Mathematical Programming with Equilibrium Constraints: An Uncertainty Perspective

Master Thesis Presented to the

Department of Economics at the

Rheinische Friedrich-Wilhelms-Universität Bonn

in Partial Fulfillment of the Requirements for the Degree of

Master of Science (M.Sc.)

Supervisor: Prof. Dr. Philipp Eisenhauer

Submitted in August 2020 by:
Pascal Heid
Matriculation Number: 3220685

Contents

Abbreviations

Term	Meaning
VI	Variational Inequality
NFXP	Nested Fixed Point Algorithm
MPEC	Mathematical Programming with Equilibrium Constraints
N – K	Newton-Kantorovich
BHHH	Berndt-Hall-Hall-Hausman Algorithm

List of Figures

List of Tables

List of Algorithms

1 Mathematical Programming with Equilibrium Constraints

Mathematical Programming with Equilibrium Constraints can be traced back notation and concept wise to Game Theory. Specifically in the theory on Stackelberg games MPEC found its first development. It was due to Luo et al. (1996), though, that MPEC was set onto a mathematically rigorous foundation. They argue to have done so in order to present its manifold possibilities of application that had been overlooked previously.

Before we go into the details of MPEC regarding its mathematical formulation, applications and use in Economics, let us break down the lengthy term into its key components. The beginning "Mathematical Program" solely captures that we look at a mathematical optimization problem. The particularity of this problem comes in with the "Equilibrium Constraints". Mathematically this means that this optimization problem is subject to variational inequalities (VI) as constraints. Nagurney (1993) explains that VIs consist of but are not limited to nonlinear equations, optimization as well as fixed point problems. More broadly spoken VIs are able to harnesses our intuitive notion of economic equilibrium for which typically a functional or a system of equations must be solved for all possible values of a given input. This is tightly linked to what is looked for when solving a Stackelberg game. Essentially, an economic equilibrium has to be found. As a reminder in a Stackelberg game, there is one leader that moves first followed by the moves of some followers. Solving this problem involves the leader to solve an optimization problem that in turn is subject to an optimization procedure of the followers given every possible optimal value the leader might find. The variational inequality here is the problem of the followers which involves solving a decision problem for every possible move of the leader and which is cast into the optimization problem of the leader as a constraint. It can be seen from the fact that the leader moves first and followers move after, as noted by Luo et al. (1996), that the MPEC formulation is a hierarchical mathematical concept which captures multi-level optimization and hence can prove useful for the modeling of decision-making processes. They further explain that this feature can further be beneficial in other fields than just Economics. They showcase that a classification problem in machine learning can be formulated as an MPEC and they further describe some problems in robotics, chemical engineering and transportation networks in MPEC notation.

While this discussion shows that MPEC problems appear in theoretical Economics, Su and Judd (2012) enter with the novel idea to formulate an estimation procedure in structural econometrics as a MPEC. In the following I present their idea using the notation they originally suggested.

In order to estimate the structural parameters of an economic model using data, researchers commonly rely on the Generalized Method of Moments or maximum likelihood estimation. If the researchers opt for the most complex way of estimation (as opposed to using methods lowering the computational burden such as in Hotz and Miller (1993)) which involves solving the economic model at each guess of the structural parameters, they frequently employ the nested fixed point algorithm (NFXP) suggested by Rust (1987). In the case of maximum likelihood estimation, the approach works like the following: An unconstrained optimization algorithm guesses the structural parameters and for each of those guesses the underlying economic model is solved. The resulting outcome of the economic model allows to evaluate the maximum likelihood which then gives new information to the optimization algorithm to form a new guess of the structural parameters. This is repeated until some stopping criteria is met. To make it more explicit, let us introduce some mathematical notation. Let us assume that an economic model is described by some structural parameter vector θ and a state vector x as well as some endogenous vector σ. Assume we further observe some data consisting of $X = \{x_i, d_i\}_{i=1}^{M}$. Here, x_i is the observed state and d_i is the observed equilibrium outcome of the underlying economic decision model. M is the number of data points.

Let us further assume that generally σ depends on the parameters θ through a set of equilibrium conditions (or in the previous notation of variational inequalities), i.e. $\sigma(\theta)$. This includes e.g. Bellman equations. The consistency of σ with θ is expressed by the following condition:

$$h(\theta, \sigma) = 0.$$

For a given θ, let $\Sigma(\theta)$ denote the set of $\sigma(\theta)$ for which the equilibrium conditions hold, i.e. for which $h(\theta, \sigma) = 0$.

$$\Sigma(\theta) := \{\sigma : h(\theta, \sigma) = 0\}.$$

Let $\hat{\sigma}(\theta)$ denote an element of the above set. In the case of an infinite horizon dynamic discrete-choice model, this represents the expected value function evaluated at a specific parameter vector θ. In the case that a unique fixed point for the expected value function exists, $\hat{\sigma}(\theta)$ would be a single value but this does not have to hold in general. If the equilibrium condition involves solving a game for instance, one could easily imagine to find multiple equilibria which causes $\Sigma(\theta)$ to have multiple elements for a given θ.

For the case of multiple $\hat{\sigma}(\theta)$ the solution to the maximization of the log likelihood function $L(.)$ given the data X becomes:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \{ \max_{\hat{\sigma}(\theta) \in \Sigma(\theta)} L(\theta, \hat{\sigma}(\theta); X) \}. \tag{1}$$

This shows that the above problem boils down to finding the parameter vector θ that gives out possibly several $\hat{\sigma}(\theta)$ and which yields in combination with one of them the highest possible log likelihood of all combinations of θ and $\hat{\sigma}(\theta)$.

As already shortly described, the NFXP attempts to solve this problem in a nested loop. First, a guess for $\hat{\theta}$ is fixed for which the corresponding $\hat{\sigma}(\hat{\theta})$ (possibly multiple) are found. For those possibly multiple combinations of $\hat{\theta}$ and $\hat{\sigma}(\hat{\theta})$ the one that yields the highest log likelihood is chosen and this procedure is repeated until the $\hat{\theta}$ is found that solves equation (1). The NFXP therefore solves this problem by running an unconstrained optimization of the log likelihood function that involves solving the economic model at each parameter guess. For the simplified version of $\hat{\sigma}(\hat{\theta})$ being single-valued this idea is captured in the following pseudocode:

Algorithm 1: Nested Fixed Point Algorithm

```
Input: $\hat{\theta}_n$, $n = 0$, $X$;
while $f(||\hat{\theta}_{n+1} - \hat{\theta}_n||) \geq$ stopping tolerance do
    Calculate $\hat{\sigma}(\hat{\theta}_n)$ and evaluate $L(\hat{\theta}_n, \hat{\sigma}(\hat{\theta}_n); X)\}$;
    Based on that fix a new guess $\hat{\theta}_{n+1}$;
end
```

The above formulation clearly conveys two points already. The problem posed in equation (1) is essentially a hierarchical one. Additionally, we work with equilibrium conditions. This gives an indication that an MPEC formulation of the above problem might exist. Su and Judd (2012) formally prove exactly this. The difference to the NFXP way of writing the problem is that one know ensures differently that a guess of θ is consistent with the equilibrium condition $h(\theta, \sigma) = 0$. In the MPEC formulation σ is modeled explicitly as another parameter vector that can be chosen freely by an optimization algorithm instead of being derived from θ. This gives rise to a new log likelihood function $L(\theta, \sigma; X)$ for which they coin the term *augmented likelihood function.* Still they have to make sure, though, that the equilibrium condition holds meaning that the parameter guess for θ is consistent with the equilibrium σ. This is done by imposing it as a constraint to the augmented log likelihood function. The optimization problem now becomes a constrained optimization looking like the following:

$$\begin{aligned} &\max_{(\theta,\sigma)} L(\theta, \sigma; X) \\ &\text{subject to } h(\theta, \sigma) = 0. \end{aligned} \tag{2}$$

Su and Judd (2012) provide a proof that the two formulations in the equations (1) and (13) are actually equivalent in the sense that they yield the same solution $\hat{\theta}$ for the structural parameters of the model. The general setup of the algorithm used for MPEC simplifies to the following:

Algorithm 2: Mathematical Programming with Equilibrium Constraints

```
Input: $\hat{\theta}_n$, $\hat{\sigma}_n$, $n = 0$, $X$;
while $f(||(\hat{\theta}_{n+1}, \hat{\sigma}_{n+1}) - (\hat{\theta}_n, \hat{\sigma}_n)||) \geq$ stopping tolerance do
    Evaluate $L(\hat{\theta}_n, \hat{\sigma}_n; X)$;
    Based on that fix a new guess $(\hat{\theta}_{n+1}, \hat{\sigma}_{n+1})$;
end
```

Having established that the two algorithms or formulations theoretically yield the same solution for the structural parameters, Dong et al. (2017) note that the different way they achieve that can be characterized in the following way: The NFXP solves the problem with an unconstrained optimization algorithm by posing the problem as a low dimensional one. The MPEC formulation on the other hand is a high dimensional problem that needs to be solved using an optimizer that can handle constrained optimization problems involving nonlinear constraints. The difference in dimensionality stems from the fact that in the MPEC also the equilibrium variables need to be chosen. This observation automatically raises the question whether there is any advantage MPEC might have over NFXP as at first sight the problem seems to be more complicated. Su and Judd (2012) identify one major advantage which rests on the fact that the solving of the economic model does not have to be taken care of by the researcher but is cast to the optimization algorithm. The first immediate advantage comes from less coding effort. In the case of the problem of infinite-horizon dynamic discrete choice posed in Rust (1987) which Su and Judd base their comparison on, this makes a significant difference. Another advantage comes from the way modern solvers such as KNITRO (based on Byrd et al. (2006)) or IPOPT (see Pirnay et al. (2011)) handle constraints. Those constraints are not solved exactly until the last guess of the structural parameters which allows them to potentially perform faster than the NFXP in which at each guess of the structural parameters θ is calculated with high precision. This can especially be a factor when the underlying model is quite complicated such as for instance a game with multiple equilibria. Dubé et al. (2012), who look at the NFXP and MPEC for a BLP demand model, see another more practical factor that might make the case for MPEC. They report that practioners tend to loosen the convergence tolerance for solving the economic model when using the NFXP in order to speed up the process (especially when the model is computationally intensive). This leads to an increasing numerical error in the equilibrium outcome which might propagate into the guess of the structural parameters as the equilibrium outcome influences the likelihood function. This can result in wrong parameter estimates or even in failure of convergence. They further report that the existing literature on durable and semi-durable good markets might profit from MPEC as there are models that would need three nested loops when using the NFXP but two of them could be easily cast into the constraints of one major loop when opting for MPEC.

MPEC has one key limitation, though, that is mentioned by several different authors. Wright (2004) reports that the speed of modern solvers based on interior point algorithms (such as the before mentioned KNITRO and IPOPT) crucially depends on the sparsity of the Jacobian and the Hessian of the Langrangian. This highlights that the higher dimensionality (the size of the Jacobian and the Hessian) of MPEC problems does not need to generally cause a problem but if it comes with few zero elements in the before mentioned matrices it might, i.e. when those matrices are rather dense. This, in turn, depends on the economic model at hand and hence gives an indication that whether NFXP or MPEC should be prefered might depend on the specific context. This is confirmed by Dubé et al. who find that MPEC is faster and more reliable (looking at the convergence rate) than the NFXP but for problems that cause the constraint Jacobian and Hessian to be sparse but this advantage deteriorates when having dense matrices. Jørgensen (2013) confirms this for the case of estimating a continuous choice model. He states that MPEC needs too much memory when the state space is large and the before mentioned matrices are dense. In a more recent study Dong et al. (2017) compare MPEC and NFXP for an empirical matching model. They obtain a more fine-grained image of the previously noted tradeoff. In their estimation, they solve the same model with a more sparse and a more dense version of MPEC. They obtain this difference by first setting up MPEC with all equilibrium conditions as constraints (sparse version) and then again with a version where they substitute in some of the equilibrium conditions into the other ones (dense version). For the comparison of the two, they find that the sparse version has better convergence rates while the dense version has a speed advantage. The authors observe another interesting element when comparing NFXP and MPEC. In their application the inner loop can potentially fail depending on the structural parameter guess provided. This adds another problematic element to the use of the NFXP. This is due to the set up of separating the structural guess from the solving of the model. The optimizer cannot take into account whether a structural parameter guess might cause the inner loop (the solving of the economic model) to fail. This is different for the MPEC formulation in which the algorithm can jointly consider the structural parameter and the equilibrium outcome.

2 The Rust Model

This section follows up on the previous one in the sense that it introduces the model created by Rust (1987) and presents how NFXP and MPEC can be used to estimate its model parameters. My notation is mainly inspired by the one employed in Su and Judd (2012).

2.1 The Economic Model

Rust's model is based on the decision making process of Harold Zurcher who is in charge of a bus fleet and has to decide in each period $t = 0, 1, 2, ...$ whether to replace the engine ($d_t^i = 1$) of one or more buses $i = 1, 2, ..., M$ in his fleet or otherwise repair them in a less costly way ($d_t^i = 0$). The agent, hence, chooses from the discrete action space $\mathcal{D} = \{0, 1\}$. This choice is assumed to be independent across buses and he is basing it on two state variables which are the observed cumulative mileage x_t^i of a bus since the last engine replacement and some unobserved (by the econometrician) factor ϵ_t^i. The state of a bus i in period t is therefore fully described by $(x_t^i, \epsilon_t^i) \in \mathcal{S}$ with $\mathcal{S}$ being the state space. The agent receives an immediate reward in period t from the chosen replacement decision $d_t^i(x_t^i, \epsilon_t^i)$. The choice in turn affects the possible state space $\mathcal{S}'$ in the period $t + 1$ as the cumulative mileage after replacement x^i_{t+1} depends on the choice of d^i_t. As the agent is forward looking with a discount factor $\beta \in (0, 1)$ he does not simply maximize the immediate reward but rather the expected discounted utility over an infinite horizon with a higher preference for reward occurring closer to the present. The immediate reward is additively separable can be characterized in the following way:

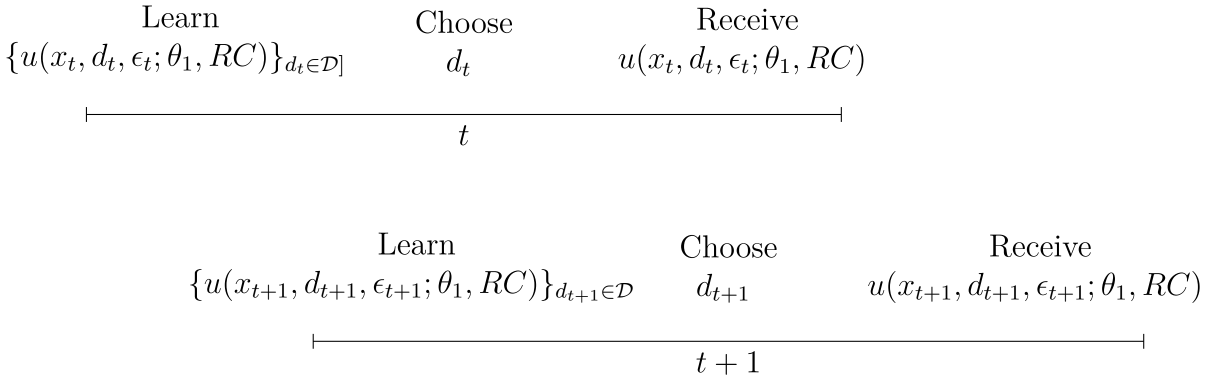
$$u(x^i_t, d^i_t, \epsilon^i_t; \theta_1, RC) = v(x^i_t, d^i_t; \theta_1, RC) + \epsilon^i_t \tag{3}$$

with

$$v(x^i_t, d^i_t; \theta_1, RC) = \begin{cases} -c(x; \theta_1) & \text{if } d^i_t = 0 \\ -RC - c(0; \theta_1) & \text{if } d^i_t = 1 \end{cases}$$

The immediate reward is hence determined by some operating cost $c(.)$ that is increasing in the cumulative mileage state x if regular maintenance as opposed to engine replacement is chosen. If replacement is picked it consists of a replacement cost RC and the operating cost $c(.)$ after resetting the cumulative mileage to zero. This shows that the choice of the agent d^i_t depends crucially on the cost parameters θ_1 and RC. The timing of events for a single bus in the decision process of Harold Zurcher is depicted in Figure 1 below.

Figure 1. Timing of the Decision Model


Learn
$\{u(x_t, d_t, \epsilon_t; \theta_1, RC)\}_{d_t \in \mathcal{D}]}$
Choose
d_t
Receive
$u(x_t, d_t, \epsilon_t; \theta_1, RC)$
t
Learn
$\{u(x_{t+1}, d_{t+1}, \epsilon_{t+1}; \theta_1, RC)\}_{d_{t+1} \in \mathcal{D}}$
Choose
d_{t+1}
Receive
$u(x_{t+1}, d_{t+1}, \epsilon_{t+1}; \theta_1, RC)$
$t+1$

The transition of the state vector $(x^i_t,\, \epsilon^i_t)$ is assumed to follow a Markov process, i.e. the current state only depends on the previous one and hence the utility maximization problem is time-invariant. This means that the problem faced by Zurcher is the same for a given state $(x^i_t,\, \epsilon^i_t)$ irrespective of the time t in which he has to make his choice. Dropping the bus index i for convenience, the optimization problem of the agent gives rise to the following value function for a single bus:

$$V(x_t, \epsilon_t) = \max_{\{d_t, d_{t+1}, \ldots\}} \mathbb{E}\left[\sum_{\tau=t}^{\infty} \beta^{\tau - t} u(x_\tau, d_\tau, \epsilon_\tau; \theta_1, RC)\right] \tag{4}$$

Solving this model leads to an optimal policy rule $\pi^* = (d^{\pi^*}_t(x_t, \epsilon_t))^\infty_t$.

2.2 The Model Solving

The solution to the above model can be characterized by the Bellman equation (Bellman (1954)) below:

$$\begin{aligned} V(x,\epsilon) =& \max_d \{v(x,d;\theta_1,RC) + \epsilon(d) \\ & + \beta \int_{x'} \int_{\epsilon'} V(x',\epsilon') p(x',\epsilon'|x,\epsilon,d,\theta_2,\theta_3) dx' d\epsilon' \} \end{aligned} \tag{5}$$

with (x,ϵ) being the current period and (x',ϵ') the next period state.

Rust (1987) simplifies the above problem now by assuming *conditional independence* on the transition probabilities of the state vector $p(.)$:

$$p(x',\epsilon'|x,\epsilon,d,\theta_2,\theta_3) = p_2(\epsilon'|x';\theta_2) p_3(x'|x,d;\theta_3) \tag{6}$$

with further assuming that $\epsilon(d)$ is following a bivariate i.i.d. extreme value distribution with θ_2 being Euler's constant.

After discretizing the possible values of the state variable x, Rust derives from the original Bellman equation above the following contraction mapping needed to solve the economic model:

$$\begin{aligned} EV(\hat{x}_k,d) =& \sum_{j=0}^{J} log \left\{ \sum_{d' \in \{0,1\}} exp[v(x',d';\theta_1,RC) + \beta EV(x',d')] \right\} \\ & \times p_3(x'|\hat{x}_k,d;\theta_3). \end{aligned} \tag{7}$$

with

$$p_3(x'|\hat{x}_k,d;\theta_3) = \begin{cases} Pr\{x' = \hat{x}_{k+j}|\theta_3\} & \text{if } d = 0 \\ Pr\{x' = \hat{x}_{1+j}|\theta_3\} & \text{if } d = 1 \end{cases}$$

for $j = 0,1,...,J$ indicating how many grid points the mileage state climbs up in the next period.

In the above equation, $EV(.)$ denotes the unique fixed point to a contraction mapping $T_f(EV,\theta)$ on the full state space $\Gamma_f = \{(\hat{x}_k,d)|\hat{x}_k \in \hat{\mathbf{x}}, d \in \mathcal{D}\}$. Here, $\hat{x}_k$ represents the grid point k of the state variable x while $\hat{x}_1 = 0$. The number of possible $\hat{x}_k$ depends on the choice of the grid size K set by the researcher. The set of possible grid points then is denoted as $\hat{\mathbf{x}} = \{\hat{x}_1,...,\hat{x}_K\}$. All the variables v depict current period variables while variables v' display the possible value in the next period. The conditional probability function $p_3(.)$ indicates how likely it is that a bus moves up a specific amount of grid points in the next period depending on the structural parameter θ_3 as well as the choice d. Imagine now we decide to set the grid size to $K = 90$ as done in Rust (1987), then we generally have to find the fixed point above which yields $EV_f = (EV(\hat{x}_1, 0), ..., EV(\hat{x}_{90}, 0), EV(\hat{x}_1, 1), ..., EV(\hat{x}_{90}, 1))$. This simplifies, though, as all the expected values from $EV(\hat{x}_1, 1), ..., EV(\hat{x}_{90}, 1)$ are actually equivalent to $EV(\hat{x}_1, 0)$. This means that in our scenario we just have to find the vector $(EV(\hat{x}_1, 0) \; ... \; EV(\hat{x}_K, 0))^T$ which I denote as EV_r. In our example the vector EV_r for which we have to actually solve the contraction mapping has a dimension of 90. This observation will later be important for the difference between NFXP and MPEC. Su and Judd (2012), hence, denote the dimension-reduced contraction mapping shorthand as:

$$EV_r = T_r(EV_r, \theta) \tag{8}$$

with $T_r(.)$ being a contraction mapping on the reduced state space $\Gamma_r = \{(\hat{x}_k, d = 0)|\hat{x}_k \in \hat{\mathbf{x}}\}$.

The unique fixed point can now be used to derive conditional choice probabilities of the agent:

$$P(d|\hat{x};\theta) = \frac{exp[v(\hat{x}, d; \theta_1, RC) + \beta EV(\hat{x}, d)]}{\sum_{d' \in \{0,1\}} exp[v(\hat{x}, d'; \theta_1, RC) + \beta EV(\hat{x}, d')]}. \tag{9}$$

The above equation describes the probability that the agent chooses d given that the observed mileage state is at a certain grid point $\hat{x}$. This derivation depends on both the cost parameters (θ_1, RC) directly and indirectly through $EV(.)$ on the transition parameter θ_3. These conditional choice probabilities together with the transition probabilities $p_3(.)$ become relevant in the next section when calibrating the model using maximum likelihood.

2.3 Calibration

In order to calibrate the parameter vector $\theta = (\theta_1, \theta_3, RC)$ using either NFXP or MPEC, let us assume that we observe some data set $X = (X^i)_{i=1}^M$ with $X^i = (x_t^i, d_t^i)_{t=1}^T$ for a single bus $i = 1, ..., M$. The data therefore consists of the engine replacement decision per bus and period as well as the cumulative mileage since the last engine replacement. The cumulative mileage x_t^i is assumed to already be discretized which means that it takes values on the grid $\hat{\mathbf{x}} = \{\hat{x}_1, ..., \hat{x}_K\}$. The log likelihood of observing the data X now becomes:

$$L(\theta) = \sum_{i=1}^{M}\sum_{t=2}^{T} log[P(d_t^i|x_t^i;\theta)] + \sum_{i=1}^{M}\sum_{t=2}^{T} log[p_3(x_t^i|x_{t-1}^i, d_{t-1}^i;\theta_3)]. \tag{10}$$

To fulfill the aim of finding the optimal parameter vector θ one now has to solve the problem of maximizing the above log likelihood:

$$\max_{\theta} L(\theta). \tag{11}$$

The path taken by the NFXP is now to hand this unconstrained optimization problem to an optimization algorithm such as the Berndt-Hall-Hall-Hausman (BHHH) algorithm based on Berndt et al. (1974). The algorithm comes up with a guess for the optimal parameter $\hat{\theta}$ for which in a subroutine the expected values in equation **??** are calculated. The expected value function is in turn needed to obtain the conditional choice probabilities in equation 9 which are then taken to evaluate the log likelihood $L(\theta)$. Based on this evaluation, the optimization algorithm comes up with a new guess for $\hat{\theta}$ and the above procedure is repeated until a certain convergence criteria of the algorithm is met. This procedure is again shown in pseudocode in Algorithm 3 on the next page.

At every structural guess of the optimization algorithm the fixed point $EV(.)$ is calculated precisely as it is needed to evaluate the log likelihood $L(\theta)$. This is deemed inefficient by Su and Judd which gives rise to the augmented log likelihood mentioned before for which they insert the conditional choice probabilities $P(d_t^i|x_t^i;\theta)$ into $L(\theta)$ making the log likelihood explicitly depend on $EV(.)$. This results in the following log likelihood.

Algorithm 3: Nested Fixed Point Algorithm for the Rust Model

```
Input: $\hat{\theta}_n$, $n = 0$, $X$;
while $f(||(\hat{\theta}_{n+1}, \hat{EV}_{n+1}) - (\hat{\theta}_n, \hat{EV}_n)||) \geq$ stopping tolerance do
    Solve fixed point
        $EV(\hat{x}_k, d) = \sum_{j=0}^{J} log \left\{ \sum_{d' \in \{0,1\}} exp[v(x', d'; \hat{\theta}_{n,1}, \hat{RC}_n) + \beta EV(x', d')] \right\} \times p_3(x'|\hat{x}_k, d; \hat{\theta}_{n,3});$
    Given the solution to $EV(.)$ calculate
        $P(d|\hat{x}; \hat{\theta}_n) = \frac{exp[v(\hat{x}, d; \hat{\theta}_{n,1}, \hat{RC}_n) + \beta EV(\hat{x}, d)]}{\sum_{d' \in \{0,1\}} exp[v(\hat{x}, d'; \hat{\theta}_{n,1}, \hat{RC}_n)) + \beta EV(\hat{x}, d')]};$
    Evaluate the log likelihood
        $L(\hat{\theta}_n) = \sum_{i=1}^{M} \sum_{t=2}^{T} log[P(d_t^i|x_t^i; \hat{\theta}_n)] + \sum_{i=1}^{M} \sum_{t=2}^{T} log[p_3(x_t^i|x_{t-1}^i, d_{t-1}^i; \hat{\theta}_{n,3})];$
    Based on that fix a new guess $\hat{\theta}_{n+1}$;
end
```

$$
\begin{aligned}
\mathcal{L}(\theta, EV) =& \sum_{i=1}^{M} \sum_{t=2}^{T} log \left[\frac{exp[v(x_t^i, d_t^i, \theta) + \beta EV(x_t^i, d_t^i)]}{\sum_{d' \in \{0,1\}} exp[v(x_t^i, d', \theta) + \beta EV(x_t^i, d')]} \right] \\
& + \sum_{i=1}^{M} \sum_{t=2}^{T} log[p_3(x_t^i|x_{t-1}^i, d_{t-1}^i; \theta_3)]
\end{aligned}
\tag{12}
$$

In this function nothing guarantees that θ and EV are actually consistent. This is healed by imposing the fixed point equation **??** as constraints to the augmented likelihood function. The MPEC formulation of the calibration problem therefore looks like the following.

$$\begin{aligned} &\max_{(\theta, EV)} \mathcal{L}(\theta, EV) \\ &\text{subject to } EV = T(EV, \theta). \end{aligned} \tag{13}$$

For the MPEC formulation the problem is given to an optimization Algorithm that can handle nonlinear equality constraints such as the already mentioned KNITRO or IPOPT. This algorithm then fixes a guess of (θ, EV) that satisfies the nonlinear constraints, i.e. that is consistent with the underlying economic model and for which the augmented log likelihood is evaluated. Based on this evaluation, the optimizer determines a new guess and the procedure starts over. Again this is done until a specific convergence criteria is met. This procedure is illustrated in Algorithm 4 on the next page.

Algorithm 4: MPEC Algorithm for the Rust Model

```
Input: $\hat{\theta}_n$, $\hat{EV}_n$, $n = 0$, $X$;
while $f(||\hat{\theta}_{n+1} - \hat{\theta}_n||) \geq$ stopping tolerance do
    Evaluate the augmented log likelihood
        $\mathcal{L}(\hat{\theta}_n, \hat{EV}_n) = \sum_{i=1}^{M} \sum_{t=2}^{T} log \left[ \frac{exp[v(x_t^i, d_t^i, \hat{\theta}_n) + \beta \hat{EV}_n(x_t^i, d_t^i)]}{\sum_{d' \in \{0,1\}} exp[v(x_t^i, d', \hat{\theta}_n) + \beta \hat{EV}_n(x_t^i, d')]} \right]$
            $+ \sum_{i=1}^{M} \sum_{t=2}^{T} log[p_3(x_t^i | x_{t-1}^i, d_{t-1}^i; \hat{\theta}_{n,3})]$
    Based on that fix a new guess $(\hat{\theta}_{n+1}, \hat{EV}_{n+1})$;
end
```

Having established both NFXP and MPEC for the Rust model, let us now turn to some particularities of the model that might be important for the performance of the two algorithms. First of all, Rust (1987) and Su and Judd (2012) show that both the likelihood for the NFXP and the MPEC are smooth for the Rust model and their first and second order derivatives exist. In the case of the NFXP this means that Newton's method can be used for the maximization problem. This further helps modern solvers such as KNITRO and IPOPT which are developed for smooth optimization problems (compare Byrd et al. (2006) and Wächter (2009)). Another special feature is that solving the model involves finding a fixed point. In the case of the NFXP it is time consuming to find as it involves contraction iterations. This caused Rust to employ a polyalgorithm to find the fixed point using contraction steps at the beginning and switching to Newton-Kantorovich (N-K) steps as soon as a guess is already close to the unique fixed point. This practically speeds up the convergence when searching the fixed point as contraction iterations solely have a linear convergence rate while N-K iterations converge at a quadratic rate when being close to the fixed point (see Rust (1987, 2000)). As already noted before, MPEC on the other hand does not solve the fixed point but instead evaluates it once at every structural parameter guess without high precision until the last iteration of the optimization algorithm. Another factor explained in the general part on MPEC is the high dimensionality of its problem formulation. Going back to our example when the grid size is set to 90, the MPEC formulation yields a problem consisting of 90 nonlinear constraints and $90 + |\theta|$ parameters to estimate. The NFXP has considerably less dimensions as only $|\theta|$ parameters have to be estimated and no constraints have to be considered by the optimizer. Su and Judd (2012) uncover the trade off of dimensionality and fixed point calculation to be the major one between MPEC and NFXP in the Rust model application. Following this line of arguments it is not obviously clear how the chosen grid size affects this trade off as the grid size increases the dimensions of the MPEC problem while also making the fixed point calculation in the NFXP more computationally expensive.

2.4 General Setup and Replication

For the remainder of this thesis I rely on the open-source Python package ruspy which was initially created by Blesch (2019). In this package, he implements Rust's Nested Fixed Point Algorithm based on Rust (2000). I adjusted the code for the NFXP to be capacable of replicating Iskhakov et al. (2016). Further I entirely added the MPEC implementation for the Rust model based on the explanations in the aforementioned paper as well as in Su and Judd (2012) [1] [2]. To grasp why Iskhakov et al. follow up on Su and Judd with a revised setup of MPEC and NFXP for the Rust model, let us go back to the latter paper. After having set up the theoretical implications of MPEC as outlined in the above sections, Su and Judd conclude with a practical implementation of both approaches in a Monte Carlo simulation of the Rust model. Their finding is that MPEC is not only easier to code up but also is significantly faster than the NFXP especially when the discount factor is close to one and has a higher convergence rate. Iskhakov et al. intervene now arguing that the NFXP implementation relied on by Su and Judd is inferior to the one suggested by Rust (2000). This is mainly due to the fact that they do not use the BHHH for the likelihood optimization and further do not employ the polyalgorithm of contraction and Newton-Kantorovich iterations for the fixed point calculation. Iskhakov et al. further add an improvement to the MPEC implementation which I also accounted for in my code. They recenter the expected value function to make it more numerically stable which improves the implementation especially when the discount factor β is very close to one. In a last

[1] Please refer to Pull Requests #46 and #47 to view my contributions to the ruspy package.

[2] The previous version of ruspy only implemented Quasi-Newton Methods based on the scipy library (see Virtanen et al. (2020)). The library does not offer the BHHH algorithm used for the NFXP in Iskhakov et al. (2016). I added the BHHH to the estimagic library (see Gabler (2019)) in the Pull Request #161 in order to use it for ruspy.

step, the authors draw on the same data generating process in a Monte Carlo simulation as was done by Su and Judd and show that with their setup MPEC and NFXP are similarly fast and that both have a very high convergence rate. In my own implementation I stay as closely as possible to the setup employed by Iskhakov et al. given some notable differences that I explain in 2.5 in the appendix.

For the simulation studies in the rest of this thesis, unless otherwise stated, I hence follow the data generating process of Iskhakov et al.. There are 120 time periods, i.e. $T = 120$ and 50 buses, i.e. $M = 50$ that are simulated. The cost function $c(x; \theta_1)$ is assumed to be linear with a scale parameter of 0.001, i.e. $c(x; \theta_{11}) = 0.001 \times \theta_{11} x$. The true parameters are the following:

$$
\begin{aligned}
&RC = 11.7257 \\
&\theta_{11} = 2.4569 \\
&\theta_3 = \begin{pmatrix} 0.0937 & 0.4475 & 0.4459 & 0.0127 & 0.0002 \end{pmatrix}
\end{aligned}
$$

with θ_3 being the Markovian probability that the state of a bus in a given period of time moves up zero, one, two, three or four grid points concerning the mileage state x, respectively. This refers back to equation 9.

As the Rust model does not allow to estimate the discount factor β, it has to be predetermined as well. The true parameter for β is consequently varied in the following setups as it affects how easily the fixed point for $EV(.)$ can be found.

$$\beta \in \{0.975, 0.985, 0.995, 0.999, 0.9995, 0.9999\}$$

For each possible β, 250 data sets following the Rust model with the above parameters are simulated. In order to check for the robustness of the two approaches to different starting values, those are varied and each data set is estimated five times using the following different starting parameter guesses:

$$(RC^0, \theta_{11}^1) \in \{(4, 1), (5, 2), (6, 3), (7, 4), (8, 5)\}.$$

The continuous mileage state is discretized into a grid of 175 points, meaning that the unique fixed point of the expected values EV is 175-dimensional vector. In the case of MPEC, also starting values for this vector have to be passed in which are set to:

$$\begin{pmatrix} EV_1 \\ \vdots \\ EV_{175} \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$$

For MPEC the algorithm IPOPT is utilized while Iskhakov et al. as well as Su and Judd take KNITRO. The latter is a commercial optimizer which cannot be used for a generally open-source package such as ruspy which is why in that package we rely on

NLOPT and IPOPT. IPOPT is used in its default settings while relative stopping tolerance of it is 10^{-6}. I hand over upper bounds of 50 for the EV vector and lower bounds of zero for RC and θ_{11} to the optimizer. Further the first order analytical derivative of the Langrangian of the constrained optimization problem is supplied.

As previously stated, the NFXP is implemented as a polyalgorithm for the fixed point calculation as well as the BHHH for likelihood optimization. For the BHHH I choose a relative stopping tolerance of 10^{-8} as well as 10^{-5} as an absolute stopping tolerance for the algorithm. The switching tolerance from contraction to N-K steps is set to 10^{-2} while the stopping tolerance for the fixed point calculation is fixed at 10^{-13}. In the case of NFXP as well the analytical first order derivative of the likelihood is provided to the optimizer. Now, I use this setup to estimate the structural parameters of the original data sets that have been created by Iskhakov et al. in their Monte Carlo simulation with the above outlined data generating process. The results are presented below.

Table 1. Replication of Iskhakov et al. (2016)

β	Converged (Out of 1250)	CPU Time (in Sec.)	# of Major Iter.	# of Func. Eval.	# of Bellm. Iter.	# of N.K. Iter.
			MPEC-IPOPT			
0.975	1250	1.151	19.6	25.9		
0.985	1250	1.187	19.9	28.4		
0.995	1250	1.352	22.2	35.1		
0.999	1249	1.613	25.3	41.5		
0.9995	1248	1.754	26.1	43.6		
0.9999	1250	1.861	28.1	49.3		
			NFXP			
0.975	1250	1.286	11.8	14.1	301.7	104.4
0.985	1250	1.306	11.4	13.6	291.2	105.6
0.995	1250	1.185	10.6	12.7	272.9	97.7
0.999	1250	1.556	10.9	12.9	278.0	138.4
0.9995	1250	2.626	10.9	12.9	277.6	250.7
0.9999	1250	2.778	10.9	12.9	276.6	270.0

As already mentioned, both MPEC and NFXP are implemented differently and run on a different computer than in the original paper making the absolute comparison of numbers fruitless. What can be seen, though, is that also when using a similar implementation to them with another programming language (they use matlab and AMPL) in the general structure similar results can be achieved. The convergence rates are very high (even higher for my implementation of MPEC than for the one of Iskhakov et al.). The CPU time is on a similar level for both MPEC and NFXP given a specific discount factor. We can observe that the with rising β the fixed point calculation becomes more complex indicated by the rising number of N-K steps needed to solve it. The original implementation is more stable in this regard which is likely explained by the fact that they allow for also back switching from N-K to contraction iterations. For MPEC the fixed point difficulty translates into more iterations and function evaluations needed. As opposed to the original paper where also the analytical Hessian as well as its sparsity pattern and that of the Jacobian of the Lagrangian are provided, my more sparse implementation performs remarkably well.

This section constitutes the starting point of a part of my remaining discussions in which I test how reactive NFXP and MPEC are to certain criteria of the model and how this translates into differences of the above values but also those of another dimension. This other dimension is the uncertainty around a quantity of interest which I will introduce in the upcoming section.

References

Bellman, R. (1954). The theory of dynamic programming. Technical report, Rand corp santa monica ca.

Berndt, E. R., B. H. Hall, R. E. Hall, and J. A. Hausman (1974). Estimation and inference in nonlinear structural models. In *Annals of Economic and Social Measurement, Volume 3, number 4*, pp. 653–665. NBER.

Blesch, M. (2019). ruspy - an open-source package for the simulation and estimation of a prototypical infinite-horizon dynamic discrete choice model based on rust (1987).

Byrd, R. H., J. Nocedal, and R. A. Waltz (2006). *Knitro: An Integrated Package for Nonlinear Optimization*, pp. 35–59. Boston, MA: Springer US.

Dong, B., Y.-W. Hsieh, and X. Zhang (2017). Implementing maximum likelihood estimation of empirical matching models: Mpec versus nfxp. Technical report, USC-INET Research Paper No. 17-16.

Dubé, J.-P., J. T. Fox, and C.-L. Su (2012). Improving the numerical performance of static and dynamic aggregate discrete choice random coefficients demand estimation. *Econometrica 80*(5), 2231–2267.

Gabler, J. (2019). A python tool for the estimation of (structural) econometric models.

Hotz, V. J. and R. A. Miller (1993). Conditional choice probabilities and the estimation of dynamic models. *The Review of Economic Studies 60*(3), 497–529.

Iskhakov, F., J. Lee, J. Rust, B. Schjerning, and K. Seo (2016). Comment on “constrained optimization approaches to estimation of structural models”. *Econometrica 84*(1), 365–370.

Jørgensen, T. H. (2013). Structural estimation of continuous choice models: Evaluating the egm and mpec. *Economics Letters 119*(3), 287 – 290.

Luo, Z.-Q., J.-S. Pang, and D. Ralph (1996). *Mathematical Programs with Equilibrium Constraints*. Cambridge University Press.

Nagurney, A. (1993). *Variational Inequality Theory*, pp. 3–37. Dordrecht: Springer Netherlands.

Pirnay, H., R. Lopez-Negrete, and L. Biegler (2011, 10). Optimal sensitivity based on ipopt. *Mathematical Programming Computation 4*.

Rust, J. (1987). Optimal replacement of gmc bus engines: An empirical model of harold zurcher. *Econometrica 55*(5), 999–1033.

Rust, J. (2000). Nested fixed point algorithm documentation manual.

Su, C.-L. and K. L. Judd (2012). Constrained optimization approaches to estimation of structural models. *Econometrica 80*(5), 2213–2230.

Virtanen, P., R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. . . Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*.

Wächter, A. (2009). Short tutorial: getting started with ipopt in 90 minutes. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Wright, M. (2004, 09). The interior-point revolution in optimization: History, recent developments, and lasting consequences. *Bulletin of The American Mathematical Society - BULL AMER MATH SOC 42*, 39–57.

2.5 Appendix A

This part explains to which extent my implementations of the NFXP and MPEC differ to the ones by Iskhakov et al. (2016). As already laid out in section 2.4, we rely entirely on open-source programs which has some implications for our methodology. The authors use matlab for the NFXP (they also implement the BHHH like this) and the modeling language AMPL in combination with the solver KNITRO for MPEC. I, on the other hand, implement everything in Python only and use IPOPT as a solver for MPEC. This alone can cause the results of them and mine to differ. Additionally, for MPEC they obtain first and second order analytical derivatives of the Lagrangian as AMPL provides them using automatic differentiation. While there are tools such as JAX [3] that provide analytical derivatives via automatic differentiation, these restrict the code to have a certain form which would have involved rewriting main parts of the existing code. This might be an interesting extension of the package for later but is out of bounds for this thesis. I decided therefore to at least code up the first order derivative and pass it on to IPOPT. The fact that the Hessian has to be approximated can potentially influence the results strongly. Iskhakov et al. also give sparsity patterns of the two derivatives to KNITRO in order to conserve memory and increase speed. This is not done in my implementation. Another practical difference comes from the fact that KNITRO and IPOPT rely on different stopping criteria which makes it impossible exactly replicate the setup for KNITRO with IPOPT. For the NFXP there are also some differences. Again, the tolerances used in the original paper are not exactly replicated. My BHHH as well as the switching from contraction to N-K iterations has other tolerances. On top of that, Iskhakov et al. allow the fixed point algorithm to switch back from N-K to contraction steps when a certain criteria is met. This flexibility is not implemented in my code.

In the light of that, the algorithms by Iskhakov et al. are slightly more complex and robust which seems to affect the number of iterations and function evaluations for MPEC and especially the amount of contraction and N-K steps needed in the case of the NFXP. This can be observed in Table 1. Although one should mention that the more comparable implementation of MPEC in which Su and Judd (2012) use matlab as the modeling language and KNITRO with only first order analytical derivatives seems to be inferior to my implementation when looking at the number of iterations and function evaluations needed [4]. For an additional sanity check I provide the mean and standard deviations of the estimated cost parameters across the Monte Carlo simulations outlined in section 2.4. These are meaningful as Su and Judd (2012) argue that this simulation actually constitutes a parametric bootstrap procedure. The two statistics are provided for both my implementations and additionally for NFXP of Iskhakov et al.. These results were not published but had to be obtained by me using their matlab replication code.

[3] See https://github.com/google/jax .

[4] To see this, have a look at Table II on page 2228 of Su and Judd (2012). One has to be cautious, though, as their implementation differs from mine in the sense that they do not recenter the expected value function.

Unfortunately, their code does not run through but only did with some additional changes. This should make one a bit cautious regarding these results. All of those results are presented in the table below in which can be seen that apart from when β equals 0.995 the results of all three approaches are very similar.

Table 2. Comparison to the Results of Iskhakov et al. (2016)

			RC	θ_{11}
β	Implementation	True Values:	**11.726**	**2.457**
0.975	MPEC	Mean	11.908	2.507
		Std. Dev.	(1.517)	(0.486)
	NFXP	Mean	11.908	2.507
		Std. Dev.	(1.517)	(0.468)
	NFXP Iskhakov	Mean	11.914	2.508
		Std. Dev.	(1.517)	(0.468)
0.985	MPEC	Mean	11.986	2.534
		Std. Dev.	(1.457)	(0.452)
	NFXP	Mean	11.986	2.534
		Std. Dev.	(1.457)	(0.452)
	NFXP Iskhakov	Mean	11.991	2.535
		Std. Dev.	(1.457)	(0.452)
0.995	MPEC	Mean	11.891	2.508
		Std. Dev.	(1.384)	(0.440)
	NFXP	Mean	11.891	2.508
		Std. Dev.	(1.384)	(0.440)
	NFXP Iskhakov	Mean	11.191	2.902
		Std. Dev.	(1.188)	(0.473)
0.999	MPEC	Mean	11.874	2.513
		Std. Dev.	(1.347)	(0.444)
	NFXP	Mean	11.874	2.513
		Std. Dev.	(1.347)	(0.444)
	NFXP Iskhakov	Mean	11.876	2.513
		Std. Dev.	(1.346)	(0.444)
0.9995	MPEC	Mean	11.849	2.509
		Std. Dev.	(1.343)	(0.445)
	NFXP	Mean	11.847	2.508
		Std. Dev.	(1.343)	(0.445)
	NFXP Iskhakov	Mean	11.849	2.509
		Std. Dev.	(1.342)	(0.445)
0.9999	MPEC	Mean	11.815	2.498
		Std. Dev.	(1.319)	(0.431)
	NFXP	Mean	11.815	2.499
		Std. Dev.	(1.319)	(0.431)
	NFXP Iskhakov	Mean	11.817	2.499
		Std. Dev.	(1.319)	(0.431)

My implementations overestimate the true parameter slightly for RC while coming gradually closer to the true value with increasing β. This pattern can also be seen in Table I of Su and Judd (2012) (who base their analysis on the same data generating process) which makes me confident that my results are correct. It seems like for $\beta = 0.995$ the NFXP Iskhakov is trapped in another local minimum that causes it to overestimate the true θ_{11} by a lot and suddenly underestimate the true RC. As mentioned before, this is likely to be caused by the fact that their code did not run through and it is not obviously clear whether the setup in the code is exactly like the one the published results are based on.

Affidavit

"I hereby confirm that the work presented has been performed and interpreted solely by myself except for where I explicitly identified the contrary. I assure that this work has not been presented in any other form for the fulfillment of any other degree or qualification. Ideas taken from other works in letter and in spirit are identified in every single case."

Place, Date

Signature