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 **I**

List of Figures **II**

List of Tables **III**

List of Algorithms **IV**

1 Mathematical Programming with Equilibrium Constraints **1**

2 The Rust Model **5**

2.1 The Economic Model 5

2.2 The Model Solving 7

2.3 Calibration 9

2.4 General Setup and Replication 11

3 Uncertainty Quantification **14**

3.1 The UQ Framework 16

3.2 The Quantity of Interest: The Demand Function 18

3.3 Uncertainty Propagation of the Simulation in Iskhakov et al. (2016) 19

4 Sensitivity of the Rust Model **21**

4.1 The Simulation Setup 22

4.2 The Results 26

4.2.1 The Partial Perspective 26

4.2.2 The Full Perspective 32

4.2.3 Discussion 35

References **37**

Appendix **40**

4.3 Appendix A: Tables 40

4.4 Appendix B: Figures 43

4.5 Appendix C: Further Details on the Replication of Iskhakov et al. (2016) . 44

4.6 Appendix D: MPEC for respy 46

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
UQ	Uncertainty Quantification

List of Figures

1 Timing of the Decision Model 6
2 Joint Distribution of $\hat{RC}$ and $\hat{\theta}_{11}$ 19
3 Distribution of the QoI 20
4 The Implied Demand Function accounting for Parameter Uncertainty . . . 21
5 The QoI when varying the Cost Function 27
6 The QoI when varying β and the Cost Function 28
7 The QoI when varying the Grid Size 29
8 The QoI varying the Grid Size and the Gradients 30
9 The QoI for a Sequence of Specifications 32
10 The MSE of the QoI across all Specifications 33
11 The PDF and CDF of the QoI using MPEC 34
12 The QoI when varying the Cost Function (BHHH) 43
13 The PDF and CDF of the QoI with the NFXP 43

List of Tables

1 Replication of Iskhakov et al. (2016) 13
2 Correlation between Parameter Estimates 40
3 The Specifications for the Sensitivity Simulation 41
4 Mean Quantitative Comparison for Specifications of Figure 8 42
5 Comparison to the Results of Iskhakov et al. (2016) 45

List of Algorithms

1 Nested Fixed Point Algorithm 3
2 Mathematical Programming with Equilibrium Constraints 4
3 Nested Fixed Point Algorithm for the Rust Model 10
4 MPEC Algorithm for the Rust Model 11

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 formulate 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 set up 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 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:

$$h(\theta, \sigma) = 0. \tag{1}$$

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

$$\Sigma(\theta) := \{\sigma : h(\theta, \sigma) = 0\}. \tag{2}$$

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:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\{\max_{\hat{\sigma}(\theta)\in\Sigma(\theta)} L(\theta, \hat{\sigma}(\theta); X)\}. \tag{3}$$

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, in the outer loop a guess for $\hat{\theta}$ is fixed for which in the inner loop 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 (3). 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 where n depicts the iteration number in the outer loop:

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 (3) 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 now 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 Su and Judd 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 an equilibrium guess of σ. This is done by imposing the equilibrium condition $h(.)$ as a constraint to the augmented log likelihood function. The optimization problem now becomes a constrained optimization looking like the following:

$$\begin{aligned} &\max_{(\theta,\sigma)} L(\theta, \sigma; X) \\ &\text{subject to } h(\theta, \sigma) = 0. \end{aligned} \tag{4}$$

Su and Judd (2012) provide a proof that the two formulations in the equations (3) and (4) 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 single loop (as the inner loop for the calculation of the economic equilibrium is avoided):

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 harder to solve. Su and Judd (2012) identify one major advantage of MPEC 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 σ 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 expensive). 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 making the previously noted advantages for MPEC more pronounced.

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 derived from the constrained optimization problem. This highlights that the increased 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 for problems with a sparse constraint Jacobian and Hessian 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 matching 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 others (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 month $t = 0, 1, 2, ...$ 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 Zurcher bases his decision 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_{t+1}^i depends on the choice of d_t^i. The decision problem is essentially regenerative as the mileage state is set back to zero after the decision to replace the engine of a bus. 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:

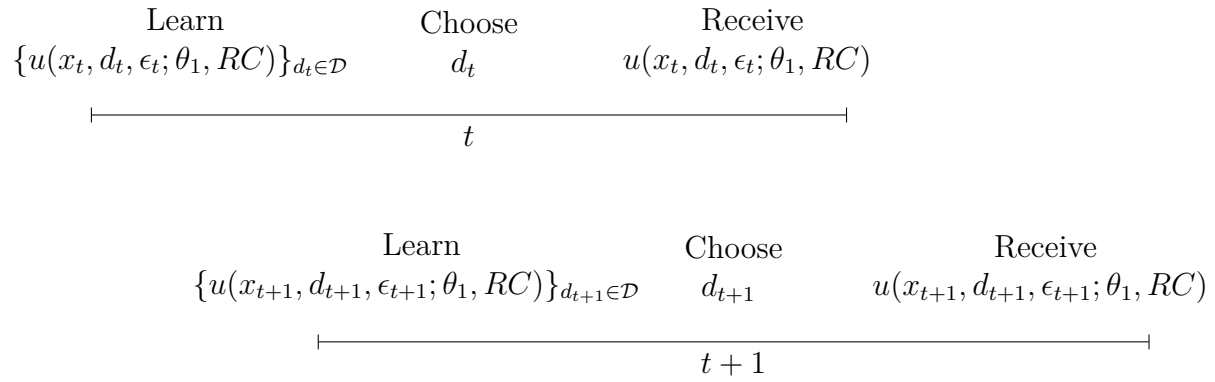
$$u(x_t^i, d_t^i, \epsilon_t^i; \theta_1, RC) = v(x_t^i, d_t^i; \theta_1, RC) + \epsilon_t^i \tag{5}$$

with

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

The immediate reward is hence determined by some operating cost $c(.)$ that is assumed to be 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_t^i depends crucially on the structural cost parameters θ_1 and RC that define the exact shape of the cost function. 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



The transition of the state vector $(x_t^i,\ \epsilon_t^i)$ 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 $(x_t^i,\ \epsilon_t^i)$ 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:

$$V(x_t, \epsilon_t) = \max_{\{d_t, d_{t+1}, \ldots\}} \mathbb{E}\left[\sum_{\tau=t}^{\infty} \beta^{\tau-t} u(x_\tau, d_\tau, \epsilon_\tau; \theta_1, RC)\right] \tag{6}$$

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

2.2 The Model Solving

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

$$\begin{aligned} V(x, \epsilon) = \max_d \{ & v(x, d; \theta_1, RC) + \epsilon(d) \\ & + \beta \int_{x'} \int_{\epsilon'} V(x', \epsilon') p(x', \epsilon' | x, \epsilon, d, \theta_2, \theta_3) dx' d\epsilon' \} \end{aligned} \tag{7}$$

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 $p(.)$:

$$p(x', \epsilon' | x, \epsilon, d, \theta_2, \theta_3) = p_2(\epsilon' | x'; \theta_2) p_3(x' | x, d; \theta_3) \tag{8}$$

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

$$\begin{aligned} EV(\hat{x}_k, d) = & \sum_{j=0}^{J} log \left\{ \sum_{d' \in \{0,1\}} exp[v(x', d'; \theta_1, RC) + \beta EV(x', d')] \right\} \\ & \times p_3(x' | \hat{x}_k, d; \theta_3). \end{aligned} \tag{9}$$

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 the solution $EV_f = (EV(\hat{x}_1, 0), ..., EV(\hat{x}_{90}, 0), EV(\hat{x}_1, 1), ..., EV(\hat{x}_{90}, 1))$. This simplifies, though, as all the expected values from $EV(\hat{x}_1, 1), ..., EV(\hat{x}_{90}, 1)$ are actually equivalent to $EV(\hat{x}_1, 0)$ due to the regenerative nature of the decision problem. This means that in our scenario we just have to solve the vector $(EV(\hat{x}_1, 0)\ ...\ EV(\hat{x}_K, 0))^T$ which I denote as EV_r and we have the whole vector EV_f at hand. In our example the vector EV_r 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:

$$EV_r = T_r(EV_r, \theta) \tag{10}$$

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:

$$P(d|\hat{x}; \theta) = \frac{exp[v(\hat{x}, d; \theta_1, RC) + \beta EV(\hat{x}, d)]}{\sum_{d' \in \{0,1\}} exp[v(\hat{x}, d'; \theta_1, RC) + \beta EV(\hat{x}, d')]}. \tag{11}$$

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 $EV(.)$ as well as 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:

$$L(\theta) = \sum_{i=1}^{M} \sum_{t=2}^{T} log[P(d_t^i | x_t^i; \theta)] + \sum_{i=1}^{M} \sum_{t=2}^{T} log[p_3(x_t^i | x_{t-1}^i, d_{t-1}^i; \theta_3)]. \tag{12}$$

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

$$\max_{\theta} L(\theta). \tag{13}$$

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 9 are calculated. The expected value function is in turn needed to obtain the conditional choice probabilities in equation 11 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:

$$\begin{aligned} \mathcal{L}(\theta, EV) = &\sum_{i=1}^{M} \sum_{t=2}^{T} log \left[\frac{exp[v(x_t^i, d_t^i, \theta) + \beta EV(x_t^i, d_t^i)]}{\sum_{d' \in \{0,1\}} exp[v(x_t^i, d', \theta) + \beta EV(x_t^i, d')]} \right] \\ &+ \sum_{i=1}^{M} \sum_{t=2}^{T} log[p_3(x_t^i | x_{t-1}^i, d_{t-1}^i; \theta_3)]. \end{aligned} \tag{14}$$

In this function nothing guarantees that θ and EV are actually consistent. This is healed by imposing the fixed point equation 9 as constraints to the augmented likelihood function. The MPEC formulation of the calibration problem therefore looks like the following.

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

$$\begin{aligned} &\max_{(\theta, EV)} \mathcal{L}(\theta, EV) \\ &\text{subject to } EV = T(EV, \theta). \end{aligned} \tag{15}$$

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.

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. It also 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

Algorithm 4: MPEC Algorithm for the Rust Model

Input: $\hat{\theta}_n$, $E\hat{V}_n$, $n = 0$, X;

while $f(||\hat{\theta}_{n+1} - \hat{\theta}_n||) \geq$ *stopping tolerance* **do**

Evaluate the augmented log likelihood

$$\mathcal{L}(\hat{\theta}_n, E\hat{V}_n) = \sum_{i=1}^{M}\sum_{t=2}^{T} log\left[\frac{exp[v(x_t^i, d_t^i, \hat{\theta}_n) + \beta E\hat{V}_n(x_t^i, d_t^i)]}{\sum_{d'\in\{0,1\}} exp[v(x_t^i, d', \hat{\theta}_n) + \beta E\hat{V}_n(x_t^i, d')]}\right] + \sum_{i=1}^{M}\sum_{t=2}^{T} log[p_3(x_t^i|x_{t-1}^i, d_{t-1}^i; \hat{\theta}_{n,3})]$$

Based on that fix a new guess $(\hat{\theta}_{n+1}, E\hat{V}_{n+1})$;

end

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 for 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 procedure. 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 significantly faster than the NFXP especially when the discount factor is close to one. Further MPEC 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 but solely contraction 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 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 section 4.3 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. (2016) which looks like the following. 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.

[1] Please refer to Pull Requests #42 and #46 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.

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

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 as well and each data set is estimated five times using the following different starting parameter guesses:

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

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 always 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 offer only NLOPT and IPOPT. IPOPT is used in its default settings with a relative stopping tolerance of 10^{-6}. I pass in 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.

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 previously stated, the NFXP is implemented as a polyalgorithm for the fixed point calculation and the BHHH for likelihood optimization is applied. For the BHHH I choose a relative stopping tolerance of 10^{-8} as well as 10^{-5} as an absolute stopping tolerance. 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 function 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 in Table 1.

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 with rising β the fixed point calculation becomes more complex indicated by the rising number of N-K steps needed to solve it. The implementation of Iskhakov et al. (2016) is more stable in this regard which is likely explained by the fact that they also allow for back switching from N-K to contraction iterations. For MPEC the increased difficulty of solving for the fixed point 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 less complex 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 qualitative differences using another variable of comparison than Su and Judd (2012) as well as Iskhakov et al. (2016). This other dimension is the uncertainty around a quantity of interest which I will introduce in the upcoming section.

3 Uncertainty Quantification

Most sciences work at least to a certain degree quantitatively and by doing so they often rely on theoretical models that in many cases approximate the underlying true mechanisms that drive a certain process or phenomenon. In many cases the quantitative model outcome of interest is the result of a transformation of model inputs through a complex mathematical system. Across disciplines, from a model perspective many inputs and initial states as well as their development are not deterministic but are uncertain. The model inputs are often informed by theory, previous quantitative studies as well as experiments or need to be calibrated using limited data but cannot be treated as known. It is now often argued that

the inherently uncertain parameters propagate their error through the model which yields an uncertain model outcome. In order to make informed decisions based on the model outcome, this uncertainty should be quantified and communicated. This constitutes one main motivation why the applied mathematics field of Uncertainty Quantification (UQ) developed and still is an active field of research. Many of its advancements come from the engineering and physical sciences in which negligence of uncertainty in the outcomes can lead to attributable and severe consequences. In climatology, the predicted path of a hurricane is reported with uncertainties which are due to the dynamical, complex weather conditions that can only be approximated. These uncertainties inform countries about whether they might be hit while giving them valuable time to prepare. An example from engineering involves the safety of a nuclear power plant which in turn is affected by factors such as the weather and its environment in general. This also includes seismological activity which was causing the catastrophe in Fukushima back in 2013. Uncertainty Quantification tries to incorporate those risks in the modeling process and make them explicit (compare Sullivan (2015) and Smith (2013)).

Taking a step back from those very precise motivations for UQ, Thompson and Smith (2019) make a simple mathematical point suggesting that it is worth to investigate uncertainty. Their motivation rests on the butterfly effect (see Lorenz (1963)) which demonstrates that already a slight perturbation in initial conditions can result in an unforeseeable change in the outcome of a complex dynamical system over time. Going back to our initial observation that models often are complex mathematical systems, this provokes the thought that also in this case small changes in inputs might have a strong impact on model predictions. But how does that matter in Economics? And how do we quantify this uncertainty? An answer to the first question and a motivation for UQ can be found in a recent study by Cai and Lontzek (2019). They estimate "the social cost of carbon (SCC), defined as the marginal economic loss caused by an extra metric ton of atmospheric carbon" [3] based on the Dynamic Integrated Climate–Economy" (DICE) model (Nordhaus (2008, 1992)). They add uncertainty regarding the economic development as well as the change in climate conditions to this framework by assuming possible ranges of those factors and propagate them through their model. The result is quite astonishing as for the year 2005 the SCC varies between 59 to 99 Dollars per ton of carbon for a realistic range of projections in economic and climatic outlook. This range of uncertainty in the model outcome is quite large but clearly sends a more informative message to policy makers.

While it is well understood in Economics that there is uncertainty affecting model outcomes and hence policy recommendations as well as its communication are crucial, it is rarely the case that a full, coherent evaluation of uncertainties is performed that goes beyond robustness checks (see Manski (2019) and Scheidegger and Bilionis (2019)). This is where the already existent UQ framework can come in and answer the second question

[3] Compare page 3 in Cai and Lontzek (2019).

from before, how to quantify uncertainty and determine which sources are contributing to it as recently done in Scheidegger and Bilionis (2019) and in Harenberg et al. (2019). In the following, I will introduce the cornerstones of this framework and apply some of its general ideas to the Rust model and the comparison of MPEC and the NFXP.

3.1 The UQ Framework

For the following description I mainly rely on Smith (2013) who outlines a comprehensive UQ framework with a focus on engineering and physics as well as on Oberkampf and Roy (2010) who deal with uncertainty in a more general scientific computing context. The starting point for both is the observation that there are many potential sources of uncertainty that can affect the outcome of a computational model. First, there is model uncertainty which translates into the underlying mathematical framework of a model. In many cases the true underlying process that the mathematical model is supposed to capture is not perfectly understood which renders the mathematical representation imprecise. In the Rust Model we impose the behavioral assumption on Harold Zurcher that his decision process follows that of a maximization of his discounted life time utility over an infinite horizon. We do not know whether this is warranted and even if it was, on a lower level there is still uncertainty about how his underlying cost function exactly looks like. Quite obviously the choice of the specific mathematical representation affects the predictions the model will make after being calibrated. A second source are the parameter inputs themselves. As in Economics they usually are estimated from data, there always remains uncertainty around the specific parameter estimates. This parameter uncertainty translates into uncertainty of the model outcomes. Thirdly, in many applications the computational implementation of a mathematical model involves approximate numerical solutions to the whole model itself or at least parts of it. In our example, we rely on numerical optimization algorithms that introduce some error but also the discretization of the expected value function is solely an approximation of the true mathematical relation. A last factor is a potential measurement error in the model inputs. This could be for instance an imprecise measurement of the cumulative mileage state of some buses.

The authors argue now that the very nature of those uncertainties can be different. On the one hand, the uncertainty can be *aleatoric* which means that it is stochastic and cannot be reduced by gaining additional economic or experimental knowledge. For those uncertainties there typically exists a probability distribution. On the other hand, there is *epistemic* uncertainty which can generally be reduced. It includes for example the previously stated uncertainty of the correct cost function which might be solved by running an experiment. For this example, there clearly is no probability distribution describing the error but a rather an interval of different possible uncertainties. Given that some of these uncertainties are uncovered in a specific model, the UQ framework prescribes the following two steps. However, before we go into this, let us first refine the terms of model and model outcome. In the UQ framework, the computational model can be described like this:

$$y = f(\theta) \tag{16}$$

where θ are the model inputs or parameters and $y \in \mathbb{R}^Q$ the model output or the quantity of interest (QoI).

The QoI can be any policy relevant vector or scalar that could be obtained from the underlying mathematical model. In Cai and Lontzek (2019) this is the social cost of carbon. In my thesis, this will be some counterfactual demand level for bus engines given a certain replacement cost. I will further go into detail about this in the next section. This computational model itself can already suffer from discrepancies to the true mathematical model due to model errors and numerical errors. Given that the model parameters have to be calibrated from data and hence suffer from uncertainty in the estimates, they are represented by a random vector Θ which has a certain probability distribution depending on the data at hand. Given the calibrated parameters, the uncertainty in the parameters affect the model outcome in the following way:

$$Y = f(\Theta) \tag{17}$$

with Y being the QoI that itself is a random vector or variable.

The fact that measurement, numerical and model uncertainty might exist is implicitly included in the fact that the computational model $f(.)$ is not equivalent to the true mathematical model, i.e. the QoI y and the parameter vector θ are already not deterministic but rather uncertain affected by the above mentioned errors.

Coming back to our two steps in the UQ framework, first, the uncertainties are accounted for in the model inputs and then propagated through the model. Second, this propagation is then accounted for in some uncertainty around the quantity of interest. In a sub field of UQ, sensitivity analysis, this propagation technique is exploited to determine which parameters of the model contribute the most to the observed uncertainty in the QoI. This can be of great interest even beyond the argument of informed policy decisions as it can given valuable information to researchers that want to calibrate a certain economic model with actual data. It gives them an indication which parameters to focus on during the calibration procedure (e.g. choose the stopping tolerance of an optimization algorithm such that it gives the most accurate estimate of a certain parameter that drives the uncertainty in QoI while loosening the tolerance for other parameters that are less important) which in turn might reduce the uncertainty in the QoI. This analysis further conveys information about the mathematical model itself an how it could be refined to provoke a lower uncertainty around the QoI (see Scheidegger and Bilionis (2019), Harenberg et al. (2019) and Ghanem et al. (2017)). In this literature, usually a host of combinations of different parametrizations are used to propagate through the model and calculate its QoI. This information from several runs is then used to systematically determine the

influence of the uncertainty in certain parameters on the uncertainty in the QoI. With increasing complexity of the computational model, this becomes computationally infeasible as it would involve too many runs of the model. This is the reason why the literature developed so called surrogate models that approximate the true computational model while maintaining higher speed of convergence for a single run (compare also Saltelli et al. (2008)). While this strand of literature focuses on parameter uncertainty, in the setting of economic models it also implicitly takes some other forms of uncertainty from the calibration procedure into account as the exact estimate of a parameter depends also on model and numerical specifications in the calibration procedure. In many cases, though, a potential model or numerical error will also affect the QoI directly and not solely through the parameter uncertainty as will be the case in what is about to follow.

First, though, in the following section I will introduce the mathematical and computational model for the QoI from which the remainder of my thesis will draw.

3.2 The Quantity of Interest: The Demand Function

Rust (1987) shortly describes a way to uncover an implied demand function of engine replacement from his model and its estimated parameters. Theoretically, for Harold Zurcher the random annual implied demand function takes the following form:

$$\tilde{d}(RC) = \sum_{t=1}^{12} \sum_{m=1}^{M} \tilde{d}_t^m$$

while $\tilde{d}_t^m$ is the replacement decision for a certain bus m in a certain month t derived from the process $\{d_t^m,\ x_t^m\}$. For convenience I will drop the index for the bus in the following. Its probability distribution is therefore the result of the process described by $P(d_t|x_t;\theta)p_3(x_t|x_{t-1}, d_{t-1};\theta_3)$. For simplification Rust actually derives the expected demand function $d(RC) = \mathbb{E}[\tilde{d}(RC)]$. Assuming that π is the long-run stationary distribution of the process $\{d_t,\ x_t\}$ and that the observed initial state $\{x_0,\ d_0\}$ is in the long run equilibrium, π can be described by the following functional equation:

$$\pi(x, d;\theta) = \int_y \int_j P(d|x;\theta)p_3(x|y, j;\theta_3)\pi(dy, dj;\theta). \tag{18}$$

Further assuming that the processes of $\{d_t,\ x_t\}$ are independent across buses the annual expected implied demand function boils down to:

$$d(RC) = 12M \int_0^\infty \pi(dx, 1;\theta). \tag{19}$$

Given some estimated parameters $\hat{\theta}$ from calibrating the Rust Model and parametrically varying RC results in different estimates of $P(d_t|x_t;\theta)p_3(x_t|x_{t-1}, d_{t-1};\theta_3)$ which in turn affects the probability distribution π which changes the implied demand. In the representation above it is clearly assumed that both the mileage state x and the replacement decision d are continuous. The replacement decision is actually discrete, though, and the

mileage state has to be discretized again which in the end results in a sum representation of equation 18 that is taken to calculate the expected annual demand function.

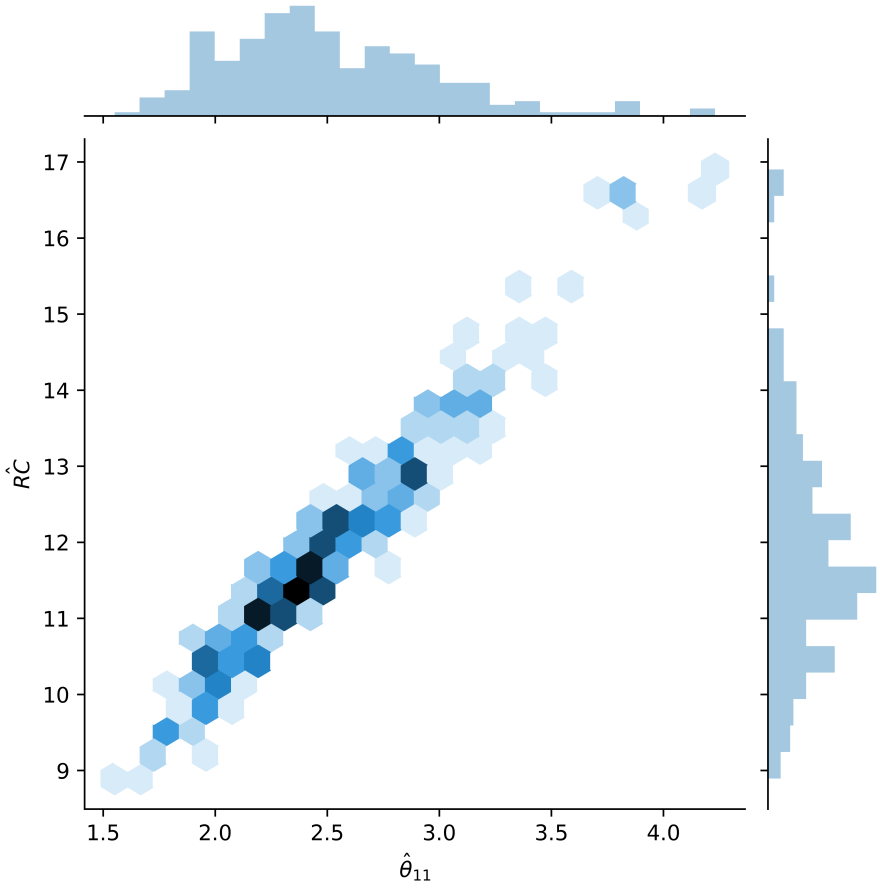
A flexible calculation of the above demand function depending on some parameter vector θ has been implemented by me in the ruspy package which I will rely on in the following. I will now showcase the UQ framework for uncertainty propagation by using the resulting empirical distribution function of the parameter vector $\hat{\theta}$ from the Monte Carlo simulation in 2.4, propagate it through the model (the calculation of the demand function) and obtain the distribution of a QoI (a certain demand level for a given RC) and consequently the uncertainty in its estimate.

3.3 Uncertainty Propagation of the Simulation in Iskhakov et al. (2016)

For this section I draw on the results of the Monte Carlo simulation for the discount factor $\beta = 0.975$ from section 2.4. As the results for MPEC and NFXP are virtually the same and they do not vary across different starting values, I rely on the MPEC results for the starting values $(RC^0, \theta_{11}^0) = (4, 1)$.

Hence, in total I have 250 parameter estimates for 250 different data sets that were,

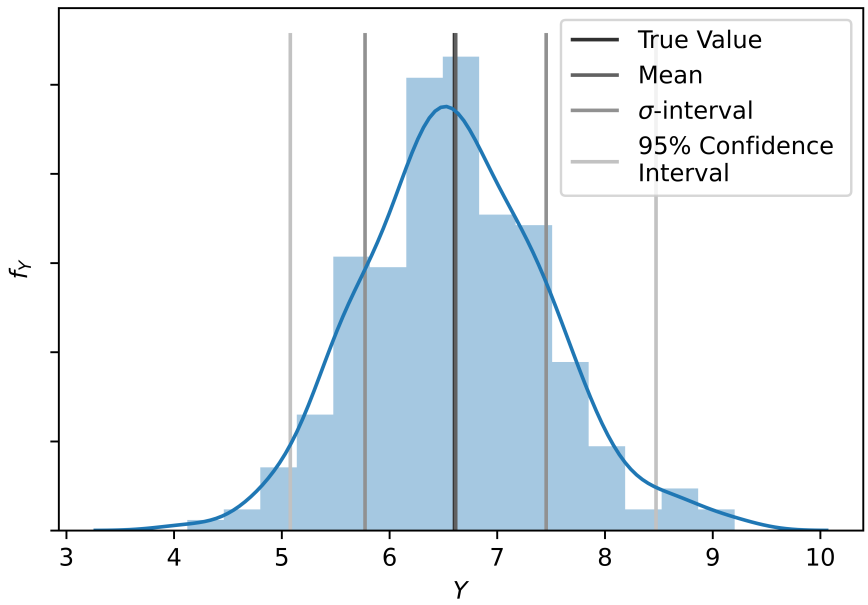
Figure 2. Joint Distribution of $\hat{RC}$ and $\hat{\theta}_{11}$



as Su and Judd (2012) point out, created by a parametric bootstrap procedure. This allows me to identify summary statistics for both the parameters as well as the QoI. As we keep both the model assumptions and numerical approximations constant across the runs, we solely have a look at the effect of uncertainty in the parameters from the calibration process on the QoI. In the context of *sensitivity analysis* which is targeting the influence of parameter uncertainty it is important to first determine whether the uncertainty in parameters is actually independent across them or whether there is some interdependence. The correlation between all parameters is reported in Table **??** in the appendix B. Here, I visualize the joint distribution of the cost parameter estimates in Figure 2 which shows that there is a strong interdependence between them. Consequently, an approach to sensitivity analysis in which only one parameter at a time is varied to determine its contribution to the uncertainty in the QoI would lead to wrong results. In the above Figure 2 we can further observe that the empirical distribution of our estimates is centered around the true values of the cost parameters and that the actual overestimation of them (compare to the appendix 4.3) seems to be driven by some outliers that yield estimates being way too large.

We are now interested in how this paremeter uncertainty translates into uncertainty around some quantity of interest. For the remainder of this paper I am interested in how high expected annual demand for bus engines will be if the equilibrium price for engine replacement drops to 11,000 Dollars (as a reminder the cost function is scaled by 0.001 which means that the true RC corresponds to 11,725.70 Dollars). In the language of the UQ framework the demand function in equation 19 corresponds to the computation model $f(.)$ in equation 16 and 17. The propagation of the random vector $\Theta = (\hat{RC}, \hat{\theta}_{11}, \hat{\theta}_3)_n^{N=250}$ across the Monte Carlo runs results in the random vector Y which equals to the expected

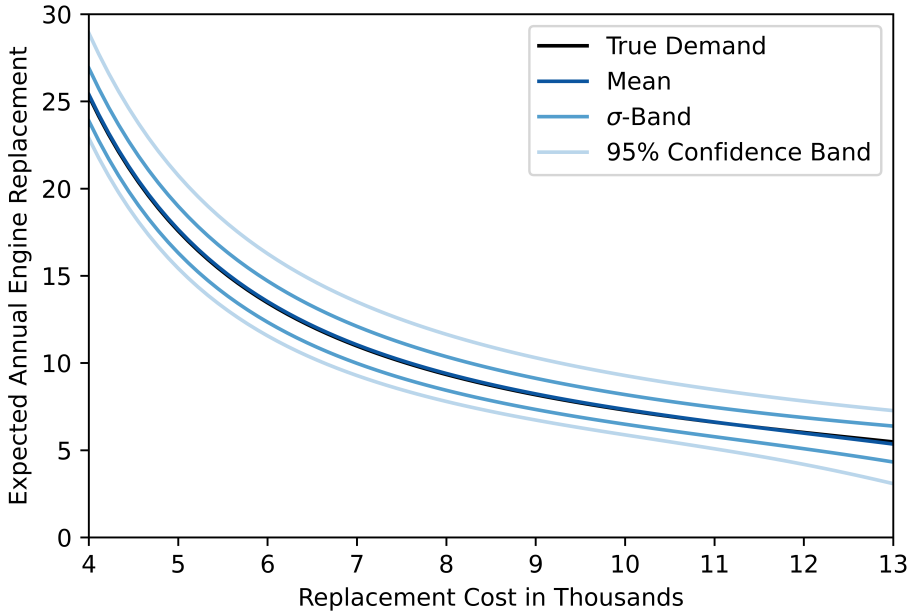
Figure 3. Distribution of the QoI



annual demand at replacement costs of 11,000 Dollars. The uncertainty in the QoI Y is depicted in Figure 3.

The calculated empirical mean of Y across the Monte Carlo runs is very similar to the true expected demand (which is derived from the true parameter vector). The interval of one standard deviation around the mean ranges from around 5.8 to 7.5 while the 95% bootstrap confidence interval stretches from 5.1 to 8.5. This uncertainty obviously also translates into the view on the whole expected demand function across a range of different replacement costs. Figure 4 below corresponds to the implied demand function in Figure 7 of Rust (1987) while being calculated for 50 buses and for the model setup of Iskhakov et al. (2016). To Rust's figure I now add uncertainty resulting from the calibration procedure. I obtain a well-behaved downward sloping demand function for which the mean across our Monte Carlo simulation corresponds closely to the true expected demand function.

Figure 4. The Implied Demand Function accounting for Parameter Uncertainty



This constitutes the end of the description on the basic idea of uncertainty quantification in the Rust Model. In the next sections, I will use this framework to vary other key specifications of the model and the calibration procedure to investigate how this translates into the uncertainty of the QoI described above whether this might differ when using MPEC as opposed to the NFXP.

4 Sensitivity of the Rust Model

As described in section 1, the economic literature exploring the benefits of MPEC and NFXP for structural estimation is limited to studies by Su and Judd (2012), Dubé et al. (2012), Jørgensen (2013), Iskhakov et al. (2016) and Dong et al. (2017). While in these papers different economic models are considered, the set up chosen is quite clean, being

mainly free of any model misspecification and allowing mostly for stable, small numerical errors in the calibration procedure. From a practical perspective and in a research setting where economic models and calibration procedures become more complex, their comparison of MPEC and NFXP is inherently limited to a world in which the modeling and calibration procedure is performed with a minimal error. In a more realistic set up that practioners commonly face where their mathematical model might be more or less off from reality and the computational model has to work with many approximations, there might actually be a larger difference between the two approaches. The previously mentioned studies comparing the two approaches unanimously find that the qualitative results, i.e. the parameter estimates are the same for both and focus on the implementation side covering the speed and rate of convergence. The idea of my following simulation study is to break with the clean comparison of MPEC and NFXP by deliberately introducing some numerical error and model misspecification to test how the two approaches react to it. In this comparison I focus on qualitative aspects investigating how well the two approaches recover the true underlying model. For this a first aspect of the UQ framework comes into play. As I will also change the level of model misspecification in the Rust model a naive comparison of how the structural parameters of the underlying data generating process are recovered, as done by Su and Judd (2012), is flawed. Therefore I rely on the previously used QoI of the counterfactual demand level at replacement cost of 11,000 Dollars. This quantity can be calculated and compared no matter the degree of model misspecification and it is an indicator on how well certain parameters in combination with some model and numerical specifications actually uncovers the predictions of the true underlying model and data generating process. A second aspect from uncertainty quantification entering my following simulation study is the general framework of Oberkampf and Roy (2010). They devise an approach for scientific computing in which they account for model and numerical error as well as parameter uncertainty and propagate them in various combinations through a model. This results in many different distributions of a QoI that in the end is visualized such that it is informative about the uncertainty in the QoI for those that intend to actually use the model outcome. In my simulation study I pick up key ideas of this framework and consequently investigate how changes in model specification and numerical implementation in the calibration procedure translate into the distribution of a QoI and hence its uncertainty. I do so by performing a large scale Monte Carlo simulation for the previously described Rust model. The model is well explored and can therefore be seen as the perfect benchmark for such a study whose results might probably be taken as a lower bound for the effects found of such a simulation in modern and much more complex structural models.

4.1 The Simulation Setup

The setup for my simulation study is inspired by that found in Iskhakov et al. (2016). Consequently, I stay close to the data generating process and ideas explored in section

2.4. First of all, I simulate 250 data sets based on the following true data generating process. The discount factor β equals to 0.975 and the cost function $c(.)$ is linear with a scale parameter of 10^{-3}, i.e. $c(x;\theta_{11}) = 0.001 \times \theta_{11}x$. The true cost parameters are:

$$RC = 11.7257$$
$$\theta_{11} = 2.4569.$$

This parametrization follows exactly the one already encountered before. The first change enters with the true transition probabilities. This change is necessary due to the nature of how the simulation of data is implemented in the ruspy package. The package gives out a data set in which the mileage state of a bus is already discretized according to a previously specified grid size. As a reminder this grid size was set to 175 in the previous setup. In this simulation I intend to investigate how a different discretization of the mileage in the same data set leads to changes in the calibration and later in the QoI. To preserve the information in the data set I have to simulate the data with a large grid size and then derive the same data set with lower grid size from it. I choose to set the grid size to 400 for the initial data generation. The previous parametrization of section 2.4 that is optimized for a grid size of 175 will yield unproportionally small variation in mileage when used for my increased grid size. Accounting for this I adjust the true transition probabilities to be the following:

$$\theta_3 = \begin{pmatrix} 0.04685 & 0.04685 & 0.22375 & 0.22375 & 0.22295 & 0.22295 & 0.00635 & 0.00635 & 0.0001 & 0.0001 \end{pmatrix}.$$

From those 250 data sets with grid size 400 I now derive the same data sets with decreased grid size of 200 and 100. These data sets are calibrated with different specifications which will yield different estimated parametrizations and parameters that might cause different predictions of the QoI. Before I will explain which dimensions of the calibration procedure are tweaked, let us check on how this relates to the UQ framework from before.

As was already described the mathematical model $f(\theta)$ is the demand function in equation 19 which in turn depends on how the Rust model is specified in order to obtain the choice probabilities $P(d|x;\theta)$ and the transition probabilities $p_3(x'|x,d;\theta_3)$ (see equation 18). When taking for granted that the general setup of the Rust model and its assumptions are warranted, there are several ways to easily specify the model differently which would result in changes of the two probabilities. As in the UQ framework, there might be uncertainty about the mathematical model itself which can be captured by varying functional forms of the cost function. Above that, there is uncertainty about how the parameters β as well as θ actually look like. This is further complicated by the fact that the model does not allow us to estimate β. This leaves us quite uninformed about which β and cost function to employ to actually explain some given data. Those circumstances already induced Rust

(1987) to estimate the model several times using different specifications. In this sense my simulation will be a natural extension of his explorations. Apart from the previously explained sources of model and parameter uncertainty, there still remains variation in the model outcome stemming from the numerical implementation of the mathematical model, i.e. from the exact specification of how to solve equation 18 with a computational model. All those sources and their effect I will account for in my simulation to a certain degree. My approach will deviate from the typical workflow in UQ, though, in some ways. Usually in uncertainty quantification (compare Oberkampf and Roy (2010); Smith (2013)) there is already some prior knowledge on how the distribution of the parameters of the model look like. This distribution is then propagated through each combination of plausible mathematical model and computational model specification. This results in a probability distribution of a QoI for each mathematical-computation-model-combination specified. In my setup, it is ad-hoc not clear how a possible distribution of the parameter vector θ might look like. Another factor is that the distribution worth propagating through the model also depends on how the mathematical and computational model are specified as the parameters are calibrated using data and a certain specification that later also affects the specification of the demand function. Another factor that is not linked directly to the demand function but potentially affects the calibration procedure and hence the plausible distribution of θ is the choice of MPEC or NFXP. This remains to be shown, though. Resting on the previous discussion, my approach can be summarized like the following. I combine possible errors in the mathematical and computational representation in the Rust model. These errors directly affect the demand function by itself and further do so indirectly by different calibration results that yield plausible, varying distributions of θ that affect the QoI. I further add another source of parameter uncertainty that stems from the calibration procedure itself and hence only changes the QoI through the variation in parameters. I test two possible numerical errors which are the difference in using either MPEC or NFXP as well as relying on numerical first order derivatives for the likelihood function as opposed to analytical ones. By this I perform an uncertainty quantification of the Rust model in which I incorporate different sources of error and in which the only information on the parametrization of the model comes from the model itself through its calibration. The goal of the approach is twofold. It shall uncover whether NFXP and MPEC might yield different qualitative results when facing unfavourable conditions while at the same time perform a broader check of the model implications when facing reasonable discrepancies in the mathematical and numerical model specification. Hence, it is supposed to be a comprehensive approach that gives as much valuable information to policy makers as possible.

I will now explain the exact steps taken to arrive at the results presented below. I run a Monte Carlos simulation with 250 runs each per specification. As a benchmark case I estimate the parameters when relying on the true specification of the model. This means that I run the estimation on the 250 data sets with gird size 400, linear cost function,

a β of 0.975 as well as analytical first order derivatives of the likelihood function. This first specification I call the "correctly specified" in the following. This and all the other specifications are hence estimated on 250 data sets and this once per approach MPEC and NFXP. This simulation is equivalent to a parametric bootstrap, as noted by Su and Judd (2012), which allows me to obtain sample statistics for the parameters. While the 250 runs do not allow me to obtain exactly unbiased estimates of the true parameters, I still do not opt for more runs as the sheer amount of different specifications I consider would make it too computationally expensive given the limited time during my thesis. Further it makes it more comparable to the set up in Iskhakov et al. (2016) and Su and Judd (2012) as they also solely simulated 250 data sets per specification. Due to estimating the model twice (once with MPEC and once with NFXP) per specification 500 runs are performed. So which specifications do I use? The specifications differ in the degree of model misspecification (or flexibility) and numerical error. The model uncertainty is introduced by allowing more flexibility in the cost function than the true data generating process actually displays. The true cost function is linear but I also allow the optimizers to employ more flexible cost function. Specifically, quadratic cost $c(x; \theta_{11}, \theta_{12}) = \theta_{11}x + \theta_{12}x^2$ or cubic cost $c(x; \theta_{11}, \theta_{12}, \theta_{13}) = \theta_{11}x + \theta_{12}x^2 + \theta_{13}x^3$ are possible. Additionally, I add the dimension of a wrongly specified β. In my setup it is therefore possible that the estimation is based on $\beta = 0.985$ although it is actually equal to 0.975.

A numerical error is introduced in two ways. While the true data generating process conveys all its information on the mileage state into a grid of 400 points, I deliberately estimate the model parameters with some loss of this information. As already mentioned, this is accounted for by running the calibration after discretizing the mileage state further from 400 to either 200 or 100 grid points. Lastly, I give the option of the realistic feature that the economic model and the estimation procedure give rise to a likelihood function that has to rely on numerical first order derivatives as opposed to analytical ones. This is the only part of the specifications that affects the QoI solely through the calibration of the structural parameters.

Clearly, it might be possible that several of those misspecifications occur at the same time and that the uncertainty introduced in the QoI might depend on this. For that reason, I build the Cartesian product of all those single misspecifications to estimate each of its possible combinations which leaves me with 36 specifications for which I separately estimate the parameters for 250 data sets using the NFXP and MPEC, respectively. A list of all specifications can be found in Table 4 in the appendix B. All this amounts to 18,000 runs for which the model is calibrated and the resulting parameters are taken to calculate the expected implied demand for bus engines for the counterfactual level of replacement cost at 11,000 Dollars. Selected results of this procedure are presented in the following section.

4.2 The Results

In the first step, we look at some partial sensitivity explorations to get a sense of how the QoI reacts to small changes in the model setup. This means that at first I only look at those specifications in which I deviate from the correctly specified model in only one dimension. This is I am either tweaking the model dimension, i.e. just the cost function and/ or β or I change the numerical dimension consisting of the grid size and the use of numerical gradients. Before I do so let us fill up on one missing ingredient of the setup discussion from before. The setup for MPEC is exactly chosen as it was described in section 2.4. Again I use IPOPT as a solver with the tolerances specified as before. My initial intention to keep the setup as close to the one in Iskhakov et al. (2016) as possible had to be eased a bit for the NFXP, though. This is due to malperformance of the initial setup that I employ to replicate the results in Iskhakov et al. (2016) when facing more flexible cost functions. This will be the starting point of the following section.

4.2.1 The Partial Perspective

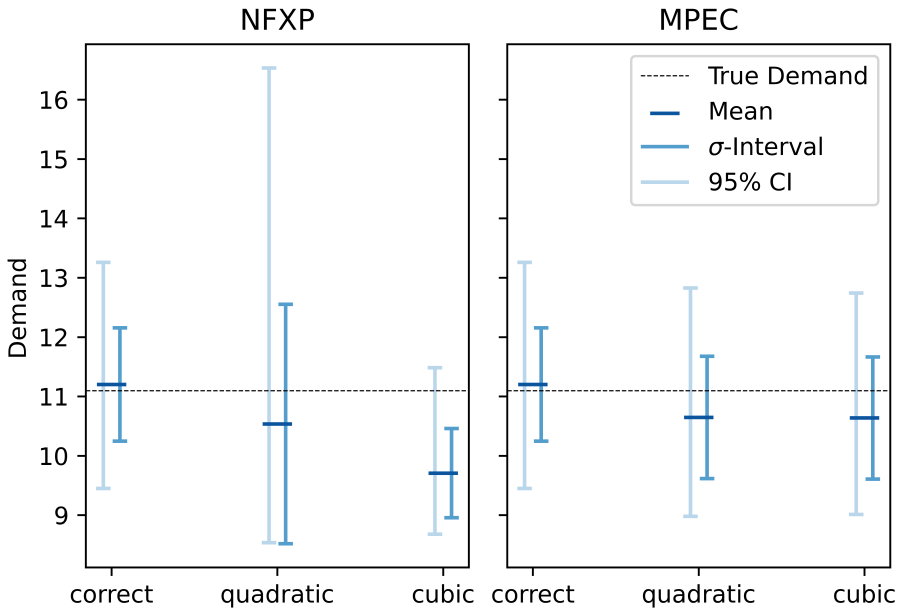
The first results are those coming from the specifications in which I only add more flexibility to the cost functions than are used to calibrate the Rust model. Apart from that the model stays correctly specified. In Figure 5, I present some summary statistics for the QoI depending on which specification is used for NFXP and MPEC separately. This same kind of figure with its labeling will follow us along for several explorations which is why the legend in the subsequent figures will be dropped. Figure 5 shows the correctly specified model at the very left as a benchmark case for both NFXP and MPEC. For this and the other specifications the mean of the QoI across the 250 runs, as well as its 95% confidence and standard deviation interval are reported.[4] As already noted before the mean of the QoI for the correctly specified model does not exactly match the true value of the QoI. This is the case as there are not sufficiently many Monte Carlo runs such that the mean of the structural parameters are estimated with some small sample bias which translates into the QoI. As a reminder the true QoI is derived from the true underlying model and its true structural parameters. In our setting the true QoI is the following:

$$QoI_{true} = 11.095.$$

When using the correctly specified model, as expected, NFXP and MPEC yield virtually the same results for all three key statistics. The mean comes in at 11.203 while the one standard deviation interval spreads from 10.248 to 12.157 with the confidence interval being equal to $(9.449, 13.261)$. In the nest specification to the right the correct model is maintained with the exception that the calibration procedure now assumes the cost function to be more flexible by adding a possible quadratic term. This reveals an interesting

[4]For the bootstrap confidence interval the percentile bootstrap is used (compare Davison and Hinkley (1997) chapter 5).

Figure 5. The QoI when varying the Cost Function

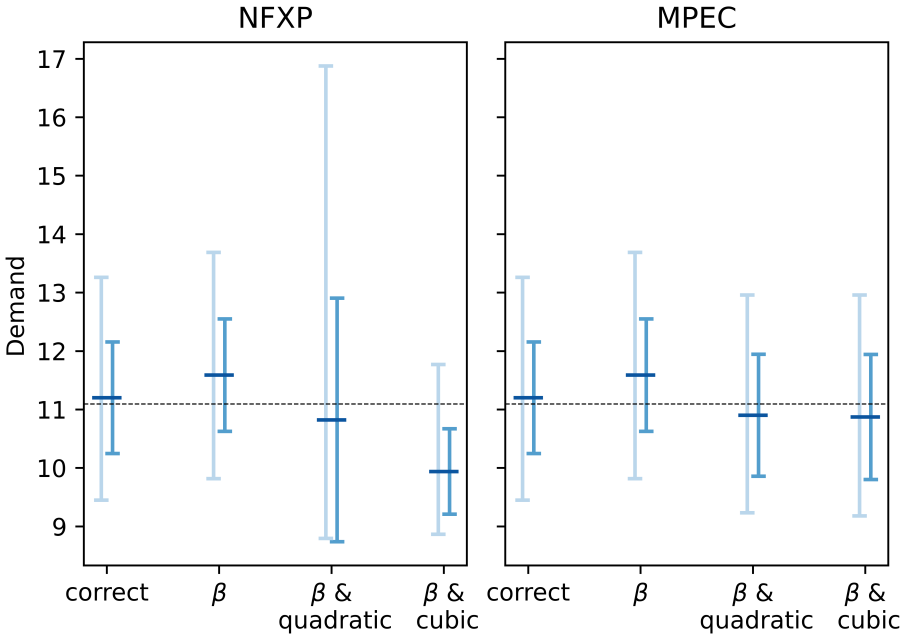


difference between the NFXP and MPEC which I tried to eliminate but could not entirely. Firstly, the means for the QoI for both approaches now underestimate the true one while the NFXP does so more strongly than MPEC. NFXP yields 10.537 while MPEC estimates 10.646. This difference stems from different estimates for all three structural cost parameters. Both approaches have a convergence rate of 100 percent but when comparing the estimates per run one can observe that the two approaches systematically find different solutions in every run. The NFXP implementation that yields these results is actually obtained by using the scipy implementation of the L-BFGS-B (see Zhu et al. (1997) and Virtanen et al. (2020)) with its default tolerances. Apart from that the setup of the NFXP algorithm used here is the same as in section 2.4. Solely I replace the BHHH by the above mentioned solver.[5] The figure further reveals that the variation in estimates for the QoI is a lot larger when relying on the NFXP as opposed to MPEC. This pattern seems to flip around, though, when assuming the cost function to have a cubic form. The underestimation of the QoI in the case of NFXP becomes even more pronounced, while all key statistics stay roughly constant when applying MPEC.

This qualitative difference between the two approaches is surprising as they theoretically both solve the same problem and find the same results as proven by Su and Judd (2012). In practice, though, the calibration procedure in the Rust model cannot be solved analytically

[5]In an initial try I ran the whole simulation with the BHHH with the tolerances exactly set to the ones from section 2.4. This produced the results in Figure 12 which are very far off. I first tried to allow for a more precise fixed point calculation by setting the maximum number of contraction and N-K iterations each from 20 to 40. This only altered the results marginally. In a second step I tried some combinations of relative and absolute tolerances: Specifically, from $(10^{-3}, 10^{-6})$ to $(10^{-13}, 10^{-16})$ in steps of increasing exponent of two. Again, it did not change the results so I switched to the L-BFGS-B. When using that one as well, the results do not change substantially when varying the tolerances which is why in the end I simply adapt the default settings for the optimizer and the tolerances from section 2.4 for the fixed point calculation. I experimented with tolerance ranging from 10^{-3} to 10^{-21}.

Figure 6. The QoI when varying β and the Cost Function



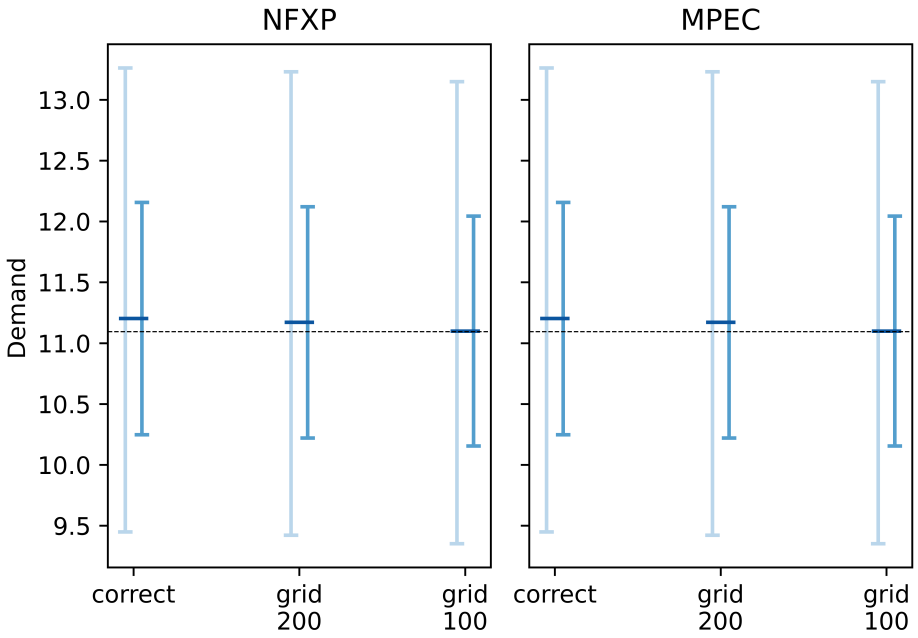
and hence different problem formulations can react differently on numerical solving. A further complicating factor is posed by the use of different optimizers for the two approaches. Internally, their functioning clearly differs as for instance both rely in my case on a numerical approximation of the Hessian matrix which already differs analytically due to the competing problem formulations but will even differ more due to contrasting approximation techniques. It is entirely possible that despite my extensive experimentation with tolerances, the NFXP can be set up such that it obtains the same results as MPEC in my specifications.

At the same time, though, as the two approaches are promoted as competitors as opposed to complements, many practioners most likely rely on only either of the two approaches which will possibly leave them with the single results of MPEC or NFXP as I found them. Clearly, looking separately at them, they both do not appear implausible. Yet, having estimated the other approach as well would convey additional information about the quality of the other.

From an uncertainty perspective it is interesting to note that giving more flexibility in the cost function to the estimation procedure does not cause the solver to simply estimate linear cost but rather exploit the two additional terms in the cost function. This results in a downward bias for the QoI irrespective of using the NFXP or MPEC. Looking at MPEC alone, it seemingly shifts the distribution of the QoI.

In Figure 6, the second part of the model dimension is tweaked. In a first step, the solver is given the information that the true β is assumed to be equal to 0.985. This leads for both MPEC and NFXP to an overestimation of the QoI in the exact same way. The mean increases now to 11.588. In the next step, I incorporate the first interaction of misspecifications. To the wrongly specified β more flexibility in the cost function is added

Figure 7. The QoI when varying the Grid Size

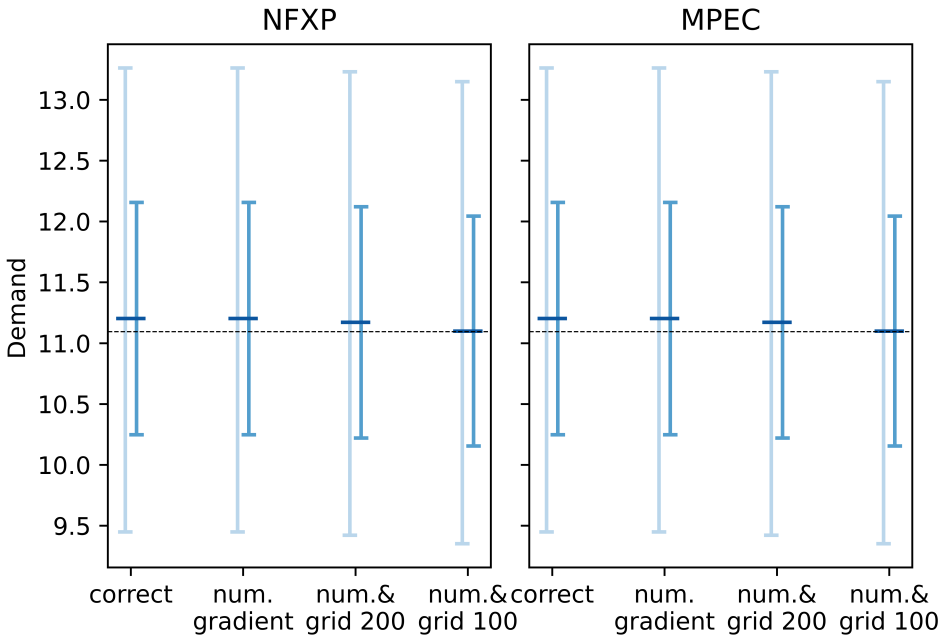


to cover all possible cases of model misspecification in my setup. As already observed in the singular view of just a change in the cost function, the increased flexibility causes the mean estimates to be shifted downwards again. This change happens in a very similar magnitude as it did in Figure 5 with β being equal to 0.975. Interestingly, the predictions when employing misspecification in β and more flexibility than needed in the cost function are improved in comparison to misspecification in only one of those two dimensions. At least when only facing model error, in my example, an increased error yields better results as the two errors balance each other out. It is further worth mentioning that in all of the so far considered specifications, every run converges irrespective of NFXP or MPEC.

After this singular view on the model dimension, we will repeat this for the numerical dimension. In Figure 7 I split the grid size from the case that holds all information (400 grid points) in half twice, ending up with a grid size of 200 first and 100 second. This loss of information introduces a small downward bias in comparison to the correct specification. With my low number of Monte Carlo runs, this actually makes the prediction better when looking at the true QoI. Although it cannot be said with certainty from my setup but it is likely that with increasing runs this downward bias will persist ending up in estimates for lower grid sizes that will be below the true parameter. With certainty it can only be said that as the mean calculated in the correct specification is a consistent estimate of the true QoI, lowering the grid size is not generally beneficial.

In Figure 8, we conclude the partial view on the numerical dimension by adding numerical gradients as opposed to analytical ones. The rationale of adding this dimension is to observe how MPEC and NFXP react to the assumption that the likelihood function is not differentiable. Both Su and Judd (2012) as well as Iskhakov et al. (2016) utilize

Figure 8. The QoI varying the Grid Size and the Gradients



the build-in tool of automatic differentiation in the AMPL modeling language to obtain analytical derivatives of the likelihood function for MPEC. I abstract from this by supposing that there is no analytical derivative and hence it has to be approximated numerically. For this I rely on two-point finite differences. As can be seen in the figure, in the qualitative results of both MPEC and NFXP this numerical error does not make any difference in the estimates of the QoI irrespective of solely having this error or whether it is paired with decreasing grid size. This means that the estimates are the same for the displayed specifications regardless of whether numerical or analytical derivatives are applied.

When it comes to the quantitative comparison as done in Iskhakov et al. (2016) there is a notable difference, though. Relying on finite differences comes with more function evaluations needed per major iteration of the solver. Per derivative evaluation of the solver the two-point finite difference approach makes two function evaluations per parameter. As there is a difference between MPEC and NFXP in the amount of derivative evaluations needed and of parameters, this translates into a different number of total function evaluations. When using IPOPT for MPEC it is necessary to supply the augmented log likelihood with its gradient as well as the constraints with its Jacobian. Assuming the grid size to be 400 and the cost function to be linear, one calculation of the numerical derivative of the likelihood function comes with 402×2 likelihood function evaluations as well as 400×2 constraint evaluations. In the case of the NFXP this looks less daunting. As there are only two parameters and solely the gradient of the likelihood function is required, one gradient evaluation results in two additional function evaluations. A single function evaluation is more costly, though, in the case of the NFXP as each time the economic model has to be solved. In the Rust model this means that with high precision a fixed point has to be calculated, while in the case of MPEC the function

is simply evaluated at a given point. This tradeoff determines whether MPEC or the NFXP is more efficient when using numerical derivatives. Clearly, in the Rust model it also depends on the grid size as it affects the dimension of the fixed point for the NFXP and the number of parameters for MPEC. This tradeoff clearly has an effect on which models might be more efficiently estimated with one or the other approach which is what I examine in section 4.6 in appendix D for the class of Eckstein-Keane-Wolpin models.

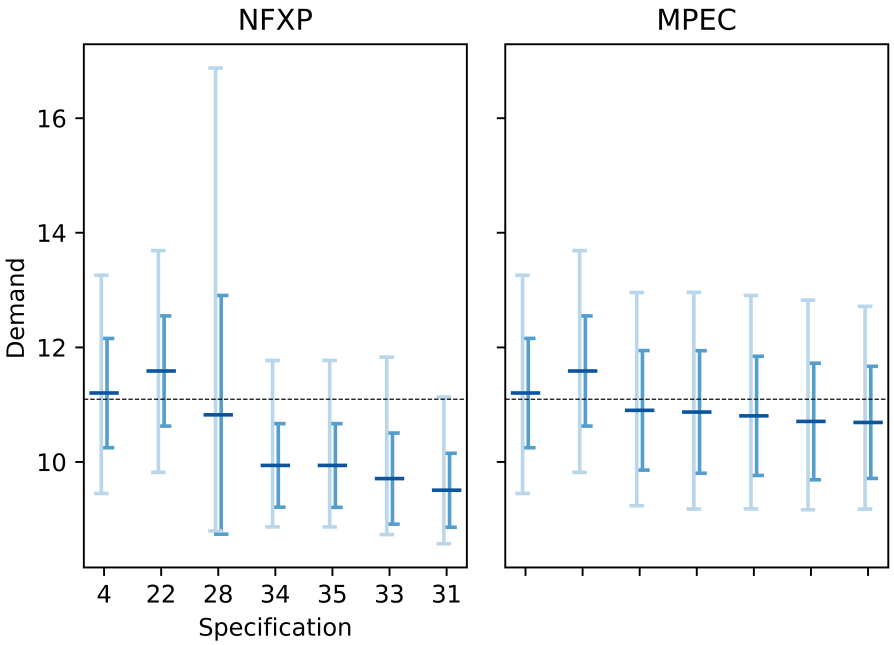
As can be seen in Table 4 in the appendix, when comparing the correctly specified to the same model but with numerical derivatives, for MPEC the number of likelihood function evaluations increases from on average roughly 24 to about 8541 (not taking into account the rising number of constraint evaluations). The amount of time needed per run on average is around seven times as high. For the NFXP the likelihood function comes from 13 on average and increases to approximately 39.[6] At the same time, this affects the number of contraction and N-K iterations. They rise on average from 285 to 1080 and from 77 to 287. The CPU time needed for the NFXP with numerical derivative is only twice as high as is the case with analytical gradient. Clearly, this tradeoff between MPEC and NFXP depends heavily on the efficient solving of the economic model and its complexity as well as the capability of solving the problem with a low number of gradient evaluations. When comparing how those statistics change from using numerical derivatives with grid size 400 to lower grid size, it becomes apparent that for MPEC this makes a large difference. The number of likelihood function evaluations roughly halves when dividing the grid size by two. The CPU time needed is every time roughly divided by seven. For the NFXP the number of contraction and N-K steps stays roughly constant while the CPU time still is divided by three to four each time. All this shows that when using finite-difference numerical derivatives the NFXP seems to be superior in my setting in regard of the quantitative efficiency due to the lower dimensionality of the problem it solves.

From an uncertainty perspective, in this partial view the use of numerical derivatives does not have an effect on the QoI. However, the numerical error introduced by lowering the grid size does shift some key characteristics of its distribution downwards.

In Figure 9, which concludes this section, I introduce a possible sequence of different specifications coming from the correctly specified model on the left to a model in which every possible dimension is wrongly specified on the right. As a matter of fact, the first four specifications are those from Figure 6 in which I first increase β and then additionally allow for more flexibility in the cost function. From four to five I add numerical gradients which in the following specifications is combined with decreasing grid size. In the last specification we hence end up with the most extreme case in which the cost function is assumed to be cubic, β equals to 0.985, numerical derivatives are used and the grid size

[6] The 39 function evaluations are solely an approximation as estimagic does not allow to give out the total number of function evaluations including those needed for the numerical derivatives. As the L-BFGS-B roughly needs as many gradient evaluations as function evaluations, the total number of function evaluations is approximated by $13 + 13 \times 2$.

Figure 9. The QoI for a Sequence of Specifications



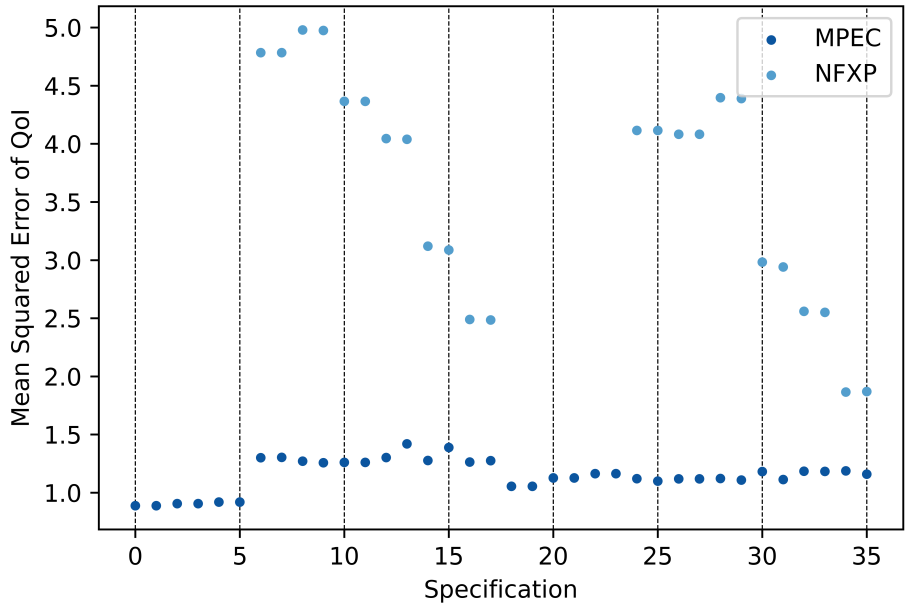
amounts to 100. When facing already some misspecification, the switch from analytical derivatives to numerical ones has a small effect for MPEC this time.

The mean shifts from 10.873 to 10.804 as well as the other statistics move downwards. As the qualitative results are only calculated for runs that successfully converged, this can be explained by a lower convergence rate of MPEC which is around 80 percent. From that on both approaches incur an increasing downward bias with decreasing grid size. Again the rate of convergence of MPEC is ranging between 66 and 83 percent. The distribution of MPEC might shift a bit in the sense that less probability mass is above twelve and more at the lower end of the estimates. For the NFXP it is interesting to see that not only the mean but also the confidence and standard deviation intervals react to the change in grid size. Those observations suggest that there might be more subtle changes in the uncertainty of the QoI when looking at its whole distribution. This will be done in the next section where I bring in a comparison of all possible 36 specifications.

4.2.2 The Full Perspective

In a real world setting when having only one data set at hand from which we do not know the data generating process, we lack the information oo how our model estimation actually performs. The true QoI is unknown as is the true mathematical model. The computational model is surely an imperfect approximation of the mathematical one and there is no direct guidance on which specification of the demand model is actually the most accurate. In this setting, the more important it becomes to obtain as much information as possible on how sensitive the model reacts to certain misspecifications in which ever dimension of error. This is where uncertainty quantification is positioned and where the previously mentioned sub field *sensitivity analysis* operates in regard of parameter uncertainty. As

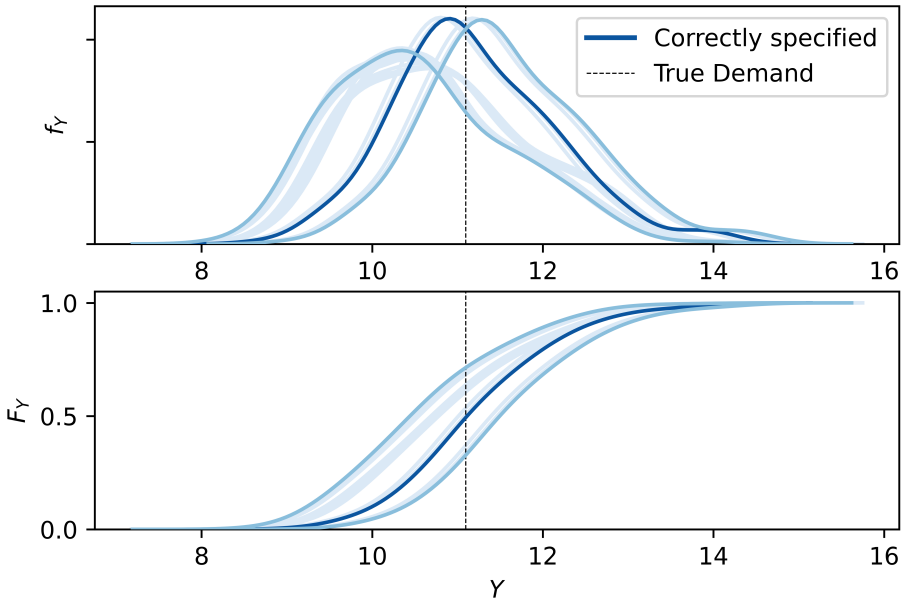
Figure 10. The MSE of the QoI across all Specifications



Oberkampf and Roy (2010) argue, as much information as possible should be incorporated into the process of obtaining the QoI by propagating this information through the model and making it available to policy makers.

In my simulation setup I am able to actually make a qualitative comparison of different specifications and take a stance on how sensitive the model prediction replies to a certain setting. In Figure 10, I now draw from a common measure for predictive performance which is the mean squared error (MSE) that I calculate for the estimated QoI per specification and approach. This yields a readable overview of which specifications and consequently which error the model is more or less reactive to. We can see that the model responds strongly do whether NFXP or MPEC is used to obtain the parameter distribution that is propagated through the model. Further the choice of the cost function has a visible effect, too, increasing the MSE with more flexibility in the cost function in comparison to linear cost. The reaction to the grid size appears to be case specific while the tradeoff between numerical and analytical gradient is only slightly observable. It is more pronounced, though, for MPEC. The increase in β leads to less prediction precision with linear costs but the converse with other cost functions. At this point, it should be mentioned that in a few cases I observe the same phenomenon as Dong et al. (2017) did considering the NFXP. As explained before already, they find that the NFXP in some cases visits structural parameter guesses that cause the model solving procedure to fail. This is due to its separation of the estimation procedure into two loops. In my simulation it happens as well four times that the N-K step cannot be calculated causing the NFXP to not converge while MPEC does. While this can be healed through back switching to contraction steps in the NFXP, it is still worth mentioning that I can generally confirm their finding in my setting.

Figure 11. The PDF and CDF of the QoI using MPEC



I conclude this section with Figure 11 which loosely refers to the visualizations in Oberkampf and Roy (2010). For the case of having used MPEC across all specifications, I plot the resulting distributions of the QoI for each specification. In the first window the probability density function (PDF) and in the second the cumulative distribution function (CDF). What all have in common is that they are bell shaped while being more or less left-skewed. The correctly specified model has its peak slightly to the left of the true parameter while the mean is actually slightly above it which is due to some strong overestimations of the QoI. I further marked in slightly lighter blue the two distributions that frame the others to visualize in which range the distributions are. The left frame corresponds to the specification in which I combine the true β with cubic cost, a grid size of 100 and numerical derivatives. The right frame is represented by the true specification with the only difference that β is equal to 0.985. All other specifications are depicted by the often overlapping light blue/gray curves. It can be seen that the resulting distributions can be roughly split up into three groups. The group on the very far right consists of all specifications in which the linear cost function is combined with an increased β. This is irrespective of grid size and choice of derivative and seems to shift the distribution of the correctly specified model to the right. The second group which is around the correctly specified model is made of those specifications that rely on a linear cost function paired with the true β. The rather large group of distributions on the left are now consisting of all those specifications with either quadratic or cubic costs irrespective of grid size and/ or numerical derivatives. This group is slightly split into two as there are two bigger bundles with a small gap in between. The right hand side again are those specifications that use the increased β of 0.985. So, in general, the increased β shifts the existing distribution to the right. Generally, the far left distributions that are defined by a more flexible cost

function actually change the distribution in the QoI to a more left-skewed one as compared to the correctly specified one. In the window below the previously explained patterns are equally visible with the probability distribution function.

The whole figure conveys three key messages. First, by propagating a reasonable distribution of parameter estimates through the demand function model we can uncover the probabilistic uncertainty around the QoI. Second, we can detect across a specific range which model and numerical errors in the calibration procedure and the modeling of the demand function translate into different estimated parameters which in turn result in a range of possible distributions of the QoI. Third, this range can be now used to uncover which errors affect the uncertainty in the QoI in which way. This range serves as additional information for policy makers that accounts for more than just simple variation in the parameter estimates stemming from estimation with sample data based on a single specification of the mathematical and computational model.

For an entirely comprehensive representation of my simulation, one would complement the figure by the same using the NFXP. This is Figure 13 which can be found in the appendix. As can already be guessed, the resulting distributions when using the NFXP look very different across specifications.

4.2.3 Discussion

Let us put ourselves in the position of having performed my previous study with one real data set from which we bootstrapped 249 other data sets. Assume that we would have, hence, obtained my results using real data and without knowledge about the data generating process and therefore the true value of the QoI. If we had only estimated the data sets with the NFXP and flexible cost functions (the two blocks of distributions that are left-skewed at the very far left of Figure 13), this would have resulted in an entirely different distribution of the QoI than with other cost functions. A policy decision solely based on this would be very much flawed. If the whole Figure 13 was given the way it is now, the policy decision would be far more difficult to make as the range of possible distributions is large. But at the same time, this uncertainty can be incorporated and accounted for in policy which makes it more informed and robust. Having the knowledge about results from MPEC would even further enhance possible policy. As has been established in my thesis, the qualitative results of MPEC and NFXP can systematically differ under certain circumstances (even after extensive robustness checks regarding solver, tolerances and fixed point setup in the NFXP). Providing the uncertainty propagation results for both approaches instead of only one of them surely conveys more comprehensive information. Seeing the two approaches as complements rather than competition for informed policy making should be considered as it is already viewed when comparing different optimizers within the same approach. In a recent working paper, Komiyama and Shimao (2018) follow this view by suggesting a model selection procedure in which they specifically allow their algorithm to choose between MPEC and the NFXP as the preferred approach.

While the uncertainty quantification as done in my thesis is computationally expensive and might not be feasible for every model, it highlights that even in a comparably simple structural model, relatively small numerical and model errors (especially combined) can have an noteworthy impact on model predictions. This indicates that for more complex models an extensive uncertainty quantification should be aimed for when certain model predictions are supposed to inform policy. As highlighted in the literature on sensitivity analysis, uncertainty quantification can not only serve for informed policies but also give information on how to best calibrate a model. Additionally, knowing about how sensitive a certain model reacts to certain errors that might likely occur in practice might help making it more robust. In regard of the complexity of a model and its feasibility for uncertainty quantification, the use of surrogate models is already widely established which can mitigate this concern. Harenberg et al. (2019) show that structured global sensitivity analysis can be performed quite easily and without large adaption on economic models. This gives scope to a lot of further research in which those techniques are further developed and applied to existing economic models. Potentially, as changes in the computational and mathematical model affect the outcome of global sensitivity analysis, a combination of different specifications paired with running a global sensitivity analysis might be beneficial.

All those approaches, can potentially be beneficial for model improvement, policy making but also for model calibration. It thus can support empiricists to later decide whether performing model averaging or model selection could improve the model predictions.

References

Aguirregabiria, V. and P. Mira (2002). Swapping the nested fixed point algorithm: A class of estimators for discrete markov decision models. *Econometrica 70*(4), 1519–1543.

Aguirregabiria, V. and P. Mira (2010). Dynamic discrete choice structural models: A survey. *Journal of Econometrics 156*(1), 38 – 67. Structural Models of Optimization Behavior in Labor, Aging, and Health.

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.

Cai, Y. and T. S. Lontzek (2019). The social cost of carbon with economic and climate risks. *Journal of Political Economy 127*(6), 2684–2734.

Davison, A. C. and D. V. Hinkley (1997). *Bootstrap Methods and their Application*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press.

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.

Gabler, J. and T. Raabe (2020). respy - a framework for the simulation and estimation of eckstein-keane-wolpin models.

Ghanem, R., D. Higdon, and H. Owhadi (2017). *Handbook of uncertainty quantification*, Volume 6. Springer.

Harenberg, D., S. Marelli, B. Sudret, and V. Winschel (2019). Uncertainty quantification and global sensitivity analysis for economic models. *Quantitative Economics 10*(1), 1–41.

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.

Keane, M. P. and K. I. Wolpin (1994). The solution and estimation of discrete choice dynamic programming models by simulation and interpolation: Monte carlo evidence. *The Review of Economics and Statistics 76*(4), 648–672.

Keane, M. P. and K. I. Wolpin (1997). The career decisions of young men. *Journal of Political Economy 105*(3), 473–522.

Komiyama, J. and H. Shimao (2018). Cross validation based model selection via generalized method of moments.

Lorenz, E. N. (1963). Deterministic nonperiodic flow. *Journal of the atmospheric sciences 20*(2), 130–141.

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

Manski, C. F. (2019). Communicating uncertainty in policy analysis. *Proceedings of the National Academy of Sciences 116*(16), 7634–7641.

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

Nordhaus, W. (2008). *A Question of Balance: Weighing the Options on Global Warming Policies*. Yale University Press.

Nordhaus, W. D. (1992). An optimal transition path for controlling greenhouse gases. *Science 258*(5086), 1315–1319.

Oberkampf, W. and C. Roy (2010). *Verification and Validation in Scientific Computing*. Cambridge University Press.

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.

Saltelli, A., M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, and S. Tarantola (2008). *Global sensitivity analysis: the primer*. John Wiley & Sons.

Scheidegger, S. and I. Bilionis (2019). Machine learning for high-dimensional dynamic stochastic economies. *Journal of Computational Science 33*, 68–82.

Smith, R. C. (2013). *Uncertainty quantification: theory, implementation, and applications*, Volume 12. Siam.

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

Sullivan, T. J. (2015). *Introduction to uncertainty quantification*, Volume 63. Springer.

Thompson, E. L. and L. A. Smith (2019). *Economics: The Open-Access, Open-Assessment E-Journal 40*, 1–15.

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.

Zhu, C., R. H. Byrd, P. Lu, and J. Nocedal (1997, December). Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw. 23*(4), 550–560.

4.3 Appendix A: Tables

Table 2. Correlation between Parameter Estimates

	$\hat{RC}$	$\hat{\theta}_{11}$	$\hat{\theta}_{30}$	$\hat{\theta}_{31}$	$\hat{\theta}_{32}$	$\hat{\theta}_{33}$
$\hat{RC}$	1.0	0.97	-0.04	-0.01	0.00	0.15
$\hat{\theta}_{11}$	0.97	1.0	-0.02	-0.03	0.01	0.14
$\hat{\theta}_{30}$	-0.04	-0.02	1.0	-0.26	-0.31	-0.07
$\hat{\theta}_{31}$	-0.01	-0.03	-0.26	1.0	-0.82	-0.06
$\hat{\theta}_{32}$	0.00	0.01	-0.31	-0.82	1.0	-0.12
$\hat{\theta}_{33}$	0.15	0.14	-0.07	-0.06	-0.12	1.0

Table 3. The Specifications for the Sensitivity Simulation

Specification	β	Cost Function	Grid Size	Analytical Gradient
0	0.975	linear	100	Yes
1	0.975	linear	100	No
2	0.975	linear	200	Yes
3	0.975	linear	200	No
4	0.975	linear	400	Yes
5	0.975	linear	400	No
6	0.975	quadratic	100	Yes
7	0.975	quadratic	100	No
8	0.975	quadratic	200	Yes
9	0.975	quadratic	200	No
10	0.975	quadratic	400	Yes
11	0.975	quadratic	400	No
12	0.975	cubic	100	Yes
13	0.975	cubic	100	No
14	0.975	cubic	200	Yes
15	0.975	cubic	200	No
16	0.975	cubic	400	Yes
17	0.975	cubic	400	No
18	0.985	linear	100	Yes
19	0.985	linear	100	No
20	0.985	linear	200	Yes
21	0.985	linear	200	No
22	0.985	linear	400	Yes
23	0.985	linear	400	No
24	0.985	quadratic	100	Yes
25	0.985	quadratic	100	No
26	0.985	quadratic	200	Yes
27	0.985	quadratic	200	No
28	0.985	quadratic	400	Yes
29	0.985	quadratic	400	No
30	0.985	cubic	100	Yes
31	0.985	cubic	100	No
32	0.985	cubic	200	Yes
33	0.985	cubic	200	No
34	0.985	cubic	400	Yes
35	0.985	cubic	400	No

Table 4. Mean Quantitative Comparison for Specifications of Figure 8

Specification	Time (in Sec.)	# of Major Iter.	# of Func. Eval.	# of Func. Eval. (total)	# of Bellm. Iter.	# of N.K. Iter.
4 & NFXP	7.38	11.73	13.25	NaN	285.04	76.98
4 & MPEC	23.13	18.68	23.75	23.75	NaN	NaN
5 & NFXP	16.43	11.73	13.25	NaN	1080.16	286.84
5 & MPEC	168.39	19.13	24.63	8540.83	NaN	NaN
3 & NFXP	3.72	14.02	15.02	NaN	1221.28	322.91
3 & MPEC	26.37	22.1	28.01	4919.5	NaN	NaN
1 & NFXP	1.19	12.35	13.82	NaN	1125.92	282.24
1 & MPEC	4.44	21.56	31.5	2457.77	NaN	NaN

4.4 Appendix B: Figures

Figure 12. The QoI when varying the Cost Function (BHHH)

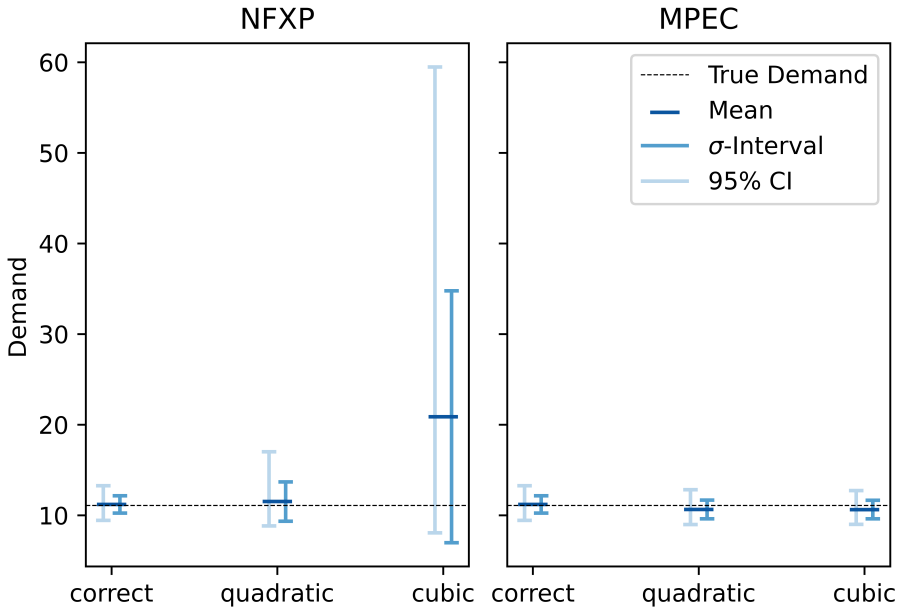
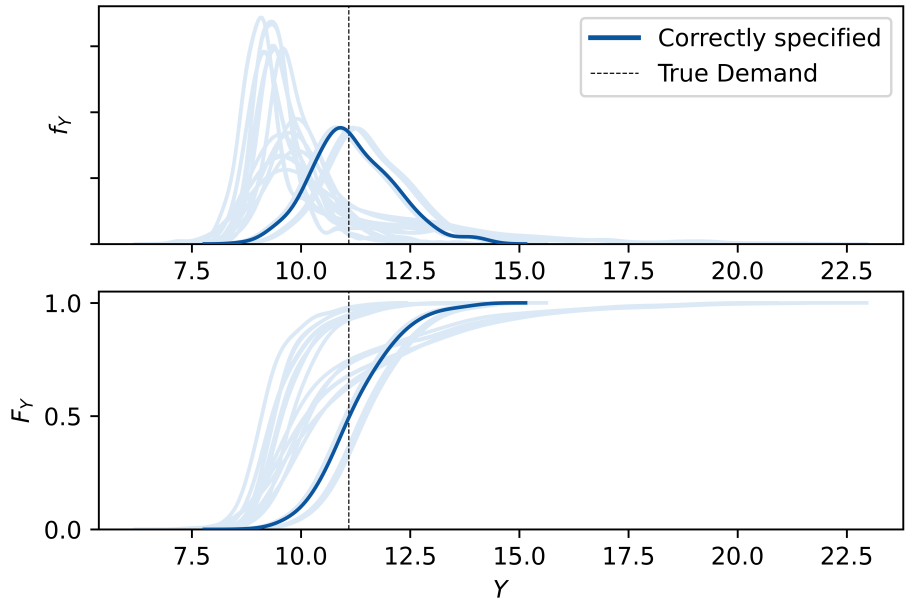


Figure 13. The PDF and CDF of the QoI with the NFXP



4.5 Appendix C: Further Details on the Replication of Iskhakov et al. (2016)

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, I rely entirely on open-source programs which has some implications for my 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 their results 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[7] that provide analytical derivatives via automatic differentiation for code written in Python, 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 scope for this thesis. I decided therefore to at least code up the first order derivative by hand and pass it on to IPOPT. The fact that in my case 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 to exactly replicate the setup for KNITRO with IPOPT. For the NFXP there are also some differences. 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 might 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 as solver seems to be inferior to my implementation when looking at the number of iterations and function evaluations needed.[8]. 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

[7] See `https://github.com/google/jax`.

[8] 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. Although this should generally only cause solver to converge more often due to increased numerically stability but does not affect the parameter estimates and should not change the speed too much.

Table 5. 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)

actually constitutes a parametric bootstrap procedure.

The two statistics are provided for both my implementations and additionally for the NFXP of Iskhakov et al. (2016). These results were not published but had to be obtained by me using their matlab replication code. 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.

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.

4.6 Appendix D: MPEC for respy

In a preliminary stage of my thesis, I explored whether it could be beneficial to use MPEC instead of the NFXP for a python package called ruspy (see Gabler and Raabe (2020)) developed at the University of Bonn. This package offers users to flexibly estimate models from the class of Eckstein-Keane-Wolpin (EKW) models as defined in Aguirregabiria and Mira (2010). While I initially set up a notebook with a simulation based on the one of Iskhakov et al. (2016) in order to explore the applicability and usefulness of MPEC for respy[9], similar arguments can be made with my major simulation in this thesis.

While the EKW models also belong to the class of single agent dynamic discrete choice models, they differ to the Rust model by relaxing some of its key assumptions. To give a few examples, first of all, they allow agents to have several choices that are mutually exclusive. Further, Aguirregabiria and Mira (2010) highlight that, as for instance in Keane and Wolpin (1994), there is permanent unobserved heterogeneity of individuals and unobservables can be correlated across choice alternatives. The assumption that the unobservables are not necessarily extreme value distributed further relaxes Rust's assumptions. This provokes the first complexity in solving the model to obtain conditional choice probabilities with which the likelihood function can be obtained. The economic model must be solved for multiple types (due to heterogeneity) each time and solving the model involves numerically solving multi-dimension integrals (as no closed form solution exists due to the flexibility in distribution of the unobservables). This is needed to obtain an integrated value function coined *EMAX* by Keane and Wolpin (1994) for the calculation of the conditional choice probabilities. As the model has a finite horizon, it also does not involve a fixed point calculation but rather can be solved using backward induction for the integrated value functions. While it should generally possible to cast the backward induction with its integrated value function calculation, it is reasonable to first explore whether an MPEC formulation of this problem in general might prove beneficial in this setting. Establishing the problem as a MPEC would involve having a new parameter

[9]See `https://github.com/Pascalheid/MPEC_for_respy/blob/master/mpec_fo_respy.ipynb`.

per integrated value function. This results in an immense amount of parameters and constraints. To see this, let us have a look at the context of the EKW models. Those are set in Labor Economics and at each point in time an agent has the option to work in specific profession or stay at home. Those are the several possible discrete choices. The state space consists at each point in time of a history of experience in a certain job or schooling and the choice of given that history. In order to solve the problem by backward induction, only obtaining the last periods choice specific value functions, it is necessary to calculate the integrated value function at each possible state space (job and schooling history) that might occur in the last period (compare Aguirregabiria and Mira (2010); Keane and Wolpin (1994, 1997)). Depending on the amount of alternative choices and time periods, the state space is extremely large. in any case, it will be way beyond the 175 constraints and 177 parameters modeled in Su and Judd (2012) and Iskhakov et al. (2016).

Another factor is that the criterion function that has to be optimized in the respy package is not differentiable everywhere which likely stays like that when switching to MPEC. This gives rise to another complication that the above mentioned authors did not have to face. I have to rely on finite-difference numerical derivatives while the authors in the Rust setting could even rely on the desirable feature of analytical first and second order derivatives. Additionally, the sparsity patterns of those are supplied to the solver. All this together makes their setting quite favorable in the sense that the implementation is as efficient as it could be yielding the very high speed and convergence. While Su and Judd (2012) report to have successfully run MPEC in this setting of the Rust model with 100,052 parameters and 100,042 constraints within an hour on 12 GB RAM work station (which is the range of dimensions, one would face in reasonable implementations of EKW models), this will likely not be possible in the less favorable setting of respy.

There are several possible reasons to arrive at this opinion. Those are already laid to a certain extent in section 4.2.1 (it makes sense to read that section before the following) and can be observed in Table 4. The increased dimensionality of the MPEC formulation in comparison to the NFXP in combination with finite-difference numerical derivatives results in a surge of function evaluations. While those function evaluations are far less costly than those of the NFXP in the Rust model, this is less clearly so in respy. In the Rust model the tradeoff stems from the speed and robustness with which the fixed point calculation is implemented. As the model in respy is solved via backward induction, the efficiency gain that might arise from casting it into the constraints as opposed to coding it up yourself is less clear. Su and Judd (2012) report that the main advantage of MPEC is that it does not enforce the fixed point to hold exactly until the last iteration of the calibration process. Aguirregabiria and Mira (2002) argue that this is computationally more efficient as it is not necessary to enforce it precisely in each iteration. From this it is also not directly clear whether this also holds true for backward induction of Bellman equations, weakening another key argument for MPEC. Another observation that can be made in Table 4 is that the fact of using numerical derivatives paired with increasing

number of constraints and parameters (here represented by increasing grid size) seems to affect the performance of MPEC more heavily than it does the NFXP. The critical factor here is the previously mentioned tradeoff between the massive amount of function evaluations for MPEC and the increased complexity of solving the fixed point for the NFXP. Seeing that already in this setting with only a low dimensionality (in comparison to the one expected in respy) the CPU time and major iterations for MPEC climb up more strongly than for NFXP does not seem to make it beneficial to further seriously test the usefulness of MPEC for respy. Although initially considered to implement MPEC in repsy as a topic for my thesis, it was not deemed to be beneficial to pursue for the above outlined reasons.

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