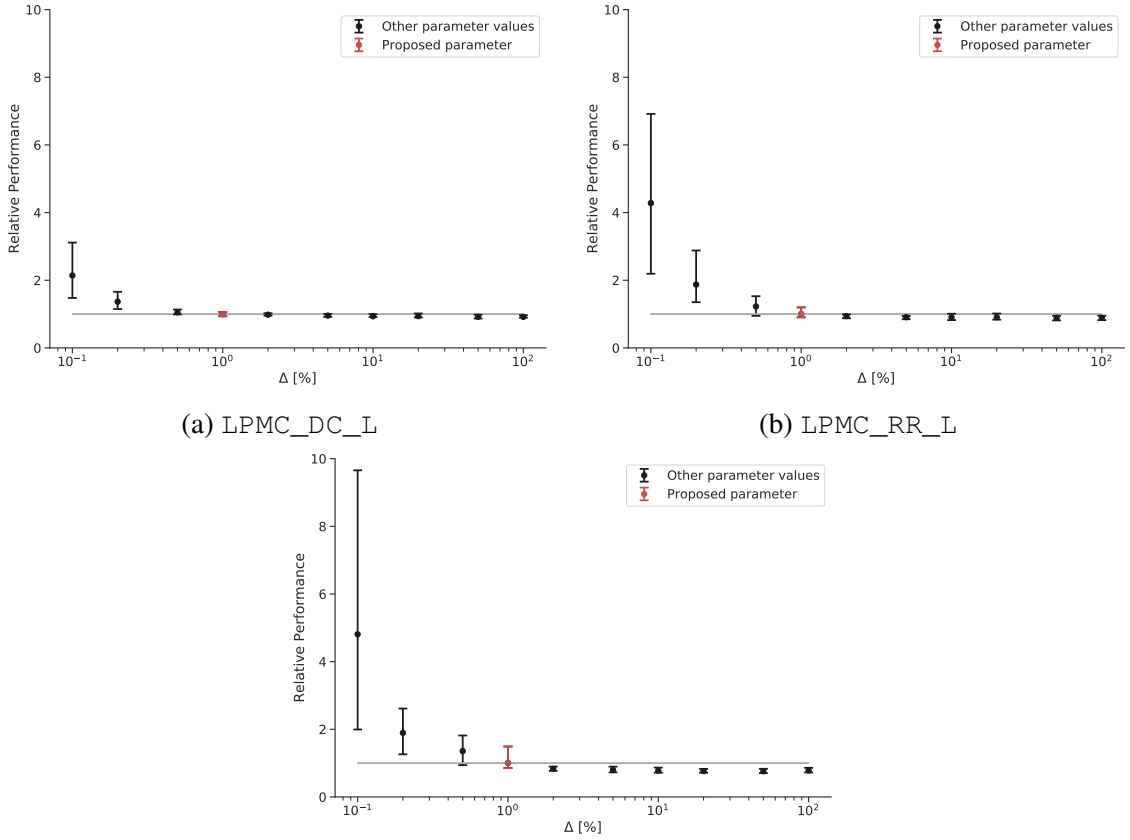
(b) LPMC_RR_L



(c) LPMC_Full_L

Figure 5: Sensitivity analysis for the parameter W for the three LPMC models using the large dataset. The black error bars correspond to the tested values and the red to the proposed value. The gray line correspond to the benchmark with the proposed parameters for the relative performance.

Figure 6 shows the analysis for the parameter Δ; the threshold on for successful iterations. Small values of this parameter lead to more substantial execution time. Indeed, the HAMABS algorithm is delaying the update of the batch size for too long. While larger values seem to perform slightly better, it also increases the probability of switching too soon for larger models. However, the value 1% corresponds to the moment where the curve is becoming a plateau. We thus propose to use this value for the parameter Δ.

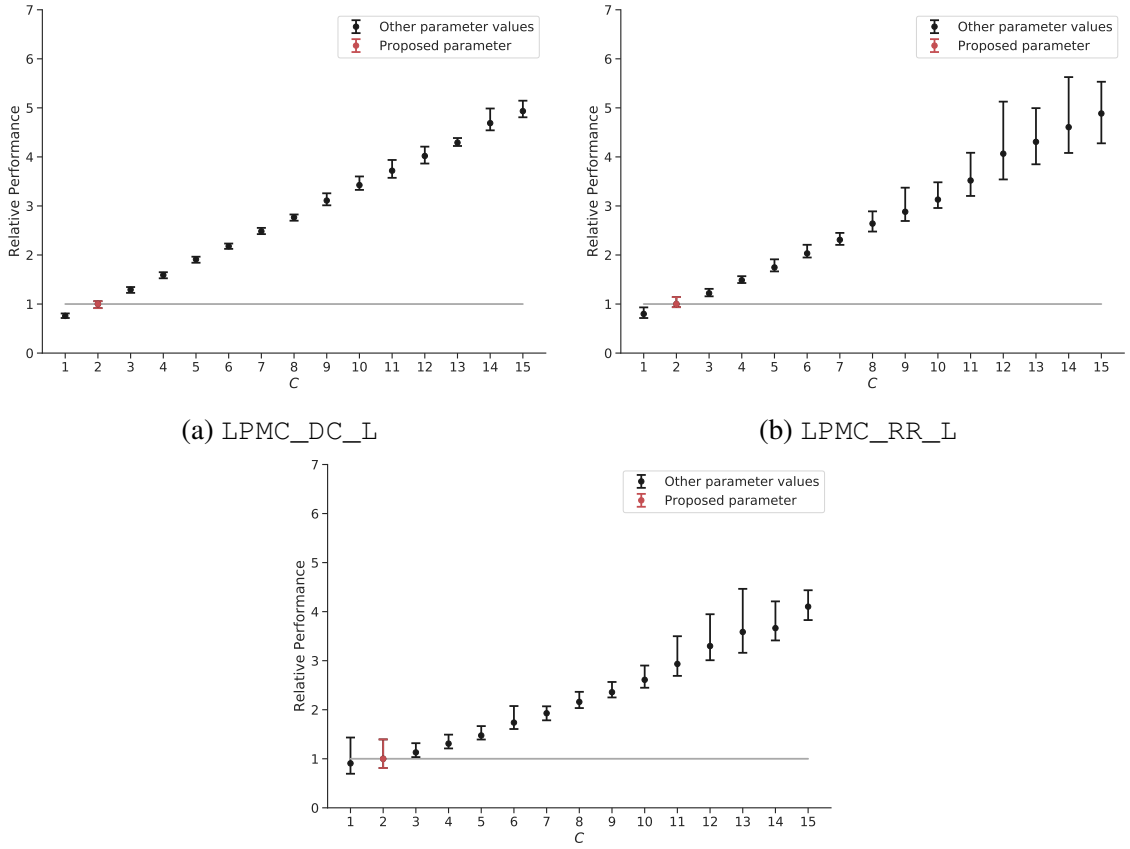


(a) `LPMC_DC_L` (b) `LPMC_RR_L`

(c) `LPMC_Full_L`

Figure 6: Sensitivity analysis for the parameter Δ for the three LPMC models using the large dataset. The black error bars correspond to the tested values and the red to the proposed value. The gray line correspond to the benchmark with the proposed parameters for the relative performance.

Figure 7 shows the analysis for the parameter C; the maximum number of unsuccessful iterations with the same batch size. It is interesting to note that for the three models, the relationship between the execution time and this parameter value is linear. Therefore, it is evident that using a value of 1 for C is faster. However, the advantage is reduced with the size of the model, as we can see when comparing Figure 7a and Figure 7c. Besides, if the count is set to 1, it could lead to triggering a false positive. Indeed, if the value of the window, W, is too small, an exceptionally lousy batch of data may lead to an increase of the batch size while the log likelihood can still be improved. It is thus preferable to slightly slow the execution time but increase the robustness of the algorithm. That is the reason we propose to use $\mathrm{C} = 2$.



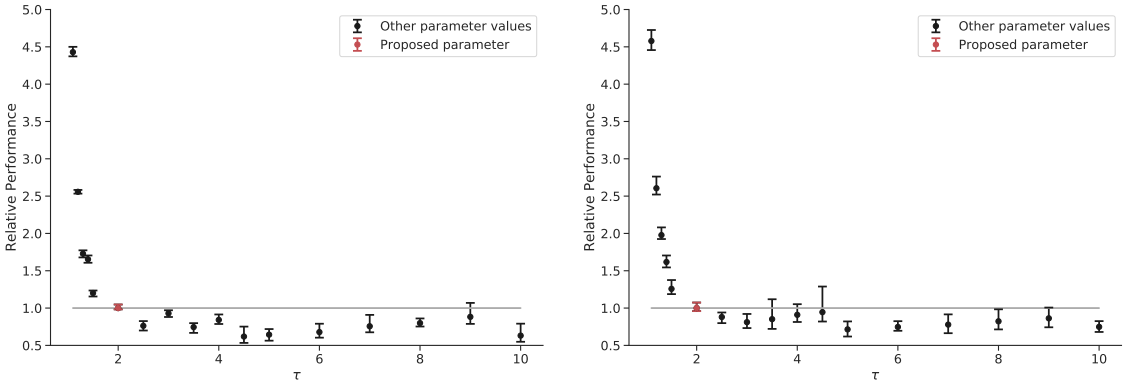
(a) LPMC_DC_L

(b) LPMC_RR_L

(c) LPMC_Full_L

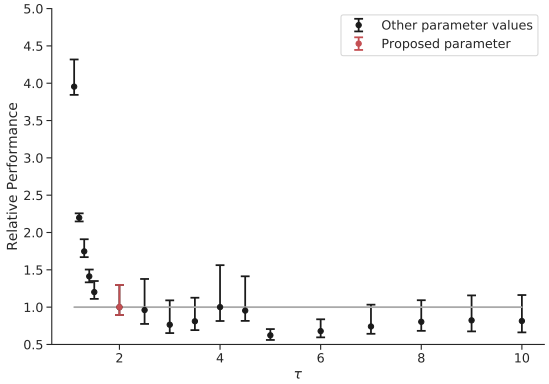
Figure 7: Sensitivity analysis for the parameter C for the three LPMC models using the large dataset. The black error bars correspond to the tested values and the red to the proposed value. The gray line correspond to the benchmark with the proposed parameters for the relative performance.

Figure 8 shows the analysis for the parameter τ; the expansion factor for the batch size. This parameter is particularly important since it decides how fast the algorithm uses the full dataset for its iterations. As expected, a small value for τ leads to longer execution time for all three models. However, a larger value for τ is more efficient for the smaller models. Indeed, for both the LPMC_DC_L and the LPMC_RR_L, using larger values lead to around 20% decrease in execution time. Since these models are quite small in the number of parameters, they already profit a lot from the starting batch size. It is then more favorable to switch the optimization algorithm and use more data. However, it seems that for the model LPMC_Full_L, switching too soon lead to more variance in the execution time. Therefore, it is better to stay more robust and use a smaller value for τ. We thus propose to use $\tau = 2$.



(a) LPMC_DC_L

(b) LPMC_RR_L

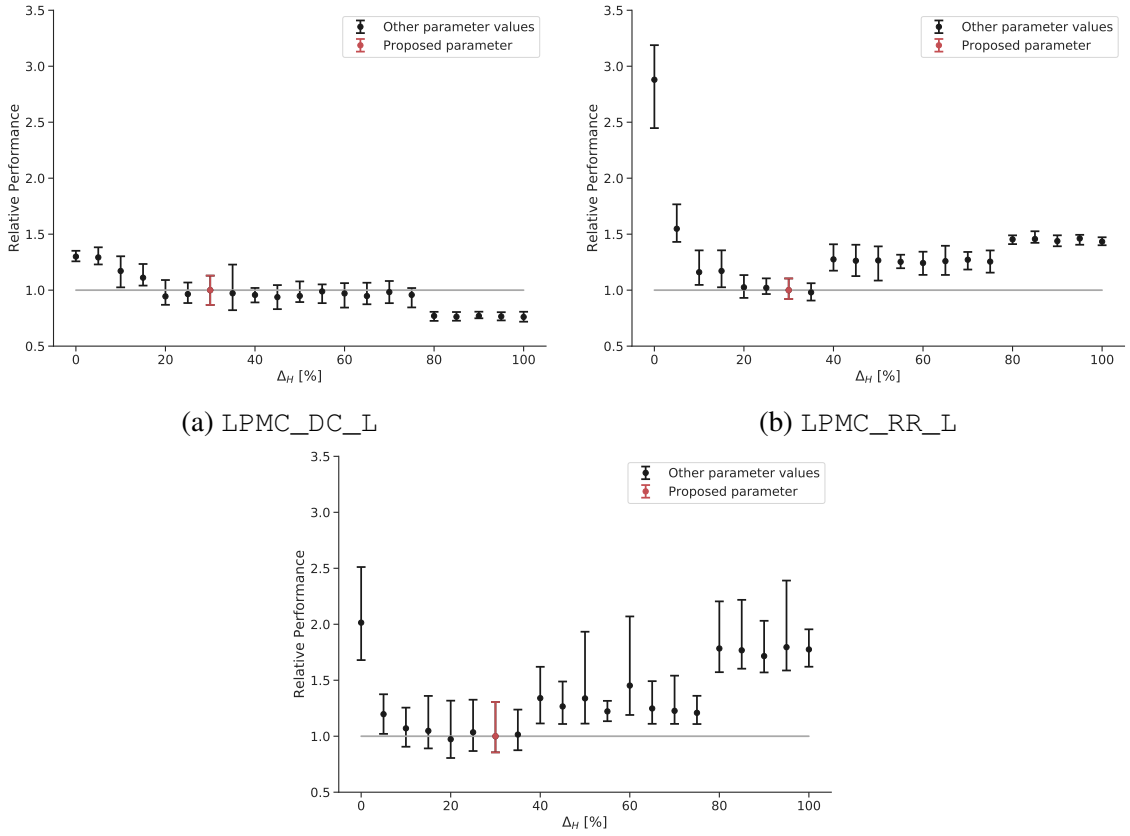


(c) LPMC_Full_L

Figure 8: Sensitivity analysis for the parameter τ for the three LPMC models using the large dataset. The black error bars correspond to the tested values and the red to the proposed value. The gray line correspond to the benchmark with the proposed parameters for the relative performance.

Figure 9 shows the analysis for the parameter Δ_H; the threshold for hybridization. Quite interestingly, this parameter does not influence much the execution time for the smallest model, the LPMC_DC_L. As discussed in Section 4.3, the HAMABS algorithm is not the fastest algorithm for the LPMC_DC models. Also, the execution is so small that the difference between Newton's method and BFGS is small. In addition, we see in Table 8, that Newton's method is faster than BFGS^{-1} on this particular model. Therefore, it is better to switch as late as possible. For the slightly larger model, LPMC_RR_L, high thresholds increase the execution time. As shown in Table 8, the BFGS methods are faster to optimize the models LPMC_RR. Therefore, it is expected that switching too late to BFGS is increasing the execution time. However, it seems that switching as soon as possible to BFGS is the most efficient for this model. This is most likely thanks to the help of the Hessian computation at the first step. Indeed, the BFGS algorithm generally starts with an Identity matrix. However, if we first perform a Newton step with the computation of the Hessian, it gives a good approximation as the starting point. Therefore, for the medium models, a smaller threshold for the hybridization is recommended. However, since the goal is to optimize large models as soon as possible, it is more important to use parameters specifically proposed for these models. As seen in Figure 9c, the best switch appears at around 30% of the data. It is slower to switch too soon since BFGS still take

more time than Newton's method to perform the early steps. And it is also slower to switch later since Newton's method takes too much time to compute the Hessian. Therefore, the proposed value is $\Delta_{\mathrm{H}} = 30$.

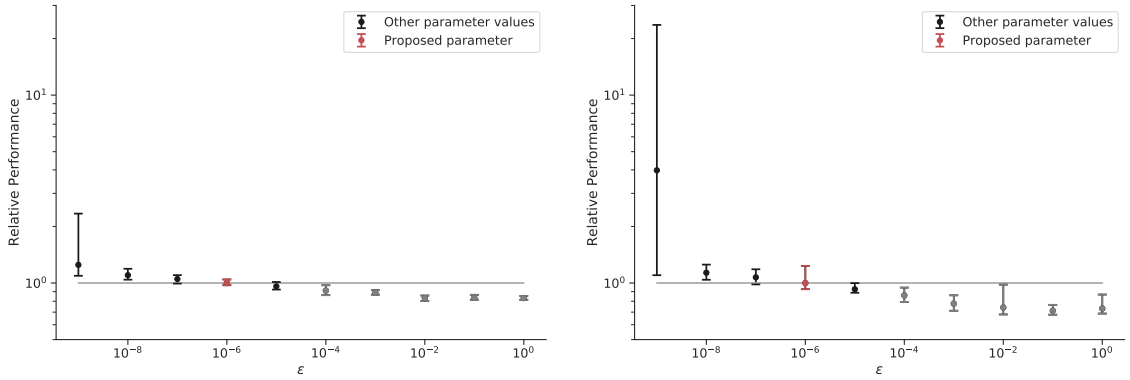


(a) `LPMC_DC_L` (b) `LPMC_RR_L`

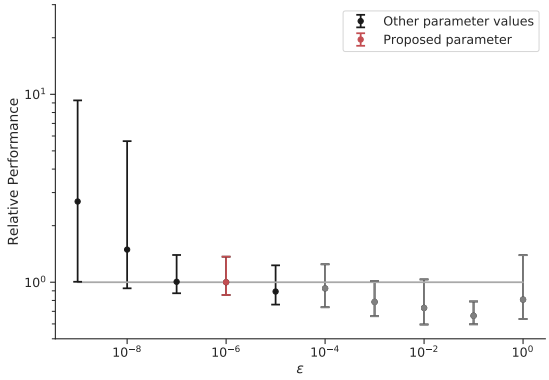
(c) `LPMC_Full_L`

Figure 9: Sensitivity analysis for the parameter Δ_{H} for the three LPMC models using the large dataset. The black error bars correspond to the tested values and the red to the proposed value. The gray line correspond to the benchmark with the proposed parameters for the relative performance.

Finally, Figure 10 shows the analysis for the parameter ε; the threshold for the stopping criterion. As shown in the three figures of Figure 10, using a stopping criterion that is too high is faster. However, it also leads to incorrect results. Indeed, the algorithm is stopping too soon, and the optimization has converged to the optimal point yet. We also see that using a stopping criterion too small leads to a significant increase in the execution time. In this case, it is unreasonable to try to be that precise. Therefore, a good value is a compromise between precision and speed. As shown in all these figures, a value for the stopping criterion between 10^{-8} and 10^{-5} is acceptable. We propose to use 10^{-6} even if other values can be used.



(a) `LPMC_DC_L` (b) `LPMC_RR_L`



(c) `LPMC_Full_L`

Figure 10: Sensitivity analysis for the parameter ε for the three LPMC models using the large dataset. The black error bars correspond to the tested values, the red to the proposed value, and the gray to optimization that were stopped too early. The gray line correspond to the benchmark with the proposed parameters for the relative performance.

As seen with the sensitivity analysis, the parameters might depend on the model's size. However, the goal of this article is to be able to speed up the optimization process of large choice models. Therefore, we selected parameters that lead to an improvement on these large models. Besides, half the execution time of a small model would only result in a gain of a few seconds. On the other hand, the same speedup would lead to minutes or even hours on larger models. It is, therefore, more rewarding to speed up the larger models.

5 Future work and conclusion

In this article, we present three primary contributions for the estimation of DCMs: the use of stochastic hessian, an adaptive batch size method, and the hybridization between optimization algorithms. We test 15 different algorithms ranging from standard algorithms to stochastic hybrid adaptive batch size algorithms. We prove that the use of an adaptive batch size technique is beneficial for the optimization time. Besides, since the AMABS method can be used with different optimization algorithms, we created three hybrid AMABS algorithms. We have shown that the best of these methods is the `HAMABS` algorithm.

It speed up the optimization by a factor of 23 on the MTMC model containing 247 parameters. Therefore, the use of faster algorithms opens the research to new possibilities for the future of choice modeling. As a concrete example, faster optimization time allows researchers to test many more specifications in the same amount of time. This can thus be used to develop automatic utility specification techniques to speed up the modelisation of DCMs.

For the future, we would like to work on two different improvements. The first one concerns the hybridization. The current way of doing it depends on the starting batch size. Indeed, due to the geometrical rule to increase the batch size in AMABS, it would be possible to miss the 30% mark to switch the optimization algorithm. Therefore, we would like to work on a better switch for the hybridization. One possible direction is to compare the improvement made by each algorithm in one step over the time it takes to do it. We would then have a metric in the percentage of improvement over seconds, and we could easily decide to switch the algorithm when BFGS leads to more improvements per second. The second improvement can be made on the rule for the update of the batch size in AMABS. Indeed, the geometrical rule seems to work well. However, it is possible that combining multiple rules could lead to faster optimization time. Finally, we would like to implement the final version of this algorithm in Pandas Biogeme.

6 Appendix

6.1 Execution time - Tables

Tables 8 and 9 show the average estimation time with the standard deviation for each models optimized by each algorithm. The bold values with the gray background represent the fastest algorithms for each model. The gray values mean that the algorithms were not able to converge in the required number of epochs (1000).

6.2 Number of epochs - Tables

Tables 10 and 11 show the average number of epochs with the standard deviation used by each algorithm to optimize each model. The bold values with the gray background represent the algorithms that have used the less epochs for each model. The gray values mean that the algorithms were not able to converge in the required number of epochs (1000).

Algorithms	Models					
	LPMC_DC_S	LPMC_DC_M	LPMC_DC_L	LPMC_RR_S	LPMC_RR_M	LPMC_RR_L
GD	52.16 ± 0.24	95.07 ± 0.45	140.95 ± 0.49	303.21 ± 0.25	567.61 ± 0.45	820.38 ± 1.46
BFGS	6.78 ± 0.03	13.11 ± 0.06	17.96 ± 0.06	183.83 ± 0.50	337.25 ± 0.30	492.90 ± 0.39
$BFGS^{-1}$	7.20 ± 0.02	12.96 ± 0.10	19.61 ± 0.12	177.66 ± 0.25	332.93 ± 0.40	473.32 ± 0.98
TR-BFGS	5.40 ± 0.04	11.05 ± 0.11	17.17 ± 0.07	227.83 ± 0.51	405.82 ± 0.60	627.43 ± 0.67
NM	0.65 ± 0.01	1.23 ± 0.01	1.95 ± 0.02	36.05 ± 0.98	70.28 ± 1.50	126.85 ± 2.18
TR	**0.40 ± 0.01**	**0.74 ± 0.01**	**1.03 ± 0.01**	23.63 ± 0.49	46.16 ± 0.66	67.13 ± 1.02
GD-ABS	50.53 ± 0.32	92.80 ± 0.51	138.20 ± 0.52	310.59 ± 1.21	568.12 ± 1.74	824.15 ± 2.54
BFGS-ABS	6.46 ± 0.29	10.80 ± 0.37	15.27 ± 0.63	169.83 ± 1.58	314.56 ± 3.83	474.42 ± 5.53
$BFGS^{-1}$-ABS	6.85 ± 0.10	11.13 ± 0.14	15.62 ± 0.16	169.36 ± 0.62	312.78 ± 2.79	460.49 ± 1.92
TR-BFGS-ABS	6.90 ± 0.37	12.49 ± 0.76	21.56 ± 1.04	194.49 ± 8.57	387.56 ± 12.13	590.62 ± 18.80
NM-ABS	7.64 ± 18.97	36.79 ± 52.75	116.95 ± 74.20	29.72 ± 2.41	53.28 ± 3.10	76.44 ± 5.78
TR-ABS	3.20 ± 0.06	6.43 ± 0.15	11.50 ± 0.13	50.64 ± 2.10	97.66 ± 4.82	175.75 ± 5.21
H-NM-ABS	2.83 ± 0.15	4.77 ± 0.25	6.97 ± 0.39	29.80 ± 1.38	48.46 ± 2.50	20.78 ± 1.96
H-TR-ABS	2.16 ± 0.14	3.82 ± 0.20	6.88 ± 0.38	34.97 ± 6.11	60.87 ± 13.06	157.69 ± 10.85
HAMABS	1.86 ± 0.12	3.11 ± 0.20	4.59 ± 0.32	**11.98 ± 1.23**	**18.46 ± 1.06**	**18.14 ± 1.06**

Table 8: Time in seconds used for the estimation of the models LPMC_DC and LPMC_RR by all the algorithms presented in Table 3. The values in light gray mean that the algorithms was not able to converge in the required number of eqpochs. The values in bold, in a gray cell, correspond to the the fastest optimization time.

Algorithms	Models			
	LPMC_Full_S	LPMC_Full_M	LPMC_Full_L	MTMC
GD	2785.66 ± 28.12	5344.42 ± 33.60	8031.02 ± 49.62	7842.92 ± 53.75
BFGS	3173.86 ± 22.37	6289.47 ± 34.27	9333.66 ± 63.08	10308.49 ± 61.36
$BFGS^{-1}$	3129.08 ± 10.83	5929.40 ± 27.85	8812.09 ± 66.60	10090.74 ± 56.43
TR-BFGS	2043.68 ± 16.10	4014.60 ± 24.31	5861.06 ± 39.09	5568.99 ± 38.86
NM	1398.81 ± 52.16	3230.83 ± 69.75	3984.04 ± 91.00	17199.76 ± 190.18
TR	540.07 ± 24.73	1021.88 ± 38.01	1501.90 ± 53.11	10613.62 ± 152.59
GD-ABS	2782.30 ± 23.46	5508.52 ± 27.21	7989.58 ± 64.22	7984.13 ± 65.41
BFGS-ABS	3163.19 ± 57.56	6119.99 ± 109.08	8944.95 ± 130.30	10225.68 ± 59.41
$BFGS^{-1}$-ABS	3021.65 ± 28.72	5814.22 ± 47.75	8721.32 ± 92.05	10083.01 ± 58.33
TR-BFGS-ABS	2035.42 ± 17.94	4001.22 ± 29.88	5796.55 ± 47.00	5586.36 ± 39.60
NM-ABS	4359.60 ± 14019.82	1982.06 ± 229.38	2721.13 ± 270.25	14535.94 ± 501.67
TR-ABS	1203.69 ± 84.31	2240.94 ± 99.74	4089.42 ± 134.02	13695.15 ± 757.32
H-NM-ABS	522.65 ± 47.01	939.05 ± 85.38	1376.32 ± 93.92	2749.46 ± 103.71
H-TR-ABS	591.33 ± 80.36	1085.78 ± 137.27	2209.45 ± 160.07	7633.65 ± 386.77
HAMABS	**257.02 ± 42.85**	**405.43 ± 43.77**	**486.31 ± 63.38**	**1243.95 ± 56.21**

Table 9: Time in seconds used for the estimation of the models LPMC_Full and MTMC by all the algorithms presented in Table 3. The values in light gray mean that the algorithms was not able to converge in the required number of eqpochs. The values in bold, in a gray cell, correspond to the the fastest optimization time.

Algorithms	Models					
	LPMC_DC_S	LPMC_DC_M	LPMC_DC_L	LPMC_RR_S	LPMC_RR_M	LPMC_RR_L
GD	1000	1000	1000	1000	1000	1000
BFGS	108	109	99	480	461	478
BFGS^{-1}	111	112	111	468	464	462
TR-BFGS	132	148	151	989	935	1000
NM	9	10	11	12	12	15
TR	**8**	**8**	**8**	**8**	**8**	**8**
GD-ABS	1000.71 $\pm$ 0.09	1000.53 $\pm$ 0.04	1000.28 $\pm$ 0.02	1000.62 $\pm$ 0.26	1000.57 $\pm$ 0.16	1000.29 $\pm$ 0.11
BFGS-ABS	85.05 $\pm$ 5.75	81.91 $\pm$ 3.91	78.88 $\pm$ 4.63	445.97 $\pm$ 4.61	442.97 $\pm$ 6.35	441.12 $\pm$ 5.45
BFGS^{-1}-ABS	90.02 $\pm$ 0.97	87.79 $\pm$ 1.55	85.64 $\pm$ 1.21	445.21 $\pm$ 1.64	439.54 $\pm$ 3.81	437.46 $\pm$ 1.80
TR-BFGS-ABS	101.91 $\pm$ 12.76	104.98 $\pm$ 11.82	109.02 $\pm$ 11.43	838.64 $\pm$ 38.49	880.85 $\pm$ 28.64	937.42 $\pm$ 31.23
NM-ABS	106.97 $\pm$ 297.92	304.96 $\pm$ 455.35	702.37 $\pm$ 455.05	10.13 $\pm$ 0.78	9.28 $\pm$ 0.60	9.11 $\pm$ 0.75
TR-ABS	15.62 $\pm$ 0.34	15.91 $\pm$ 0.33	20.41 $\pm$ 0.18	16.77 $\pm$ 0.35	16.71 $\pm$ 0.59	20.54 $\pm$ 0.42
H-NM-ABS	30.68 $\pm$ 2.86	30.26 $\pm$ 2.39	28.07 $\pm$ 2.40	67.84 $\pm$ 4.32	60.45 $\pm$ 4.39	11.96 $\pm$ 1.15
H-TR-ABS	37.53 $\pm$ 3.55	37.54 $\pm$ 2.78	47.26 $\pm$ 3.90	147.98 $\pm$ 30.61	144.95 $\pm$ 36.72	218.49 $\pm$ 21.42
HAMABS	13.79 $\pm$ 1.70	13.50 $\pm$ 2.05	12.57 $\pm$ 1.93	15.31 $\pm$ 2.53	13.95 $\pm$ 1.67	9.55 $\pm$ 0.56

Table 10: Number of epochs used for the estimation of the models LPMC_DC and LPMC_RR by all the algorithms presented in Table 3. The values in light gray mean that the algorithms was not able to converge in the required number of eqpochs. The values in bold, in a gray cell, correspond to the the fastest optimization time.

Algorithms	Models			
	LPMC_Full_S	LPMC_Full_M	LPMC_Full_L	MTMC
GD	1000	1000	1000	1000
BFGS	872	901	885	1000
$BFGS^{-1}$	878	859	868	1000
TR-BFGS	1000	1000	1000	1000
NM	20	24	20	23
TR	**7**	**7**	**7**	**14**
GD-ABS	1000.71 ± 0.15	1000.54 ± 0.05	1000.29 ± 0.03	1000.30 ± 0.03
BFGS-ABS	877.97 ± 16.90	870.00 ± 13.58	867.96 ± 11.58	1000.32 ± 0.05
$BFGS^{-1}$-ABS	856.91 ± 4.67	841.77 ± 6.28	843.83 ± 3.78	1000.37 ± 0.06
TR-BFGS-ABS	1000.67 ± 0.07	1000.28 ± 0.04	1000.79 ± 0.37	1000.74 ± 0.30
NM-ABS	65.91 ± 214.46	15.21 ± 1.62	14.11 ± 1.20	20.12 ± 0.60
TR-ABS	17.41 ± 1.06	17.10 ± 0.50	21.34 ± 0.39	18.40 ± 1.03
H-NM-ABS	98.20 ± 4.60	94.75 ± 5.36	99.43 ± 4.52	145.89 ± 8.99
H-TR-ABS	219.83 ± 37.59	216.80 ± 34.95	344.72 ± 27.79	564.62 ± 81.41
HAMABS	24.98 ± 2.16	20.42 ± 1.84	21.36 ± 2.05	18.63 ± 1.58

Table 11: Number of epochs used for the estimation of the models LPMC_Full and MTMC by all the algorithms presented in Table 3. The values in light gray mean that the algorithms was not able to converge in the required number of eqpochs. The values in bold, in a gray cell, correspond to the the fastest optimization time.

References

Agarwal, N., Bullins, B., and Hazan, E. (2016). Second-Order Stochastic Optimization for Machine Learning in Linear Time. *arXiv:1602.03943 [cs, stat]*. arXiv: 1602.03943.

Balles, L., Romero, J., and Hennig, P. (2016). Coupling Adaptive Batch Sizes with Learning Rates. *arXiv:1612.05086 [cs, stat]*. arXiv: 1612.05086.

Beiranvand, V., Hare, W., and Lucet, Y. (2017). Best practices for comparing optimization algorithms. *Optimization and Engineering*, 18(4):815–848.

Bierlaire, M. (2003). BIOGEME: a free package for the estimation of discrete choice models. *Swiss Transport Research Conference 2003*.

Bierlaire, M. (2018). PandasBiogeme: a short introduction. Technical report, Technical Report TRANSP-OR 181219, Transport and Mobility Laboratory, Ecole ....

Bollapragada, R., Mudigere, D., Nocedal, J., Shi, H.-J. M., and Tang, P. T. P. (2018). A Progressive Batching L-BFGS Method for Machine Learning. *arXiv:1802.05374 [cs, math, stat]*. arXiv: 1802.05374.

Bordes, A., Bottou, L., Gallinari, P., Chang, J., and Smith, S. A. (2010). Erratum: SGDQN is Less Careful than Expected. *Journal of Machine Learning Research*, 11(Aug):2229–2240.

Brathwaite, T., Vij, A., and Walker, J. L. (2017). Machine Learning Meets Microeconomics: The Case of Decision Trees and Discrete Choice. *arXiv:1711.04826 [stat]*. arXiv: 1711.04826.

Conn, A., Gould, N., and Toint, P. (2000). *Trust Region Methods*. MOS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics.

Danalet, A. and Mathys, N. (2018). Mobility Resources in Switzerland in 2015. In *Proceedings of the 18th Swiss Transport Research Conference*, Ascona, Switzerland.

Defazio, A., Bach, F., and Lacoste-Julien, S. (2014). SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 1646–1654. Curran Associates, Inc.

Dennis, J. and Schnabel, R. (1996). *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics.

Devarakonda, A., Naumov, M., and Garland, M. (2017). AdaBatch: Adaptive Batch Sizes for Training Deep Neural Networks. *arXiv:1712.02029 [cs, stat]*. arXiv: 1712.02029.

Dolan, E. D. and Moré, J. J. (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213.

Dozat, T. (2016). Incorporating Nesterov Momentum into Adam.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

Fletcher, R. (1987). *Practical Methods of Optimization; (2Nd Ed.)*. Wiley-Interscience, New York, NY, USA.

Gower, R. M., Hanzely, F., Richtárik, P., and Stich, S. (2018). Accelerated Stochastic Matrix Inversion: General Theory and Speeding up BFGS Rules for Faster Second-Order Optimization. *arXiv:1802.04079 [cs, math]*. arXiv: 1802.04079.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. (2018). Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv:1706.02677 [cs]*. arXiv: 1706.02677.

Hillel, T. (2019). *Understanding travel mode choice: A new approach for city scale simulation*. PhD thesis, University of Cambridge, Cambridge.

Hillel, T., Elshafie, M. Z. E. B., and Jin, Y. (2018). Recreating passenger mode choice-sets for transport simulation: A case study of London, UK. *Proceedings of the Institution of Civil Engineers - Smart Infrastructure and Construction*, 171(1):29–42.

Jones, E., Oliphant, T., and Peterson, P. (2014). SciPy: open source scientific tools for Python.

Keskar, N. S. and Berahas, A. S. (2016). adaQN: An Adaptive Quasi-Newton Algorithm for Training RNNs. In *Machine Learning and Knowledge Discovery in Databases*, Lecture Notes in Computer Science, pages 1–16. Springer, Cham.

Kingma, D. P. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*. arXiv: 1412.6980.

Kiros, R. (2013). Training Neural Networks with Stochastic Hessian-Free Optimization. *arXiv:1301.3641 [cs, stat]*. arXiv: 1301.3641.

Lederrey, G., Lurkin, V., and Bierlaire, M. (2018a). Optimization of Discrete Choice Models using first-order methods. In *Proceedings of the 7th Symposium of the European Association for Research in Transportation*, Athens, Greece.

Lederrey, G., Lurkin, V., and Bierlaire, M. (2018b). SNM: Stochastic Newton Method for Optimization of Discrete Choice Models. *IEEE - ITSC'18*.

Martens, J. (2010). Deep learning via Hessian-free optimization. *ICML*, 27:735–742.

Mokhtari, A. and Ribeiro, A. (2014). RES: Regularized Stochastic BFGS Algorithm. *IEEE Transactions on Signal Processing*, 62(23):6089–6104.

Mutny, M. (2016). Stochastic Second-Order Optimization via von Neumann Series. *arXiv:1612.04694 [cs, math]*. arXiv: 1612.04694.

Nesterov, Y. E. (1983). A method for solving the convex programming problem with convergence rate O (1/k^ 2). In *Dokl. Akad. Nauk SSSR*, volume 269, pages 543–547.

Newman, J. P., Lurkin, V., and Garrow, L. A. (2018). Computational methods for estimating multinomial, nested, and cross-nested logit models that account for semi-aggregate data. *Journal of Choice Modelling*, 26:28–40.

Polyak, B. and Juditsky, A. (1992). Acceleration of Stochastic Approximation by Averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855.

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17.

Rafati, J., DeGuchy, O., and Marcia, R. F. (2018). Trust-Region Minimization Algorithm for Training Responses (TRMinATR): The Rise of Machine Learning Techniques. In *2018 26th European Signal Processing Conference (EUSIPCO)*, pages 2015–2019. ISSN: 2076-1465, 2219-5491.

Reddi, S. J., Kale, S., and Kumar, S. (2018). On the Convergence of Adam and Beyond.

Robbins, H. and Monro, S. (1951). A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400–407.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv:1609.04747 [cs]*. arXiv: 1609.04747.

Schmidt, M., Roux, N. L., and Bach, F. (2013). Minimizing Finite Sums with the Stochastic Average Gradient. *arXiv:1309.2388 [cs, math, stat]*. arXiv: 1309.2388.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.

Wang, X., Ma, S., and Liu, W. (2014). Stochastic Quasi-Newton Methods for Nonconvex Stochastic Optimization. *arXiv:1412.1196 [math]*. arXiv: 1412.1196.

Wolfe, P. (1969). Convergence Conditions for Ascent Methods. *SIAM Review*, 11(2):226–235.

Wolfe, P. (1971). Convergence Conditions for Ascent Methods. II: Some Corrections. *SIAM Review*, 13(2):185–188.

Wu, C. W. (2019). ProdSumNet: reducing model parameters in deep neural networks via product-of-sums matrix decompositions. *arXiv:1809.02209 [cs, stat]*. arXiv: 1809.02209.

Ye, H. and Zhang, Z. (2017). Nestrov's Acceleration For Second Order Method. *arXiv:1705.07171 [cs]*. arXiv: 1705.07171.

You, Z. and Xu, B. (2014). Investigation of stochastic Hessian-Free optimization in Deep neural networks for speech recognition. In *The 9th International Symposium on Chinese Spoken Language Processing*, pages 450–453.

Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method. *arXiv:1212.5701 [cs]*. arXiv: 1212.5701.