Machine Learning for Dynamic Incentive Problems

Philipp Renner	Simon Scheidegger[*]
Department of Economics	Department of Economics
University of Lancaster	University of Lausanne
United Kingdom	Switzerland
p.renner@lancaster.ac.uk	simon.scheidegger@unil.ch

November 13, 2022

Abstract

We introduce a flexible and scalable method for solving discrete-time dynamic incentive problems with heterogeneous agents and persistent types. Our framework entails a generic and numerically tractable model reformulation that can be solved with standard dynamic programming techniques. We propose to embed the recast problem into a parallelized value function iteration algorithm, where high-dimensional and nonlinear functions are efficiently approximated using Gaussian process regression in combination with Bayesian active learning. This combination enables us to address the previously intractable question to what extent risk heterogeneity causes adverse selection in insurance markets, where the theoretical and empirical literature are at odds. We posit and solve a dynamic adverse selection model with heterogeneous agents and persistent types. Unlike the classical view, we find that the presence of multiple agents with different risks has only a minimal effect on the payoff to the principal, thereby explaining some discrepancies between theory and empirics.

JEL classification: C61, C73, D82, D86, E61.

Keywords: Dynamic Contracts, Principal-Agent Model, Heterogeneous Agents, Dynamic Programming, Machine Learning, Gaussian Processes, Reinforcement Learning.

[*]We thank Marlon Azinovic, Ben Brooks, Lorenzo Bretscher, Johannes Brumm, Hui Chen, Antoine Didisheim, Rick Evans, Min Fang, Jesús Fernández-Villaverde, Luca Gaegauf, Ken Judd, Narayana Kocherlakota, Felix Kubler, Luca Mazzone, John Rust, Karl Schmedders, Tony Smith, Chris Sleet, Aleh Tsyvinski, Sevin Yeltekin, and seminar participants at Yale University, the University of Lausanne, the University of Zurich, EPFL Lausanne, ETH Zurich, MIT Sloan School of Management, Imperial College, CEF 2018 in Milan and LGTC 2018, and the NBER Conference on Big Data and High-Performance Computing for Financial Economics 2019 for their valuable comments. This work was generously supported by grants from the Swiss National Supercomputing Centre (CSCS) under project IDs s885, s995, the Swiss Platform for Advanced Scientific Computing (PASC) under project ID “Computing equilibria in heterogeneous agent macro models on contemporary HPC platforms”, the Swiss National Science Foundation (SNF), under project IDs “Can Economic Policy Mitigate Climate-Change?”, “New methods for asset pricing with frictions”, and the Enterprise for Society (E4S). The work was also supported by the High End Computing facility at Lancaster University. Simon Scheidegger gratefully acknowledges support from the MIT Sloan School of Management and the Cowles Foundation at Yale University.

1 Introduction

Dynamic incentive problems are of key importance in model-based economics. They occur whenever two parties form a contract with asymmetric information and encompass applications such as manager remuneration, optimal taxation, and insurance contracts.[2] One major cause that drives this asymmetry is heterogeneity, as preferences and risks are generally only known to one contract party. The seminal work by Akerlof [1970], for instance, predicts that heterogeneity in risk will adversely impact insurance markets. The author argues that since insurance companies cannot screen for risks, higher risk types increase insurance premia, thereby driving individuals with lower risk out of the market. This finding is often used as a primary argument for governments to intervene in insurance markets. In contrast, the empirical literature has failed to confirm model-implied adverse selection due to risk in prominent settings such as health insurance markets (see, e.g., Cardon and Hendel [2001] or Spinnewijn [2017] and references therein).

One possible explanation for the discrepancy between theory and observation is that combining key economic ingredients such as the persistence of hidden information, heterogeneity across market participants, and dynamics in a single model render it intractable for conventional solution methods and therefore, these have been avoided from the outset.[3] In consequence, the existing literature has relied on simplifying assumptions that could create a significant wedge between models and their objectives. To take but one example, insurance contracts are typically dynamic and not static, that is, the insurer receives a steady flow of information about the insured's risk type. Furthermore, the type itself, such as an individual's health condition, is often persistent,[4] and heterogeneity across market participants should also be taken into account (see, e.g., Spinnewijn [2017]). Therefore, to determine how, and to what extent, risk heterogeneity causes adverse selection, one needs to study dynamic problems with multiple risk types, persistent hidden information, and subsequently compare the payoffs in the former setting with those of a single risk type; a model setting for which neither analytical nor numerical solution methods are generally available today.

In this article, we make progress on these issues by proposing a generic and scalable computational framework based on machine learning that will enable researchers to solve a wide variety of dynamic incentive problems that were previously considered to be out of reach. We then use this framework to study a discrete-time dynamic adverse selection model with persistent shocks and heterogeneous agents; a problem that, to the best of our knowledge, has not been studied before despite its immanent relevance. We find

[2] See, e.g., Golosov et al. [2016] and references therein for a thorough review.

[3] Sandroni and Squintani [2007] for example study a static adverse selection model with heterogeneous agents in the context of insurance, whereas Cicala et al. [2022] consider adverse selection as a policy instrument in a climate-change question, thereby assuming that the contract is exogenous. Other work limits the heterogeneity of the agents by restricting the number of hidden states to two [Mathevet et al., 2022] or solely considers one dimension of heterogeneity by differentiating with respect to ability [Stantcheva, 2017].

[4] Previous research suggests that private information in the economic environments we are interested in is highly persistent [see, e.g., Meghir and Pistaferri, 2004, Storesletten et al., 2004].

that considering multiple agents with a hidden risk profile in a model only has a small effect on the overall value of the contract, implying that, at least in the model we posit, heterogeneity in risks fails to explain adverse selection, thereby confirming the findings of the empirical literature [Cardon and Hendel, 2001] in a theoretical setting.

Two major bottlenecks create substantial difficulties in solving dynamic adverse selection problems with persistent hidden information and heterogeneous agents if existing methods are applied. The first has to deal with the fact that models with repeated agency require full history dependence [see, e.g., Lambert, 1983, Rogerson, 1985]. This, in turn, leads to time-inconsistent dynamic programs. A way of formally dealing with this issue is to introduce *promised utilities* as artificial state variables [Fernandes and Phelan, 2000]. This, however, leads to a severe complication in practical applications: the feasible set becomes endogenous to the problem, that is, it has to be computed as well. Abreu et al. [1986, 1990] provide a constructive proof for the existence of the set of continuation payoffs. However, it is, in general, unclear how to numerically represent this possibly multi-dimensional and non-convex set explicitly. The second bottleneck has to deal with solving these models in "reasonable" time as, for every individual type of agent added to the model, the dimensionality of the (irregularly shaped, that is, non-hypercubic) feasible set increases. Thus, if standard Cartesian grid-based value function iteration algorithms are applied in the solution process, the computational effort and storage requirements grow exponentially and render even models of only moderate complexity computationally intractable; an effect that is termed the *curse of dimensionality* [Bellman, 1961].[5] Due to these complications, the existing literature, therefore, is typically limited to two-dimensional models [see, e.g., Broer et al., 2017, Doepke and Townsend, 2006, Abraham and Pavoni, 2008], where the curse of dimensionality is avoided from the outset.

In this article, we try to remedy these shortfalls. Specifically, our contribution is fourfold. First, we introduce a reformulation of dynamic incentive problems that is numerically easier to handle than the standard recursive formulation commonly used in the literature (see, e.g., Fernandes and Phelan [2000], Golosov et al. [2016]). Concretely, we propose to combine ideas from penalization methods that emerged in the constrained optimization literature [Luenberger and Ye, 2008, Ch. 13] to relax the recursive formulation of the models, thereby bypassing the difficult numerical task of performing set-valued dynamic programming [Abreu et al., 1986, 1990] to characterize the feasible set. Our relaxation is computationally tractable, as it can now be solved with standard value function iteration. To show the validity of our approach, we provide a formal proof that it provides a solution

[5]There are two key computational challenges when a model is solved by iterating on a Bellman equation. First, in each iteration step, value and policy functions need to be approximated. For this purpose, the function values have to be determined at many points in the high-dimensional state space. Second, at each point, one has to solve a high-dimensional maximization problem. These two important features of the considered problems create difficulties in achieving a fast time-to-solution process. To see now the intuition for the curse of dimensionality, consider the task of approximating a univariate value function by visiting 10 locations along the input state. Generalizing to d states, this procedure requires visiting $O(10^d)$ locations in the d-dimensional state space and evaluating the function at all these locations. Even in a situation where a single function evaluation is relatively inexpensive to compute, naively attempting to approximate a high-dimensional function in this way can quickly become infeasible.

to the original problem.

Second, and to the best of our knowledge, we are the first in quantitative economics to solve recursively formulated dynamic incentive problems by applying a dynamic programming algorithm that uses Gaussian process regression (GPR) [see, e.g., Rasmussen and Williams, 2005] in conjunction with Bayesian active learning (BAL) [see, e.g., Deisenroth et al., 2009]—methods from the supervised and reinforcement machine learning literature[6]—to efficiently approximate high-dimensional value and policy functions within the value function iteration. BAL is shown to be a crucial ingredient for the overall efficiency of our algorithm, as it helps to alleviate the curse of dimensionality by focusing the Gaussian process (GP) approximation on the equilibrium path, which is a portion of the state space that is typically much smaller than the entire feasible set (see, e.g., Sannikov [2022], and references therein for a general discussion). Thus, given a fixed computational budget, BAL steers the solution process so that the quality of the approximated function is highest where it matters the most, along simulated paths within the feasible set. In addition, to reduce the time-to-solution potentially by orders of magnitude, we propose and implement a parallelization scheme for dynamic incentive problems that allows the efficient use of contemporary high-performance computing hardware. This combination allows us to solve challenging problems with many state variables in a relatively short amount of time and to tackle problems that have, thus far, been insurmountable.

Third, we apply our computational framework to the dynamic adverse selection model by Fernandes and Phelan [2000] as a verification test for our method as, for their two-dimensional baseline setting, the approximate numerical solutions are known and the results can be compared. Furthermore, since the models we study no longer have analytical solutions, but have to be determined iteratively, the final ingredients for our framework are measures to assess the credibility and correctness of our computational results. To do so, we follow the best practices in a sub-field of computational sciences called *validate, verify, and uncertainty quantification* (VVUQ; see, e.g., Oberkampf and Roy [2010], and references therein), and propose to use two particular error criteria jointly.

Fourth, to study the central question of this article, that is, how heterogeneity in risk across agents affects adverse selection in insurance markets, we posit and solve a stylized dynamic adverse selection model with heterogeneous agents that builds on Fernandes and Phelan [2000].[7] We find that considering multiple risk types in a dynamic adverse selection model only minimally impacts the overall value of the contract. Thus, the reduction in profitability for the principal is low. Consequently, she still would offer a contract to the agents, even if there is more than one risk type present.

[6]Appendix A provides a short glossary of terms that we use in this paper and that are common in the machine learning literature. In addition, we try to relate the machine learning terminology to the terms commonly used in economics.

[7]Note that the method proposed in this paper has a far broader scope: it could as well be applied, for example, to moral hazard problems or dynamic games, where one of the major difficulties also lies in finding the equilibrium sets [see, e.g., Wang, 1995, Judd et al., 2003]. For discrete actions and lotteries over payoffs, the correspondences are convex valued. However, for other assumptions, they are not, which demands a more general approach such as that proposed in this paper.

This finding implies that risk heterogeneity cannot explain adverse selection, at least within the model setup under consideration, in conformity with the empirical literature. Finally, code examples that illustrate our methodology are provided at https://github.com/GaussianProcessesForDynamicEcon/DynamicIncentiveProblems.

The remainder of this article is organized as follows. In Sec. 2, we review the related literature. Next, we outline in Sec. 3 the dynamic adverse selection model with heterogeneous agents and history-dependent shocks we intend to study. Section 4 introduces our generic solution framework and discusses its computational advantages relative to other existing methods. To study how heterogeneity in risk across agents affects adverse selection in insurance markets, we discuss in Sec. 5 the solutions to a heterogeneous agent model with multiple types. Finally, section 6 concludes.

2 Related Literature

Previous research extensively studied various approaches to make dynamic incentive models formally tractable.[8] However, the numerical treatment of the said models has often been tailored to specific instances, and consequently, many problem formulations remain intractable. Being numerically substantially restricted is an unfortunate situation since additional state variables are frequently required to address the questions of interest. Kocherlakota [2005], when solving an optimal taxation model, states that *"It would be desirable to use an infinite horizon example as in Albanesi and Sleet [2006]. However, to answer the questions of interest, the example would have to include aggregate shocks, persistent hidden state variables, and probably should allow for endogenous physical state variables. At a conceptual level, it is known how to attack problems of this kind, thanks to the work of Fernandes and Phelan [2000] and Doepke and Townsend [2006]. Practically, it is still impossible to implement their procedures in an example that includes the elements of interest."* With the work presented in this paper, we intend to complement the formal approaches by providing a generic and scalable computational framework that also renders them computationally operational.

The seminal work by Abreu et al. [1986, 1990] (henceforth APS) introduced a constructive procedure for computing feasible sets. In particular, the authors showed that there exists a monotone set-valued operator whose fixed point is the feasible set, similar to the Bellman operator in dynamic programming. In practical applications, one has to repeatedly approximate non-convex equilibrium correspondences with some numerical technique before the model at hand can be solved recursively. Judd et al. [2003] and Yeltekin et al. [2017], for example, provide a numerical scheme for determining the feasible sets of discrete state supergames by using polygons. Their approach, however, relies on the convexification of the payoff set and suffers from the curse of dimensionality. Sleet

[8] For an incomplete list of research in discrete-time settings, see, e.g., Spear and Srivastava [1987], Fernandes and Phelan [2000], Cole and Kocherlakota [2001], Werning [2002], Doepke and Townsend [2006], Abraham and Pavoni [2008], Mele [2014], Pavoni et al. [2017], Kapička [2013], Pavan et al. [2014], and for models in continuous time, see, e.g., DeMarzo and Sannikov [2006], Sannikov [2008], Williams [2009, 2011], He et al. [2017].

and Yeltekin [2016] provide an extension to this method for the case of continuous state variables, but their extension has the same issues. Abreu and Sannikov [2014] provide methods for computing feasible sets in discrete state, two-player games that are restricted to convex sets and also suffer from the curse of dimensionality. Abreu et al. [2020] extend the former work by exploiting the difference between bounding and slack incentive constraints to speed up the convergence of the algorithm. Unfortunately, their approach relies on polytopes and thus will only function in higher dimensions for specific cases. In contrast, the approach we propose has several desirable features that lift the aforementioned issues. First, our reformulation of dynamic incentive problems completely bypasses the need to pre-determine the feasible set by introducing slack variables invoked in the regions of the computational domain where the problem otherwise would be infeasible, turning the task of solving it into one of dealing with an ordinary dynamic programming problem [Stokey et al., 1989a]. Second, our computational framework alleviates the curse of dimensionality and, therefore, can deal with problems involving many types as well as other additional state variables if needed. Third, it can handle problems with both convex and non-convex feasible sets. Moreover, the approximate feasible set can still be determined if needed via information that is implicitly available. For instance, the slack variables are approximated over the entire computational domain with GPR. If the latter are non-zero, we know that a location in the state space can be deemed infeasible.

Several authors have proposed alternatives to Fernandes and Phelan [2000] for formally dealing with dynamic incentive problems. Marcet and Marimon [2019], for example, look at a planner's problem with forward-looking constraints. In particular, they introduce Lagrange multipliers as state variables, which then leads to a recursive saddle point problem. As a result, they can avoid the procedure of finding a feasible set. Pavoni et al. [2017] recently extended the approach by Marcet and Marimon [2019] towards more general constraints. Their state space is the positive orthant. Mele [2014] extends the Lagrange multiplier approach to also include hidden actions. However, for all the recursive formulations by Marcet and Marimon [2019], Pavoni et al. [2017], and Mele [2014], the state space is non-compact. Thus, it is impossible to apply standard numerical dynamic programming techniques [see, e.g., Judd, 1998] because the latter require a compact domain [Stokey et al., 1989a]. Kapička [2013] and Pavan et al. [2014] take a different route by looking at a continuum of types. They use a first-order approach to consider a relaxed problem, that is, a simplified version of the optimization problem, where one has to track utility promise and marginal utility alongside the physical state variables. They both impose strong assumptions on the distribution of the hidden information to ensure that their approach is potentially valid and also requires ex-post verification of any results. As in the case of finitely many types, one does not know what promises are feasible a priori, and thus it is necessary to first determine the feasible set. To make the first-order approach numerical tractable, Kapička [2013] uses additional assumptions to reduce the model to a two-dimensional version, whereas Pavan et al. [2014] consider the problem from a purely analytical perspective. However, the first-order approach is unsuitable for dealing with multi-dimensional heterogeneity due to its stringent assumptions on the distribution.

Concretely, applying this method permits considering, for instance, continuous shocks to income, but not additionally including different risk types because of the said restrictions. In contrast, in a setting with discrete states, enriching a model with more heterogeneity is straightforward as one simply can add more types, which in turn increases the dimensionality of the problem. Furthermore, Battaglini and Lamba [2019] showed that the first-order approach is generically invalid, meaning that for most assumptions on preferences and distributions, the first-order approach fails. Independently of these issues, models with one single continuous type still have an irregularly-shaped state space of at least three dimensions and, therefore, can benefit from our complementary computational approach if reformulated accordingly.

Once a dynamic incentive problem is reformulated as proposed in this article, one has to compute global solutions[9] to the recursively formulated dynamic adverse selection model. For this purpose, we introduce a parallelized discrete-time dynamic programming algorithm that uses GPR to approximate the value and policy functions, and that distributes the workload by using distributed memory parallelism [see, e.g., Skjellum et al., 1999]. Moreover, to focus the observations where needed most, we augment the set of data over which GPs are fitted by applying BAL [see, e.g., Krause and Guestrin, 2007]. This measure substantially increases the efficiency of approximating functions using GPR with high precision with as few observations as possible. GPR is a form of supervised machine learning that has successfully been utilized in various applications in data science, engineering, and other fields to perform approximation and classification tasks. Early work consists of Engel et al. [2003], who apply GPs in the context of reinforcement learning to study a two-dimensional maze in order. Deisenroth et al. [2009] use GPs and BAL to approximate value and policy functions in finite-horizon control problems in low dimensions to study pendulum swing. Berkenkamp et al. [2016] use GPs in the context of safe controller optimization for quadrotors. Busoniu et al. [2010] provide a textbook treatment of reinforcement learning with kernel-based function approximators. In economics, Scheidegger and Bilionis [2019] applied GPs to solve dynamic stochastic growth models and to perform uncertainty quantification, whereas Kotlikoff et al. [2021] embed parts of the methodology presented in this article into a time iteration algorithm [Judd, 1998] to study overlapping generation models in the context of climate change. We complement and extend this prior work in economics by proposing to combine GPs with BAL to mitigate the curse of dimensionality efficiently and to solve dynamic adverse selection models with a substantial degree of heterogeneity. A defining feature of GPs is that they *learn*, that is, approximate a function in a non-parametric fashion based on the observations available at so-called design points, and do so without any geometric restriction. Thus, GPs stand in stark contrast to ordinary, grid-based approximation schemes for high-dimensional state

[9]Below, we follow Brumm and Scheidegger [2017] and use the term "global solution" for a solution that is computed using equilibrium conditions at many points in the state space of a dynamic model—in contrast to a "local solution", which rests on a local approximation around a steady state of the model as, for example, perturbation methods do. For a method that computes such a global solution, we use the term "global solution method". This use of the word "global" is not to be confused with its use in the phrase "global optimization method".

spaces such as Smolyak's method [see, e.g., Krueger and Kubler, 2004b, Judd et al., 2014, Fernández-Villaverde et al., 2015], adaptive sparse grids (see, e.g., Brumm and Scheidegger [2017], Brumm et al. [2022], and references therein), high-dimensional model representation [see, e.g., Eftekhari and Scheidegger, 2022], or projection methods [see, e.g., Judd, 1992]. These grid-based approximators are restricted to hyper-rectangular state spaces and thus are not a natural modeling choice in the context of solving dynamic adverse selection models.

Empirical research (see, e.g., Cardon and Hendel [2001]) so far has failed to confirm adverse selection due to risk in the insurance industry as theoretically predicted by Akerlof [1970]. Thus, researchers have looked for various other sources of heterogeneity such as differences in preferences [Cohen and Einav, 2007] or demand frictions [Spinnewijn, 2017] that could drive adverse selection. There is growing evidence that multiple dimensions of heterogeneity are relevant to studying the insurance market (see Finkelstein and McGarry [2006], Cohen and Einav [2007], and Fang et al. [2008]). In contrast, models that are typically used in empirical research avoid considering endogenous contracts from the outset, and those studies that do have no hidden information in a dynamic context. Azevedo and Gottlieb [2017] for instance point to this limitation by stating that *"However, they restrict consumers to be heterogeneous along a single dimension, despite evidence on the importance of multiple dimensions of private information"*. Our proposed method can be viewed as a potential tool to bridge this gap. Since our framework is capable of dealing with model settings richer than previously considered in the literature, it can help researchers to remove the "straight jacket" imposed by the the present computational restrictions.

3 Dynamic Incentive Problems with Heterogeneous Agents

This section posits an adverse selection model with heterogeneous agents and persistent types. To do so, we first introduce in Sec. 3.1 an infinitely repeated, dynamic adverse selection problem in its most general form. This benchmark model is relatively easy to explain. In addition, it can be scaled up in a straightforward and meaningful way to a setting with heterogeneous agents. Second, we generalize this benchmark in Sec. 3.2 to a stylized heterogeneous agents model, which is sufficiently rich to study the extent to which hidden information causes adverse selection in a dynamic context.

3.1 A Baseline Dynamic Incentive Model

Throughout this paper we consider infinite-horizon, discrete-time economies, where t represents the current period.[10] It consists of a risk-averse agent with unobserved, persistent taste shocks and a risk-neutral planner who provides optimal, incentive-compatible insurance against taste shocks. The agent reports his type to the principal who then trans-

[10]We follow the concise notation by Golosov et al. [2016] in describing the general setting of Fernandes and Phelan [2000].

fers consumption to the agent dependent on the report. Since the principal can only see the reports, the observed shock process is fully history-dependent.

More formally, we assume that the principal and the agent have the same discount factor $\beta \in (0, 1)$. Furthermore, the agent receives an idiosyncratic taste shock $\theta_t \in \Theta \subset \mathbb{R}$ in period t, and $U : C \times \Theta \to \mathbb{R}$ with $C \subset \mathbb{R}$ is his utility function. The set Θ of taste shocks is discrete and finite with cardinality $|\Theta|$. For the ease of exposition, we assume that Θ has $|\Theta| = m$ elements. To ensure the compactness of the resulting dynamic programming problem's state space, we also assume that the set of feasible consumption is a closed and bounded interval $C = [\underline{c}, \overline{c}]$ for some $\underline{c}, \overline{c} \in \mathbb{R}$.

The idiosyncratic shocks are stochastic, and the history of shocks is denoted by: $\theta^t = (\theta_1, \ldots, \theta_t) \in \prod_{i=1}^{t} \Theta = \Theta^t$. $\pi_t(\theta^t)$ denotes the probability of realization of the history θ^t. The agent learns his type at the beginning of period t and therefore has the information θ^t to condition her decisions on. The idiosyncratic shock θ_t depends only on the prior realization θ_{t-1} and therefore evolves following a first-order Markov process:

$$\pi_t \left(\theta_t | \theta^{t-1} \right) = \pi \left(\theta_t | \theta_{t-1} \right), \forall \theta^{t-1} \in \Theta^{t-1}, \theta_t \in \Theta. \tag{1}$$

Consequently, $\pi_t(\theta^t \mid \theta^s)$ for $t > s$ denotes the probability that we observe θ^t given history θ^s. We set $\pi_t(\theta^t \mid \theta^s) = 0$ if θ^s is not identical to the first s components of θ^t. Finally, we sort Θ in ascending order and enumerate it, that is, $\Theta = \{\theta_{(1)}, \ldots, \theta_{(m)}\}$ with $\theta_{(i)} < \theta_{(i+1)}$.

The principal, chooses consumption allocations $c_t : \Theta^t \to C$. Below, we use the shorthand notation c to denote $\left\{ c_t \left(\theta^t \right) \right\}_{t \geq 1, \theta^t \in \Theta^t}$. The agent's period zero utility can now be written as

$$U_0(c) \equiv \mathbb{E}_0 \left[\sum_{t=1}^{\infty} \beta^{t-1} U \left(c_t, \theta_t \right) \right] = \sum_{t=1}^{\infty} \sum_{\theta' \in \Theta^t} \beta^{t-1} \pi_t \left(\theta^t \right) U \left(c_t \left(\theta^t \right), \theta_t \right), \tag{2}$$

where $\mathbb{E}_0$ is the ex-ante, "period-0" expectation before the initial shock θ_1 was realized.

Note that the principal does not observe the taste shock θ_t. Thus, he relies on reports from the agent. The agent's reporting strategy in time period t is given by $\tilde{\sigma}_t : \Theta^t \to \Theta$, and the principal chooses an allocation rule dependent on these reports $\tilde{c}_t : \Theta^t \to C$. Following the literature, we define $\tilde{\sigma} = \left\{ \tilde{\sigma}_t \left(\theta^t \right) \right\}_{t \geq 1, \theta^t \in \Theta^t}$ and $\tilde{c} = \left\{ \tilde{c}_t \left(\hat{\theta}^t \right) \right\}_{t \geq 1, \hat{\theta}^t \in \Theta^t}$. Taken together, they induce a measure over the consumption path, which is denoted as $\tilde{c} \circ \tilde{\sigma}$. We call $\tilde{\sigma}$ *incentive-compatible* if

$$\mathbb{E}^{\tilde{c} \circ \tilde{\sigma}} \left[\sum_{t=1}^{\infty} \beta^{t-1} U \left(c_t, \theta_t \right) \right] - \mathbb{E}^{\tilde{c} \circ \tilde{\sigma}'} \left[\sum_{t=1}^{\infty} \beta^{t-1} U \left(c_t, \theta_t \right) \right] \geq 0, \text{ for all strategies } \tilde{\sigma}', \tag{3}$$

where $\mathbb{E}^{\tilde{c} \circ \tilde{\sigma}}$ is the expectation with respect to the measure $\tilde{c} \circ \tilde{\sigma}$. In particular, if a reporting strategy is incentive-compatible there is no deviation that improves the agent's payoff.

A critical insight necessary for analyzing the contracting environments we are interested in is the *revelation principle* (see, e.g., Golosov et al. [2016, Th. 1], and references therein). It states that for any incentive-compatible allocation, there is an incentive-compatible allocation with the same consumption path where the agent only reports truthfully. Thus, without loss of generality, we only consider a truth-telling agent and consequently replace Eq. (3) by

$$\sum_{t=1}^{\infty} \sum_{\theta^t \in \Theta^t} \beta^{t-1} \pi_t \left(\theta^t\right) \left[U\left(c_t\left(\theta^t\right), \theta_t\right) - U\left(c_t\left(\sigma'^t\left(\theta^t\right)\right), \theta_t\right)\right] \geq 0 \quad \forall \sigma'. \tag{4}$$

This constraint ensures that truth-telling is incentive-compatible and therefore preferred over lying. The agent has a reservation utility that yields a utility amount $\underline{U}$. As a direct consequence, the principal has to offer more than the reservation utility, that is,

$$U_0(c) \geq \underline{U}. \tag{5}$$

We assume that the principal's preferences are given by the utility function $v : \mathbb{R} \times \Theta \to \mathbb{R}$. Thus, she solves the following optimization problem:

$$\begin{aligned} V(\underline{U}) \equiv \sup_{c} & \sum_{t=1}^{\infty} \sum_{\theta^t \in \Theta^t} \beta^{t-1} \pi_t \left(\theta^t\right) v\left(c_t\left(\theta^t\right), \theta_t\right) \\ & \text{subject to Eqs. (4), (5).} \end{aligned} \tag{6}$$

However, even after all these simplifications, the dynamic adverse selection problem under consideration is still intractable in its present form because it is a time-inconsistent dynamic programming problem. Its solution depends on time t and thus has no recursive formulation "as is". To obtain a time consistent problem, we follow the literature and introduce promise utility vectors $(\hat{v}(\theta_{(1)}), \ldots, \hat{v}(\theta_{(m)})) \in \mathbb{R}^m$ as auxiliary state variables. At each point in time, they represent the utility levels that each type of agent will receive in the form of direct transfers and promises for future payments. This is sufficient [see, e.g., Golosov et al., 2016, Sec. 2.5] to obtain a recursive formulation.

In the next step, we introduce the auxiliary variable $w(\hat{\theta}|\theta)$ to be the utility promise if $\hat{\theta}$ is reported, but θ actually happened. Written in a recursive form, we then obtain a new constraint called the *promise-keeping* constraint. It ensures that the utility promises $\hat{v}\left(\theta_{(1)}\right), \ldots, \hat{v}\left(\theta_{(m)}\right)$ that were made in the previous period are fulfilled by a combination of payments in the current period and future promises, that is,

$$\hat{v}\left(\theta_{(j)}\right) = \sum_{\theta \in \Theta} \pi\left(\theta|\theta_{(j)}\right) [U(c(\theta), \theta) + \beta w(\theta|\theta)], \quad \forall j \in \{1, \ldots, m\}. \tag{7}$$

The incentive compatibility constraint turns into

$$U(c(\theta), \theta) + \beta w(\theta|\theta) \geq U(c(\hat{\theta}), \theta) + \beta w(\hat{\theta}|\theta), \quad \forall \theta, \hat{\theta} \in \Theta, \tag{8}$$

which ensures that *one-shot* deviations from the truth-telling path are excluded. This condition is sufficient to ensure that the agent reports truthfully throughout.

Next, we need to establish that there is a state space $\mathcal{V}$ such that for any $\hat{v} \in \mathcal{V}$, there is a $w(\theta|\cdot) \in \tilde{\mathcal{V}}$ that satisfies the promise-keeping (cf. Eq. (7)) and incentive constraints (cf. Eq. (8)). Following Abreu et al. [1986, 1990], we define a set-valued mapping as

$$\begin{aligned}
\mathcal{A}\tilde{\mathcal{V}} = \Big\{ & \hat{v}(\cdot) \in \mathbb{R}^m \mid \\
& \hat{v}\left(\theta_{(j)}\right) = \sum_{\theta \in \Theta} \pi\left(\theta|\theta_{(j)}\right)\left[U(c(\theta),\theta) + \beta w(\theta|\theta)\right], \quad j = 1, \ldots, m \\
& U(c(\theta),\theta) + \beta w(\theta|\theta) \geq U(c(\hat{\theta}),\theta) + \beta w(\hat{\theta}|\theta), \quad \forall \theta, \hat{\theta} \in \Theta \\
& c(\theta) \in C, w(\theta|\cdot) \in \tilde{\mathcal{V}}, \quad \forall \theta \in \Theta \Big\}.
\end{aligned} \tag{9}$$

From Abreu et al. [1986, 1990] and Golosov et al. [2016, Prop. 8], it is known that there exists a set-valued fixpoint $\mathcal{V}$, that is, $\mathcal{V} = \mathcal{A}\mathcal{V}$. This set is the domain of our recursive dynamic programming problem's value function, and can be of irregular, that is, non-hypercubic geometry, and high-dimensional.[11]

In sum, the recursive formulation for the value function $K(\cdot,\cdot)$ of the dynamic incentive problem now reads as follows:

$$\begin{aligned}
K\left(\hat{v}(\cdot), \theta_-\right) = & \max_{\{c(\theta), w(\theta|\cdot)\}_{\theta \in \Theta}} \sum_{\theta \in \Theta} \pi\left(\theta|\theta_-\right)\left[v(c(\theta),\theta_-) + \beta K(w(\theta|\cdot),\theta)\right] \\
& \hat{v}\left(\theta_{(j)}\right) = \sum_{\theta \in \Theta} \pi\left(\theta|\theta_{(j)}\right)\left[U(c(\theta),\theta) + \beta w(\theta|\theta)\right], \quad j = 1, \ldots, m \\
& U(c(\theta),\theta) + \beta w(\theta|\theta) \geq U(c(\hat{\theta}),\theta) + \beta w(\hat{\theta}|\theta), \quad \forall \theta, \hat{\theta} \in \Theta \\
& c(\theta) \in C, w(\theta|\cdot) \in \mathcal{V}, \quad \forall \theta \in \Theta,
\end{aligned} \tag{10}$$

where θ_- denotes the current discrete shock. Moreover, the state space is given by $\mathcal{V} \times \Theta$, and v denotes the principal's utility. Note that this problem no longer depends on the time t. However, as can be seen in Eq. (10), we still have the unknown set $\mathcal{V}$ as part of the constraints. More precisely, the issue is that the auxiliary problem stated in Eq. (10) contains promise utilities that are absent in the original formulation given in Eq. (6). Therefore, we describe in the following how to obtain the solution to the original problem from the solution to the auxiliary problem. Given a seed type θ_0 for the Markov chain, we pick a state in $\mathcal{V}$ that maximizes the value function, that is,

$$v_{max} \in \arg\max_{\hat{v} \in \mathcal{V}} K(\hat{v}, \theta_0). \tag{11}$$

[11]For notational simplicity, we describe here a problem where the feasible set does not change depending on the taste shock. However, our methodology is not limited to this setting: if there are constraints that depend on the current shock θ–as in the model Fernandes and Phelan [2000] that we consider in our numerical experiments below–so will the feasible set. In such situations, we will denote a feasible set that depends on the shock as $\mathcal{V}(\theta)$.

Then, given the sequence of shock realizations $\theta_0, \theta_1, \theta_2, \ldots$, the principal's optimal policy is recursively defined by setting initially

$$\begin{aligned} \hat{v}_0 &= v_{max}, \\ c_1 &= c(\theta_1), \\ \hat{v}_1 &= w(\theta_1|\cdot), \end{aligned} \tag{12}$$

where $c(\theta_1)$ and $w(\theta_1|\cdot)$ are the optimal solutions to Eq. (10) given a state $(\hat{v}_0, \theta_0)$. Thus, at time $t = 1$, we solve problem (10) with the state $(\hat{v}_1, \theta_1)$, and again define

$$\begin{aligned} c_2 &= c(\theta_2), \\ \hat{v}_2 &= w(\theta_2|\cdot). \end{aligned} \tag{13}$$

Proceeding in an iterative fashion yields a sequence of consumption transfers $c_1, c_2, \ldots$, which then are an optimal solution to Eq. (6).

3.2 A Stylized Adverse Selection Model with Heterogeneous Agents

To study the question to what degree heterogeneity in risk drives adverse selection in insurance markets, we now posit a stylized model with heterogeneous agents and multiple persistent types per agent, thereby building on the environment outlined in Sec. 3.1. Since the benchmark model discussed above can be scaled up in a straightforward and meaningful way, only minimal notational and conceptional extensions are required. For the explicit parameterization of the model presented here, we refer the reader to Sec. 5.1.

We consider again a discrete-time economy with n agents of m_i different shocks per agent i. The agents are further differentiated by their preferences and risk profile. That is, let U_i and $\mathbf{\Pi}^{(i)}$ denote the utilities and individual Markov transition matrices, respectively. The agents are fully aware of their type. In addition, we assume that the principal cannot identify the agent type and thus has to rely on incentive mechanisms to elicit the correct type. Therefore, the principal has to offer a contract for each type of agent to incentivize them to report their types truthfully. Consequently, $\sum_i m_i$ different contracts are offered, and the agents self-select the contract corresponding to their risk type and current shock.

Due to the one-shot deviation principle (cf. Sec. 3.1), it is not possible to trivially reduce the complexity of the problem by simply considering each agent separately: The principal has to simultaneously keep track of all agent types to ensure truthful reporting (cf. Eq. (8)), since the truthful report always has to dominate the false report. Thus, the resulting dynamic adverse selection problem is $\sum_i m_i$–dimensional, with a feasible set $\mathcal{V} \subset \mathbb{R}^{\sum_i m_i}$ that grows exponentially with the number of agents and respective types, and that can again can be of irregular geometry.

Let $\mathbf{\Pi}^{(\mathrm{i})} = \pi(\theta_k^{(i)}|\theta_l^{(i)})_{k=1,\ldots,m_i,l=1,\ldots,m_i}$ denote the $m_i \times m_i$ transition matrix of an individual agent i (cf. Eq. (1)), that is, each agent follows a first-order Markov process. It is now straightforward to generalize this setting to n agents by assembling the transition matrices

$\mathbf{\Pi^{(i)}}$ from the individual agents into one single matrix as follows:

$$\mathbf{\Pi} = \begin{bmatrix} \mathbf{\Pi^{(1)}} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{\Pi^{(2)}} & & \\ \vdots & & \ddots & \\ \mathbf{0} & & & \mathbf{\Pi^{(n)}} \end{bmatrix}. \tag{14}$$

With this setup, we assume to model n distinct agents, with no transition from one to the other. It would be simple for our method to consider the more general case where an agent can transition to other risk types. However, such an extension would, for our purposes, just unnecessarily complicate the problem as it would render a comparison across agents more difficult.

With these notational extensions at hand, the recursive model formulation of a multi-agent problem follows from the single-agent setup:

$$\begin{aligned} K\left(\hat{v}\left(\cdot\right), \theta_{-}^{(i)}\right) = & \max_{\left\{c\left(\theta_{(k)}^{(l)}\right), w\left(\theta_{(k)}^{(l)}|\cdot\right)\right\}} \sum_{j=1}^{m_i} \pi\left(\theta_{(j)}^{(i)}|\theta_{-}^{(i)}\right)\left[v\left(c\left(\theta_{(j)}^{(i)}\right), \theta_{-}^{(i)}\right) + \beta K\left(w\left(\theta_{(j)}^{(i)}|\cdot\right), \theta_{(j)}^{(i)}\right)\right] \\ & \hat{v}\left(\theta_{(k)}^{(l)}\right) = \sum_{j=1}^{m_l} \pi\left(\theta_{(j)}^{(l)}|\theta_{(k)}^{(l)}\right)\left[U_i\left(c\left(\theta_{(j)}^{(l)}\right), \theta_{(j)}^{(l)}\right) + \beta w\left(\theta_{(j)}^{(l)}|\theta_{(j)}^{(l)}\right)\right], \quad \forall l, k \\ & U_i\left(c\left(\theta_{(k)}^{(l)}\right), \theta_{(k)}^{(l)}\right) + \beta w\left(\theta_{(k)}^{(l)}|\theta_{(k)}^{(l)}\right) \geq U_i\left(c\left(\theta_{(p)}^{(q)}\right), \theta_{(k)}^{(l)}\right) + \beta w\left(\theta_{(p)}^{(q)}|\theta_{(k)}^{(l)}\right), \quad \forall l, k, p, q \\ & c\left(\theta_{(k)}^{(l)}\right) \in C, w\left(\theta_{(k)}^{(l)}|\cdot\right) \in \mathcal{V}, \quad \forall l, k \end{aligned} \tag{15}$$

In analogy to the single-agent model (cf. Eq. (10)), any solution to problem (15) will have the property that agents will truthfully report their type and shock at any time. In particular, every agent will choose the contract corresponding to his type.

Solving a model with increasingly many agents and persistent shocks becomes a challenging endeavor due to the curse of dimensionality. Thus, an efficient solution technique that complements the research question by rendering it numerically tractable becomes crucial. Thus, in the following, we present a novel solution framework that can handle these types of models.

4 Dynamic programming using Gaussian Processes

This section introduces a generic computational framework to solve dynamic incentive problems with multiple agents, persistent shocks, and potential non-linearities in the value and policy functions. Solving such models computationally is a formidable task since we have to deal with a variety of complex issues at the same time: First, we have to deal with the feasible sets of utility promises. In most of the interesting cases, such sets have a non-trivial, that is, a non–hypercubic geometry [see, e.g., Fernandes and Phelan, 2000, Broer et al., 2017]. Second, we have to solve a dynamic incentive problem recursively, for example, with value function iteration [see, e.g., Judd, 1998, Ljungqvist and Sargent, 2000]. Thus, we need to repeatedly approximate and evaluate potentially high-dimensional functions at arbitrary coordinates on the entire computational domain. To meet all these challenging modeling demands, we propose to combine ideas from penalization methods that emerged in the constrained optimization literature [Luenberger and Ye, 2008, Ch. 13] to relax the recursive formulation of the models, thereby bypassing the difficult numerical task of performing set-valued dynamic programming techniques to characterize the irregularly shaped equilibrium value correspondence (cf. Sec. 2, and references therein). This relaxed problem can then be solved with standard value function iteration techniques that apply to compact sets. The methodological novelty we propose in the context of the presented models is to use GPR in combination with BAL in each iteration step to efficiently approximate the value function and, if needed, the policy functions on the potentially high-dimensional state space.

We now outline our proposed global solution method in six distinct steps. We first characterize in Sec. 4.1 the formal structure of the models our method will be able to tackle and also point out which computational challenges arise. Next, we provide in Sec. 4.2 a brief introduction to GPs, and how they can be used to approximate and interpolate multi-dimensional functions. Subsequently, Sec. 4.3 introduces BAL, a method from reinforcement machine learning that can strategically determine the locations in the state space where functions have to be evaluated (e.g., by solving Bellman equations) to maximize the quality of the GP function approximation with as few points as possible. Sec. 4.4 introduces a numerically tractable reformulation of the dynamic adverse selection models described in Sec. 3 such that they can be solved solely with value function iteration, thereby bypassing the need to explicitly determine the endogenous state space of promise utilities via applying the APS algorithm [Abreu et al., 1986, 1990]. Sec. 4.5 combines all ingredients in a comprehensive value function iteration framework for solving dynamic incentive problems. Finally, we solve in Sec. 4.6 the original model by Fernandes and Phelan [2000] as a basic verification test for our method, given that the solutions for their two-dimensional baseline setting are known. Moreover, Appendix B provides supplementary materials discussing some computational bottlenecks of GPs, and how they can be lifted. Appendix C presents a brief overview of the parallelization scheme of our value function iteration algorithm, which can accelerate the time-to-solution by orders of magnitude if the appropriate hardware is available.

4.1 Abstract Problem Formulation & Value Function Iteration

Recall that throughout this paper the abstract class of models we consider are discrete-time, infinite-horizon stochastic optimal decision-making problems. We briefly characterize them here by the subsequent general description: let $x_t \in \mathcal{B} \subset \mathbb{R}^d$ denote the state of the economy at time $t \in \mathbb{N}^+$ of dimensionality $d \in \mathbb{N}$. Controls (actions) are represented by a *policy function* $v : \mathcal{B} \longrightarrow \zeta$, where $\zeta \subset \mathbb{R}^{d_c}$ is the space of possible controls. The discrete-time transition function of the economy from one period to the next is given by some distribution π, which depends on the current state and policies, that is, $x_{t+1} \sim \pi(\cdot|x_t, v(x_t))$. The transition function π that stochastically maps a state-action pair to a successor state is assumed to be given, whereas the policy function v needs to be determined from optimality conditions. The standard way to do so is to use dynamic programming (see, e.g., Bellman [1961], Stokey et al. [1989b]), where the task is to find an infinite sequence of *controls* $\{\chi_t\}_{t=0}^{\infty}$ to maximize the *value function* $V(x_0) := \mathbb{E}\left[\sum_{t=0}^{\infty} \beta^t r(x_t, \chi_t)\right]$ for an initial state $x_0 \in \mathcal{B}$, $r(\cdot,\cdot)$ is the so-called *return function*, and $\chi_t \in \Gamma(x_t) \subset \mathcal{B}$, with $\Gamma(x_t)$ being the set of feasible choices given x_t. The discount factor $\beta \in (0,1)$ weights future returns. Dynamic programming seeks a time-invariant policy function v mapping the state x_t into the action χ_t, such that for all t, $\chi_t = v(x_t) \in \Gamma(x_t)$, and $\{\chi_t\}_{t=0}^{\infty}$ solves the original problem. The *principle of optimality* states that we can find such a solution by solving the *Bellman equation*, that is,

$$V(x) = \max_{\chi}\{r(x,\chi) + \beta\mathbb{E}[V(\tilde{x})]\}, \tag{16}$$

where the successor state is distributed as $\tilde{x} \sim \pi(\cdot|x,\chi)$. The solution is a fixed point of the Bellman operator T, defined by $(TV)(x) = \max_{\chi}\{r(x,\chi) + \beta\mathbb{E}[V(\tilde{x})]\}$. Under appropriate conditions (see, e.g., Stokey et al. [1989b]) the Bellman operator is a contraction mapping. In this case, iteratively applying the operator T provides a sequence of value functions that converges to a unique fixed point. This procedure is called *value function iteration* (for a textbook treatment, see, e.g., Bertsekas [2000], Judd [1998], Rust [1996], or Ljungqvist and Sargent [2000]) and is motivated by this theoretical justification and numerically implements the iterative application of the Bellman operator to successive approximations of the value function. The corresponding dynamic programming recursion thus starts from any bounded and continuous guess for the value function, and the solution is approached in the limit as $j \longrightarrow \infty$ by iterations on

$$V^{j+1}(x) = T(V^j)(x) := \max_{\chi^{j+1}}\{r(x,v) + \beta\mathbb{E}\left[V^j(\tilde{x})\right]\}. \tag{17}$$

In practice, we say that value function iteration has converged if numerical convergence in some norm, for example,

$$||V^{\tau}(\cdot) - V^{\tau-1}(\cdot)||_{\infty} \leq \epsilon_{VFI}, \quad \epsilon_{VFI} \in \mathbb{R}^{+}, \tag{18}$$

and at some iteration step τ is reached. The (approximate) equilibrium value function is denoted as $V^* = V^{\tau}$ and the corresponding optimal policy functions, as $\chi^* = \arg\max V^*$.

No matter how rich the exact model specifications of a dynamic incentive problem are, the key roadblocks remain the same when one attempts to solve them numerically: to efficiently approximate and interpolate in every iteration step high-dimensional value and policy functions on potentially irregularly shaped geometries (cf. Sec. 4.5 for more details). We, therefore, describe next in Secs. 4.2 and 4.3 how we tackle these challenges by combining GPs with BAL.

4.2 Function Approximation and Interpolation with Gaussian Processes

In the following, we briefly introduce GPR, a nonparametric regression method from supervised machine learning that has universal approximation properties (see, e.g., Micchelli et al. [2006]) and that we will use to approximate and interpolate multivariate policy and value functions (for more details, see, e.g., Rasmussen and Williams [2005], Murphy [2012]).

Given a so-called *training data* set $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ consisting of N input states $x_i \in \mathcal{B} \subset \mathbb{R}^d$ and corresponding observations $y_i \in \mathbb{R}$,[12] where $y_i = f(x_i) + \varepsilon$, $\varepsilon \sim \mathcal{N}\left(0, \sigma_\epsilon^2\right)$,[13] we want to infer a model of the yet unknown function f that generated the data such that we then can make predictions at a point x_* that we have not seen in the training set, that is, we obtain an interpolation value $f(x_*)$. The matrix $\mathbf{X} = [x_1, \ldots, x_\mathrm{N}]$ contains the so-called *training inputs*, whereas $\mathbf{y} = [y_1, \ldots, y_n]^\top$ consists of corresponding observations, that is, *training targets* that can – as in our case – be generated via computer code,[14] or stem from empirical data.

To enable predictions based on information contained in $\mathcal{D}$, we must make assumptions about the characteristics of the underlying functions. A GP defines a distribution over functions such that if we pick any two or more points in a function (i.e., different input–output pairs), observations of the outputs at these points follow a joint (multivariate) Gaussian distribution. More formally, a GP is defined as a collection of random variables, any finite number of which have a joint (multivariate) Gaussian distribution. We start by defining a probability measure on the function space, where f lives corresponding to our beliefs. Before seeing any data, we model our state of knowledge about f by assigning a GP prior to it, that is:

$$f(x) \sim \mathcal{GP}\left(\mu(x), k\left(x, x'\right)\right). \tag{19}$$

A GP is a distribution over functions and is defined by a mean and a covariance function.

[12]For notational simplicity, we restrict ourselves to the univariate case. However, all expressions derived in the following carry over to the situation where GPs have to deal with multivariate output [Bilionis et al., 2013].

[13]This assumption is similar to that made in linear regression, in that we assume an observation consists of an independent "signal" term $f(x)$ and "noise" term ϵ . In GPR, however, we assume that the signal term is also a random variable that follows a particular distribution. This distribution is subjective in the sense that the distribution reflects our uncertainty regarding the function. The uncertainty regarding f can be reduced by observing the output of the function at different input points. The noise term ϵ reflects the inherent randomness in the observations, which is always present no matter how many observations we make.

[14]In our concrete cases below, the training data will consist of solving a constrained optimization problem as stated for instance in Eq. (10) at N strategically chosen points x_i from within a set $\mathcal{B}$ (see Secs. 4.3, 4.6, and 5 for more details), and where corresponding observations y_i are given by the value and policy function at that particular location in the state space, respectively.

The mean function $\mu(x)$ reflects the expected function value at input x:

$$\mu(x) = \mathbb{E}[f(x)], \tag{20}$$

that is, the average of all functions in the distribution evaluated at input x. The prior mean function is required to model any general trends of the response surface and can have any functional form. However, it is often set to $\mu(x) = 0$ to avoid expensive posterior computations and only do inference via the covariance function. For simplicity, we set it to 0 throughout the remainder of this subsection. The covariance function $k(x, x')$ models the dependence between the function values at different input points x and x':

$$k(x, x') = \mathbb{E}\left[(f(x) - \mu(x))\left(f(x') - \mu(x')\right)\right]. \tag{21}$$

The function k is commonly called the *covariance kernel* of the GP [Rasmussen and Williams, 2005]. The choice of an appropriate kernel is the most crucial part of GPR and needs to be based on assumptions such as smoothness and likely patterns to be expected in the data. A sensible assumption is usually that the correlation between two points decays with the distance between the points. This means that closer points are expected to behave more similarly than points that are further away from each other. One very popular choice of a kernel fulfilling this assumption is the radial basis function kernel (also known as the *square exponential (SE) kernel*), which is defined as

$$k_{\mathrm{SE}}(x, x'; \phi) = s^2 \exp\left\{-\frac{1}{2}\sum_{j=1}^{d}\frac{\left(x^j - x'^j\right)^2}{\ell_j^2}\right\}, \tag{22}$$

with *hyperparameters* $\phi = \{s, \ell_1, \ldots, \ell_d\}$, with $s > 0$ being the variability of the latent function f, and $\ell_j > 0$ the characteristic lengthscale of the j-th input dimension.[15] The exact choice of the kernel within an application boils down to how the modeler encodes prior knowledge about the function(s) to be approximated, such as differentiability and periodicity. In our applications below, we typically work with SE or piecewise polynomial kernels (for more details on kernels, see, e.g., Murphy [2022, Ch. 18.2], and `https://www.cs.toronto.edu/~duvenaud/cookbook`).

Suppose we have collected observations $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$, for example, by solving N Bellman equations at various locations of a set $\mathcal{B}$, and we want to make predictions for new inputs, collected in the matrix $\mathbf{X}_*$, by drawing the corresponding $\mathbf{f}_*$ from the posterior distribution $p(f \mid \mathcal{D})$. By definition of the GP, the previous observations $\mathbf{y}$ and function values $\mathbf{f}_*$ follow a joint multivariate normal distribution, which can be written as

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} k(\mathbf{X}, \mathbf{X}) + \sigma_\epsilon^2 \mathbf{I} & k(\mathbf{X}, \mathbf{X}_*) \\ k(\mathbf{X}_*, \mathbf{X}) & k(\mathbf{X}_*, \mathbf{X}_*) \end{bmatrix}\right), \tag{23}$$

[15]Note that the hyperparameters of the covariance function are typically estimated by maximizing the likelihood [Rasmussen and Williams, 2005].

where $k(\mathbf{X}, \mathbf{X})$ is the covariance matrix between all observed points so far, $k(\mathbf{X}_*, \mathbf{X}_*)$ is the covariance matrix between the newly introduced set of points for which we want to make a prediction, $k(\mathbf{X}_*, \mathbf{X})$ is the covariance matrix between the new input points and the already observed points, and $k(\mathbf{X}, \mathbf{X}_*)$ is the covariance matrix between the observed points and the new input points. Moreover, $\mathbf{I}$ is an identity matrix, and σ_ϵ^2 is the assumed noise level of observations, that is, the variance of ϵ. Using standard results (see, e.g., Rasmussen and Williams [2005], Rasmussen and Nickisch [2010]), the conditional distribution $p(\mathbf{f}_* \mid \mathbf{X}, \mathbf{y}, \mathbf{X}_*)$ is then a multivariate normal distribution with a posterior, which by itself is a GP with mean function

$$\tilde{\mu}(x) = k(x, \mathbf{X}) \left[k(\mathbf{X}, \mathbf{X}) + \sigma_\epsilon^2 \mathbf{I}\right]^{-1} \mathbf{y}, \tag{24}$$

and a covariance kernel that reads:

$$\tilde{k}(x, x') = k(x, x') - k(x, \mathbf{X}) \left[k(\mathbf{X}, \mathbf{X}) + \sigma_\epsilon^2 \mathbf{1}\right]^{-1} k(\mathbf{X}, x'). \tag{25}$$

The previous two equations imply that calculating the posterior mean and covariance of a GP involves first calculating the four different covariance matrices contained in Eq. (23) and then combining them according to Eqs. (24) and (25). Done naively, this step scales as $\mathcal{O}(N^3)$ with the number of observations N in the training set $\mathcal{D}$ [Rasmussen and Williams, 2005].[16] In order to predict $\mathbf{f}_*$, we can simply use the mean function, our best estimate for the desired value, which is given by Eq. (24), and evaluate it at the location of interest: $\tilde{\mu}(x_*)$. Furthermore, Eq. (25) provides the *predictive variance* $\tilde{\sigma}^2(x_*) := \tilde{k}(x_*, x_*)$. The latter can be used to derive point-wise predictive error bars and thus provide information about the model confidence, that is, about the quality of the function approximator at a point x_*. Fig. 1 shows an example posterior mean function after the data from an analytical test function has been observed, as well as the 95% confidence interval.

Note the predictive mean $\tilde{\mu}(x)$ given by Eq. (24) can also be written as

$$\tilde{\mu}(x) = \sum_{i=1}^{N} a_i k(x_i, x), \tag{26}$$

where each x_i is a previously observed input value in $\mathbf{X}$, and the weights are collected in the vector $\boldsymbol{a} = (a_1, \ldots, a_N) = \left(k(\mathbf{X}, \mathbf{X}) + \sigma_\epsilon^2 \mathbf{I}\right)^{-1} \mathbf{y}$. Intuitively, we can think of the GP posterior mean as an approximation of f using N symmetric radial basis functions (RBFs) centered at each observed input. Thus, by choosing a covariance function $k(x, x')$ that vanishes when x and x' are separated by a lot, for example, the SE covariance function (cf. Eq. (22)), we see that an observed input-output will only affect the approximation locally. This observation also establishes a connection between GPR and reproducing-kernel Hilbert spaces, which are discussed in Rasmussen and Williams [2005]. Moreover, the radial basis function has universal approximation and regularization capabilities. Theoretically, the

[16]For contemporary methods to alleviate this performance bottleneck, see Appendix B.

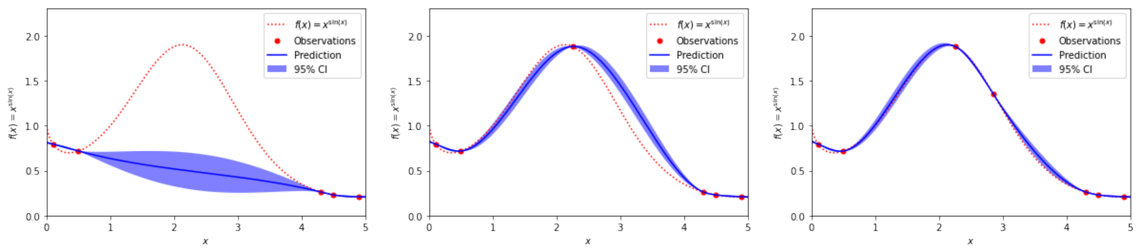

Figure 1: The left panel shows a training data set $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ (red dots) that was generated by evaluating the analytical function $y = x^{sin(x)}$ (red dashed line) at the points $\mathbf{X} = \{0.1, 0.5, 4.3, 4.5, 4.9\}$ to which we fitted a GP. Furthermore, the predictive mean as well as the 95 percent confidence intervals, corresponding to the point-wise predictive mean (blue line) plus and minus two standard deviations (blue shaded area) for each input value of predictive variance, are also shown. The figure in the center shows the same, but with one observation strategically inserted into the training set around $x = 2.2$ via BAL. The panel on the right enhances the training set by another data point around $x = 2.9$ that was added via BAL.

RBF can approximate any continuous function arbitrarily well (see, e.g., Poggio and Girosi [1990], Park and Sandberg [1991], Wu et al. [2012]). For more details on why, in the age of deep learning, GPs can (and should) be used in the presented context of models, see Appendix D, and references therein.

4.3 Exploring the State Space by Bayesian Active Learning

In order to approximate functions efficiently with GPs in general, and specifically, to perform the Bellman recursion to solve dynamic models, we need to find a set of states $\mathbf{X}$ that covers the relevant state space $\mathcal{B}$ in such a way that a GP can approximate value and policy functions sufficiently well with a minimal amount of training data $\mathcal{D}$. The task at hand, thus, is to answer the question of what data one should gather to learn about the function(s) of interest as quickly as possible, especially in our case where training data is expensive to acquire, as we need to solve many constrained optimization problems in each step of the value function iteration.

The simplest way to generate training data is to place the observation points so as to train the Gaussian processes randomly from some compact set $[a, b] \subset \mathbb{R}^d$, $a, b \in \mathbb{R}^d$, for example, via a Halton sequence [Halton, 1964, Niederreiter, 1992]. This strategy can be highly inefficient, particularly in situations where functions that have to be approximated show distinct local features, such as steep gradients in some regions in the state space, since the observations that are used to generate a good approximation of functions may not necessarily be created where they improve on the approximation quality most. To still obtain a good approximation of functions in the presence of strong nonlinearities by naively sampling from the domain of interest, one would have to increase the observations substantially. Since the training of standard GPs scales cubically with the number of observations N, that is, $\mathcal{O}(N^3)$, the runtime consequently would increase drastically with increasing sample size (cf. Appendix B for more details). To improve matters substantially, we propose to use BAL (see, e.g., MacKay [1992], Chaloner and Verdinelli [1995], Krause et al. [2008], Deisenroth et al. [2009], Makarova et al. [2022]). The latter is a technique from the reinforcement learning literature to automatically place observations in regions of the computational domain $\mathcal{B}$ where they improve on the quality of the approximator most according to a chosen metric.[17]

BAL tackles this so-called exploration-exploitation dilemma, that is to say, the use of as few observations as possible to best represent the a priori unknown functions by inducing observations to the training set such that only the relevant part of the state space will be explored. Hence, BAL can be seen as a strategy for optimal data selection to make learning more efficient. In our case, the training data is selected according to a function that we call the *score function*, which is given by

$$U(\tilde{x}) = \sigma_m \tilde{\mu}(\tilde{x}) + \frac{\sigma_v}{2} \log\left(\tilde{\sigma}(\tilde{x})\right), \tag{27}$$

where σ_m and σ_v are positive weighting factors, and where $\tilde{\mu}$ and $\tilde{\sigma}$ are the predictive mean and variance of a GP, trained at input locations $\mathbf{X}$, and evaluated at a point $\tilde{x}$ that is not contained in the training set, respectively. The first term of Eq. (27), given by the predictive mean (cf. Eq. (24)), expresses how much total reward is expected from $\tilde{x}$. The second term of Eq. (27) is given by the predictive variance (cf. Eq. (25)) and measures how uncertain the GP approximation is expected to be, given $\mathbf{X}$, in terms of Shannon entropy [Chaloner and Verdinelli, 1995]. Moreover, the parameters σ_m and σ_v control exploitation and exploration respectively, and have to be set manually.[18] Thus, for a given observation $\tilde{x}$, one can evaluate Eq. (27) and assign a score on how important it would be to include this particular observation in the training set.

To construct a concrete training set that leverages BAL, we follow the literature and proceed as follows: First, we start from a small set $\mathbf{X}$ of initial input locations that are generated via a Halton sequence from a set $\mathcal{B}$ and train a GP on the resulting training targets $\mathbf{y}$. Second, we generate s candidate points $\tilde{x}_i \in \mathcal{B}$ at which we evaluate Eq. (27) given the fitted GP at every $\tilde{x}_i$. Third, we rank all the observations $\tilde{x}_i$ given their score (cf. Eq. (27)) and add $\xi \in \mathbb{N}$ points with the highest score to our set of training inputs $\in \mathbf{X}$. Thus, the set $\mathbf{X}$ dynamically grows as long as the size of the training set is below some pre-defined size, but only in the regions of the state space where the observations matter, that is to say, in a data-efficient fashion.

We now demonstrate the ability of the joint workings of GPs and BAL to efficiently approximate and interpolate functions on three analytical examples $f : [a, b] \to \mathbb{R}$, $[a, b] \subset$

[17]In loose terms, BAL could be considered to be the grid-free equivalent of an adaptive sparse grid algorithm [Brumm and Scheidegger, 2017]. In the latter global solution method, the adaptivity of the grid now ensures that grid points are placed close to the nonlinearities while the grid remains coarse where policy functions are smoother.

[18]The results reported in Secs. 4.6 and 5 are not highly sensitive to the values prescribed. In practical applications, choosing $\sigma_m = 1$ and $\sigma_v = 10$ yielded satisfactory results.

$\mathbb{R}^d, a, b \in \mathbb{R}^d$, which are adopted from a representative suite of test problems by Genz [1984].[19] We generate various GP approximations to those functions, and measure their accuracy as follows: We randomly generate $N_{test} = 1,000$ test points x_i from a uniform distribution on $[a, b] \subset \mathbb{R}^d$, $a, b \in \mathbb{R}^d$, and compute the mean approximation error, which is given by

$$\frac{1}{N}\sum_{i=1}^{N} |f(x_i) - \tilde{\mu}(x_i)|, \tag{28}$$

where $\tilde{\mu}(x_i)$ is the predictive mean of the GP (cf. Eq. (24)) approximating the test function f.

As a first example, we consider $y = f(x) = x^{sin(x)}$, where $x \in [0, 5]$. The initial training set $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ consists of only five samples, to which we fit a GP (cf. the left panel of Fig. 1). Next, we generate $N_{cand} = 1,000$ candidate points, and score them according to Eq. (27). The top-ranked sample is added to the training set, and the GP approximation is re-computed (cf. the middle panel of Fig. 1). This process is repeated one more time (cf. the right panel of Fig. 1). By adding only two extra sample points, the mean approximation error drops from 51% down to 1%. From those three panels it becomes clear that using GPs in combination with BAL can very rapidly increase the quality of the function approximation by monitoring where one is uncertain about the predictive mean and improving on it by adding observations at locations of the computational domain where they are needed most.

As a second example, we approximate $f(x) = |\sin((\frac{\pi}{2}) * x)|$, where $x \in [-1, 1]$. We compare two situations: In the first, we approximate f with GPs by using training data that was generated by BAL, and monitor the error as a function of the number of samples in the training set. We start BAL with a single, randomly chosen data point, and subsequently enhance the training set until it consists of ten points. Next, we approximate the same f with the same number of training points, which are generated by randomly sampling from the domain of interest. The left panel of Fig. 2 depicts the situation where the training set consists of 5 sample points, out of which four were strategically chosen via BAL. The central panel of Fig. 2 shows the same, but for the case where the GP approximation was based on a randomly generated training set. The right panel of Fig. 2 shows the convergence behavior of the GPs (as a function of samples in the training set) for approximating f naively (denoted as "uniform") or via BAL (denoted as "BAL"). Strikingly, BAL outperforms the naive setting by almost one order of magnitude in accuracy for a fixed number of samples due to its ability to add samples to the training set where needed most according to Eq. (27). We expect such an effect to be even more pronounced in a multi-dimensional setting.

In the third example, we consider a two-dimensional test from the "Gaussian peak

[19]The corresponding Python codes illustrating BAL can be found under the following URL: https://github.com/GaussianProcessesForDynamicEcon/DynamicIncentiveProblems/tree/main/analytical_examples.

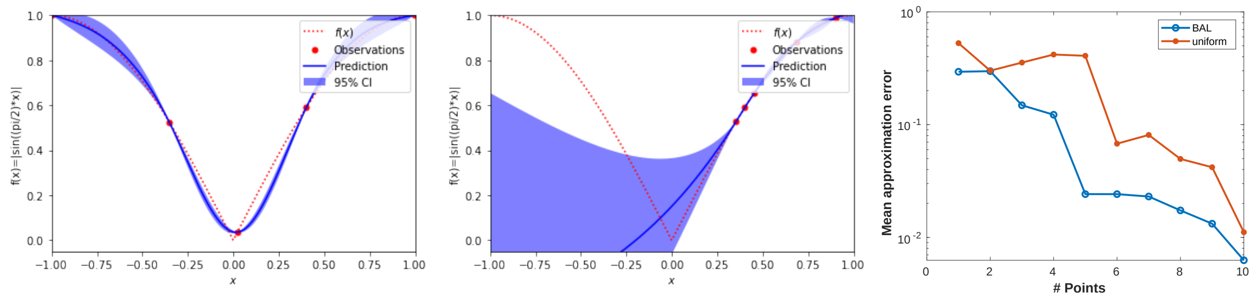

Figure 2: The left panel shows a training set (red dots) that was generated by evaluating the analytical function $f(x) = |\sin((\frac{\pi}{2}) * x)|$ (red dashed line) at five points, four of which were strategically chosen via BAL. Furthermore, the predictive mean, as well as the 95 percent confidence intervals corresponding to the point-wise predictive mean (blue line) plus and minus two standard deviations (blue shaded area) for each input value of predictive variance, are also shown. The figure in the middle depicts the same, but for a GP fitted to a randomly generated training set. The right panel compares the interpolation error (see Eq. (28)) of GPs approximating f naively (denoted as "uniform"), or via BAL (denoted as "BAL").

family" of functions, that is,

$$f(x) = \exp\left(-\sum_{i=1}^{d} a_i^2 (x_i - u_i)^2\right), \tag{29}$$

and choose $d = 2$, $x \in [0,1]^2$. Furthermore, we set $a_i = (5,5)$ and $u_i = (0.8, 0.8)$. The top left panel of Fig. 3 depicts the resulting function. We have chosen this test case as it loosely resembles the shape of the value functions we wish to approximate below in our dynamic incentive models (cf. Fig. 6). Thus, the performance results found here should carry over. To approximate the function (29), we proceed as in the economic applications (cf. Sec. 4.6). We start by generating an initial training set consisting of 100 points randomly drawn from the two-dimensional domain. Next, we enhance the training set via BAL by an additional 40 points and compute the corresponding approximation errors as a function of the sample size. We compare this setup to the case where the training set is naively enhanced by adding samples randomly from the domain of interest. The top right panel of Fig. 3 displays the extra points added to the training set via BAL. We depict them in two individual batches of the first and second 20 points. This figure illustrates that BAL adds new samples to the training set where they are needed most to improve the performance of the approximator, that is, in the top right corner and at the boundary of the domain. In addition, we display the extra 40 uniform samples we generated, indicating that their random placement is not optimal for increasing the performance of the GP. The lower left panel of Fig. 3 compares the mean approximation error for the GPs, generated from the BAL-enhanced as well as the naively enhanced training set. Strikingly, BAL outperforms

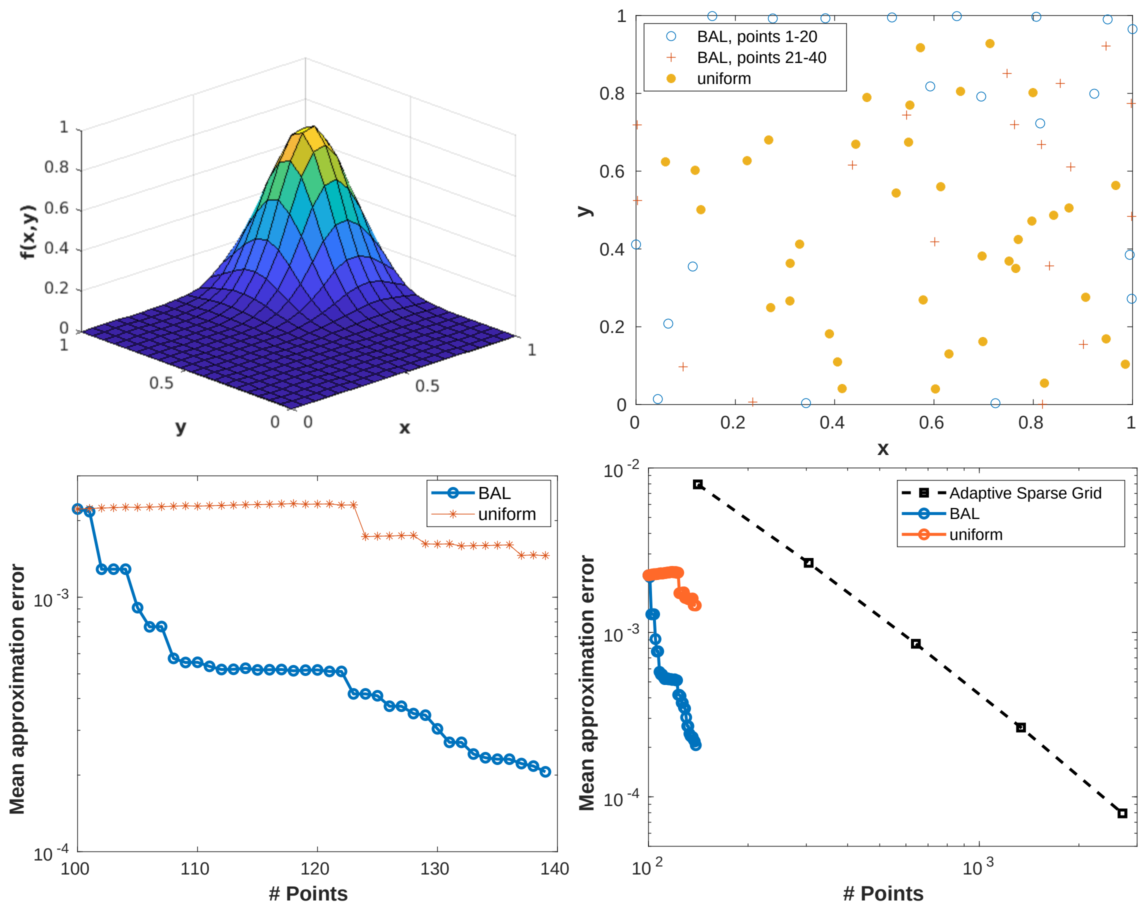

Figure 3: The top left panel depicts the test function (29). The top right panel displays how BAL successively enhances the training set by 40 points. The first 20 points added to this set are depicted as (blue) dots, whereas the subsequently added 20 points are labeled as (red) crosses. This situation is contrasted by adding uniform 40 samples to the original training set (labeled as yellow points). The lower left figure depicts the interpolation error (see Eq. (28)) of GPs approximating f naively (denoted as "uniform"), or via BAL (denoted as "BAL"). The lower left panel shows the same and compares the performance to an adaptive sparse grid of increasing size.

the naive approach by about one order of magnitude. Finally, the right panel of Fig. 3 compares the mean approximation error for the GPs to adaptive sparse grids, a "curse of dimensionality breaking" state-of-the-art method to efficiently approximate multi-variate functions (see, e.g., Brumm and Scheidegger [2017]). The data points reported for the adaptive sparse grid were obtained by computing the approximation error at grid levels 3 to 7 and a refinement threshold $\epsilon = 10^{-6}$. The figure shows that the joint working GPs and BAL can save substantial resources and outperform advanced methods such as adaptive sparse grids.

4.4 A Numerically Tractable Reformulation of the Recursive Problem

Solving dynamic adverse selection models in their recursive form poses significant challenges to the solution method: one needs to solve a (potentially high-dimensional) dynamic programming problem on an a priori unknown and irregularly shaped set. If the problem is simple enough, it can be possible to find an analytical solution [Mailath et al., 2002]. However, in general, one needs to use a numerical approach. The literature thus far usually applies a two-step procedure: First, one needs to determine the feasible set ex ante via a variant of the APS algorithm [Abreu et al., 1986, 1990]. Secondly, one has to solve a dynamic program on the domain determined in step 1. Several versions of this approach have been proposed thus far. However, none of them are generally applicable, are often restricted to a sub-class of models such as those with convex feasible sets, and are not scalable beyond two to three dimensions as they all suffer from the curse of dimensionality (cf. Sec. 2).

To lift the abovementioned issues, we propose a generic, numerically tractable penalty reformulation of the Bellman equation of dynamic adverse selection problems that provides three advantages over the previous literature. First, it bypasses the need to predetermine the feasible set. Second, it is generally applicable to dynamic adverse selection problems and is not restricted to a particular subset of that class of models, such as those with convex feasible sets. Third, computing solutions to the reformulated problem reduces to solving an ordinary dynamic programming problem such as the one stated in Sec. 4.1. Thus, decades of theory and practice in solving these well-studied standard problems carry over. While other numerical methods could be applied, below we will specifically use GPs (cf. Sec. 4.2) in combination with BAL (cf. Sec. 4.3) as a highly efficient way of performing value function iteration in the presented types of models, thereby opening the door to tackle dynamic incentive models of unprecedented complexity (cf. Sec. 4.5 below).

The basic idea behind our reformulation is the following: While the feasible set $\mathcal{V}$ for the models under consideration is ex-ante unknown, we are usually able to give an estimate of a box $\mathcal{B}$ containing it, that is, $\mathcal{V} \subset \mathcal{B}$. Thus, we start the value function iteration algorithm by uniformly drawing a moderate number of sample points $\mathbf{X}$ from the compact domain $\mathcal{B}$ at which we would like to evaluate the Bellman equation in order to obtain a training set $\mathcal{D}$ to approximate the value and policy functions with GPs. However, if we were to solve the original Bellman equations at these randomly drawn states, some of the individual optimization problems might be infeasible, as depicted in the left panel of Fig. 4. To this end, we propose to relax the original recursive model by introducing slack variables on the promise-keeping constraints. Furthermore, we also add a penalty to the objective function that is large and negative whenever the slacks are non-zero. This measure serves two purposes: First, it renders all optimization problems numerically feasible; that is, it is possible to evaluate the relaxed Bellman operator on the entire domain $\mathcal{B}$ and thus to fit GPs over all training data. Secondly, the relaxed problem provides a binary classifier of the training data, that is, $\mathcal{D} = \mathcal{D}_{\mathcal{V}} + \mathcal{D}_{\mathcal{B} \setminus \mathcal{V}}$. There are two situations where the relaxed Bellman equation can be deemed infeasible: The first is when the penalty function in

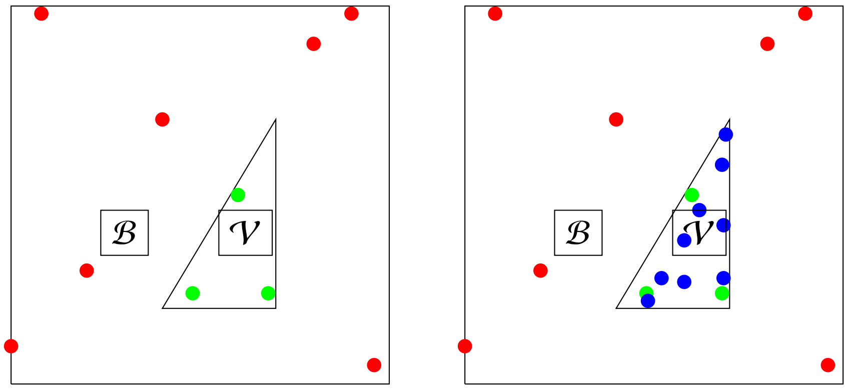

Figure 4: The left panel depicts a small set of sample points $\mathbf{X}$ that are uniformly drawn from the (hypercubic) set $\mathcal{B}$ that contains the *true* feasible set $\mathcal{V}$ (schematically depicted as a triangle). The red dots represent the points where the original Bellman operator (cf. Eqs. (10) or (15)) would be deemed infeasible to solve, whereas the green points label locations in the state space where the solution to the original problem can be classified as feasible. The right panel shows the same situation as the left panel, but with the difference that the blue labeled points were systematically added with BAL to the original training set in the neighborhood of the points that were previously deemed feasible, thereby focusing the computational resources where they are needed the most.

the objective is non-zero. The second case is indirect: An infeasible state, that is, a state outside of $\mathcal{V}$, can have a zero penalty. The reason for this is that the promise-keeping and incentive constraints might be feasible, but only by providing infeasible future promises, that is, promises where the penalty is non-zero. However, these future promises will have a lower utility due to the penalty term. Thus, $\mathcal{D}_{\mathcal{V}}$ provides a first, rough approximation of the feasible set $\mathcal{V}$.[20] Next, to improve both the approximate value and policy functions on the *true* feasible set, we apply BAL to $\mathbf{X}_{\mathcal{V}}$ (the training points determined by $\mathcal{D}_{\mathcal{V}}$), thereby systematically increasing the number of samples in the region of interest[21] until global error estimates (see, e.g., Eq. (33) below) become sufficiently small. Thus, a high-quality approximation of value and policy functions on an approximate feasible set is reached (cf. the right panel of Fig. 4).[22]

After having laid down the basic intuition for our proposed reformulation, we next outline its mathematical details in the context of the baseline model introduced in Sec. 3.1. Subsequently, we will briefly elaborate on how the approach generalizes to the heteroge-

[20]In practical applications, the cardinality of the initial draw of $\mathbf{X}$ needs to be sufficiently large such that $\mathcal{D}_{\mathcal{V}}$ is non-empty.

[21]In our applications, we perform BAL by generating candidate points along a simulation path of 2,000 steps that is started from the maximizer. For more details, see Secs. 4.5 and 4.6.

[22]Note that in practice, after every few steps of the value function iteration we add new points to the training set via BAL, while preserving the points from the previous iteration steps. However, the training targets, of course, are have to be recomputed, and the training points are relabelled until numerical convergence is reached. For more practical details, see Sec. 4.5.

neous agent model introduced in Sec. 3.2.

In the first step, we need to determine a hypercube (cf. Fig. 4), that is, a compact set $\mathcal{B}$ which is the product of closed intervals and which contains $\mathcal{V}$.[23] Recall from Sec. 3.1 that we assume that the consumption transfers c are bounded from above by $\bar{c}$, and that the agent's utility is to be bounded from below by $U(\underline{c},\theta)$. With these bounds, we can determine the following hypercube from which we can draw samples $\mathbf{X}$, that is,

$$\mathcal{V}(\theta) \subset \mathcal{B} = \prod_{\tilde{\theta}\in\Theta} [U(\underline{c},\tilde{\theta})/(1-\beta), U(\bar{c},\tilde{\theta})/(1-\beta)] \quad \forall\theta. \tag{30}$$

Next, we need to replace the original recursive formulation with a penalized one that renders the Bellman equation feasible on the entire domain $\mathcal{B}$, and not solely on $\mathcal{V}$ [Luenberger and Ye, 2008, Ch. 13]. To do so, we need to replace the set $\mathcal{V}$ with the hypercube $\mathcal{B}$ in Eq. (10). Furthermore, we need to add additional variables and a penalty function to Eq. (10) to relax the promise-keeping that otherwise would be infeasible outside of $\mathcal{V}$ (cf. Fig. 4). Thus, we rewrite the Bellman equation as follows:

$$\begin{aligned}
K(\hat{v}(\cdot),\theta_-) = \max_{c(\theta),w(\theta|\cdot),\xi_j} & \sum_{\theta\in\Theta} \pi(\theta|\theta_-)\left[v(c(\theta),\theta_-) + \beta K(w(\theta|\cdot),\theta)\right] - M\sum_{j=1}^{m}\xi_j^2 \\
& \hat{v}\left(\theta_{(j)}\right) + \xi_j = \sum_{\theta\in\Theta} \pi\left(\theta|\theta_{(j)}\right)\left[U(c(\theta),\theta) + \beta w(\theta|\theta)\right], \quad \forall j \in \{1,\dots,m\} \\
& U(c(\theta),\theta) + \beta w(\theta|\theta) \geq U(c(\hat{\theta}),\theta) + \beta w(\hat{\theta}|\theta), \quad \forall\theta,\hat{\theta}\in\Theta \\
& c(\theta)\in C, w(\theta|\cdot)\in\mathcal{B} = \prod_{\theta\in\Theta}[U(\underline{c},\theta)/(1-\beta), U(\bar{c},\theta)/(1-\beta)], \quad \forall\theta\in\Theta,
\end{aligned} \tag{31}$$

where M is a non-negative constant,[24] and $\xi_j \in \mathbb{R}$ are slack variables. Notice that the reformulation of the model given by Eq. (31) renders the Bellman equation (numerically) feasible on the entire domain $\mathcal{B}$ by relaxing the promise-keeping constraints via the variables ξ_j, and by adding a penalty function $-M\sum_{j=1}^{m}\xi_j^2$ to the objective.

There are two ways to use the solution to the relaxed problem (31) to determine whether a point in the state space is infeasible (cf. the right panel of Fig. 4). The first way is by looking at the slack variables: if they are non-zero, the point can be deemed infeasible. The second way is to look at the level of the value function $K(\cdot,\cdot)$ of the relaxed problem. The original problem (10) has a natural minimum that reads as follows:

$$K_{min} = \sum_{\theta\in\Theta} \pi(\theta|\theta_-)\, v(c,\theta_-)/(1-\beta). \tag{32}$$

[23]Note that our numerical approach can, in principle, also deal with unbounded sets. For such a situation, we merely require a good guess for the set to start from. However, in this situation, we lose the convergence guarantees from the theory of dynamic programming over compact sets.

[24]In our numerical applications, we set $M = 10$.

Thus, if the objective function of the relaxed model is lower than the said value at a particular location in the state space, a training point can be classified as infeasible once the value function iteration has converged.[25]

To show that the reformulated problem can be used without concern, we prove now that the relaxed problem provides a solution to the original problem:

Lemma 1 *Let $(\theta_i)_{i=0}^{\infty}$ be a sequence of shocks and $v^{(0)} \notin \mathcal{V}$. Furthermore, let $v^{(i)}$ be the optimal utility promises for expression (31) for a given shock sequence. Then, there exists a $j \in \mathbb{N}$ such that the slack vector of the optimal policy at $v^{(j)}$ has the property $\|\xi\| > 0$. In particular, given an initial state θ_0 and shock realization $(\theta_i)_{i=1}^{\infty}$, we can generate an optimal policy $(\xi_t)_{t=0}^{\infty}$ by using the solution to (31), with an initial promise $v_{max} \in \arg\max_v K(v, \theta_0)$. In this situation, we obtain $\|\xi\| = 0$ at all steps. It follows that v_{max} is an optimal solution to problem (10).*

Proof: Assume that there exists a $w^{(0)} \notin \mathcal{V}$ such that there is an optimal simulation trajectory $w^{(i)}$ such that $\|\xi\| = 0$ at all points along the simulation. Take S to be a set containing $w^{(i)}$ and $\mathcal{V}$. Then, by definition, $w^{(i)} \in \mathcal{A}^n S$ for all n, where $\mathcal{A}$ is the operator defined in (9). Therefore, we have $w^{(0)} \in \mathcal{V}$ according to Golosov et al. [2016, Prop. 8], which is a contradiction.

From Lemma 1, it follows that once we have solved problem (31), it is easy to check whether a solution to the original problem (10) was found by simulating an optimal trajectory that is launched at the global maximum of the value function $K(\cdot, \cdot)$ of $\mathcal{V}$.

The recursive problem stated in Eq. (31) is an ordinary dynamic programming problem over a compact set. Thus, the classical theory [Stokey et al., 1989a, Ch. 9] applies, we have a contraction (cf. Sec. 4.1), and consequently, any general-purpose dynamic programming method for multi-dimensional problems (see, e.g., Cai and Judd [2014], Maliar and Maliar [2014], Brumm et al. [2021], and references therein) could in principle be applied to solve the said model via value function iteration. In addition, our reformulation bypasses the need for pre-computing the feasible set via an APS-type of algorithm [Abreu et al., 1986, 1990]. We will elaborate on this point in greater detail in Sec. 4.5 below, thereby outlining why our reformulation provides a substantial improvement over the previous literature from a computational point of view.

Since the models under consideration in general do not have analytical solutions, but have to be determined numerically, we need measures to assess the credibility and correctness of our computational results. To do so, we follow the best practices of a sub-field from computational sciences called *validate, verify, and uncertainty quantification* (VVUQ; see, e.g., Oberkampf and Roy [2010], and references therein), and propose to use two particular criteria jointly. The first and necessary metric to assess the quality of the solution to the reformulated problem (31) is the contraction of the value function

[25]Note that while we do not have to approximate the feasible set within our reformulated model explicitly, we can still obtain it implicitly. When solving the model recursively, we approximate the value function and policies, including the slack variables, with GPs. Thus, looking either at the approximate slack variables or the value function itself will equip us with a smooth classifier that, on the entire set $\mathcal{B}$, will tell us approximately whether a location in the state space is feasible or not.

iteration in some norm that is computed over the entire set $\mathcal{B}$. The latter is used to stop the value function iteration once numerical convergence between iteration steps is reached (cf. Secs. 4.1 and 4.5). If this error is sufficiently small, we can be optimistic that the global solution to the model is sufficiently accurate and that we are not missing parts of the state space that are part of the optimal solution and not resolved well enough. However, recall that we are not overly interested in the convergence of the relaxed problem on the entire set $\mathcal{B}$, but rather aim to have a high-quality solution for the policies; a potentially much smaller subset. Fernandes and Phelan [2000, p. 236] pointed out this issue: *"The area above $\hat{w}^*(w)$ is, in an important sense, irrelevant. Examination of the Bellman equation [...] reveals that no efficient $t = 0$ contract would ever map to this area."* Thus, we propose a second metric that assesses the quality of the computed optimal solution more precisely where it matters most, that is, in the part of the state space that is being used by an optimal policy, and that BAL (cf. Sec. 4.3, and the right panel of Fig. 4) should accurately trace out during our computations (cf. Sec. 4.6 below). Thus, inspired by the error measures used in Haan et al. [2011] and Azinovic et al. [2022], we propose to look at the value function error along the simulated path of the optimal solution. Concretely, starting from the initial shock type with the corresponding value function maximizer, we simulate the optimal policy, thereby obtaining a sequence of states $\hat{v}^{(1)}_{\theta^{(1)}}, \ldots, \hat{v}^{(s)}_{\theta^{(s)}}$ with shock realizations $\theta^{(1)}, \ldots, \theta^{(s)}$. We then use these points to compare two value functions $K^{\tau-1}$ and K^{τ} at two consecutive time steps, and where the convergence is reached at the iteration step τ (cf. Alg. 1),

$$\frac{1}{s}\sqrt{\sum_{f=1}^{s}\left(\frac{K^{\tau}(\hat{v}^{(f)}_{\theta^{(f)}}, \theta^{(f)}) - K^{\tau-1}(\hat{v}^{(f)}_{\theta^{(f)}}, \theta^{(f)})}{1 + |K^{\tau}(\hat{v}^{(f)}_{\theta^{(f)}}, \theta^{(f)})|}\right)^2}, \tag{33}$$

where the denominator is used to ensure the error is unit free and can be seen as percentages. This is necessary since we have large changes in the value function across $\mathcal{B}$ due to the penalty.

Recall that three steps have to be taken to reformulate an adverse selection model, that is, i) finding a $\mathcal{B}$ that contains $\mathcal{V}$, ii) adding slacks to the promise-keeping constraints, and iii) adding a penalty to the objective function, generalize to any other setting. Thus, it is straightforward to rewrite the heterogeneous agent model given by expression (15) into a recursive problem that no longer depends explicitly on the feasible set $\mathcal{V}$:

$$\begin{aligned}
K\left(\hat{v}\left(\cdot\right), \theta_{-}^{(i)}\right) = & \max_{c\left(\theta_{(k)}^{(l)}\right), w\left(\theta_{(k)}^{(l)}|\cdot\right), \xi_{(k)}^{(l)}} \sum_{j=1}^{m_i} \pi\left(\theta_{(j)}^{(i)}|\theta_{-}^{(i)}\right)\left[v\left(c\left(\theta_{(j)}^{(i)}\right), \theta_{-}^{(i)}\right) + \beta K\left(w\left(\theta_{(j)}^{(i)}|\cdot\right), \theta_{(j)}^{(i)}\right)\right] - M\sum_{l,k}\left(\xi_{(k)}^{(l)}\right)^2 \\
& \hat{v}\left(\theta_{(k)}^{(l)}\right) + \xi_{(k)}^{(l)} = \sum_{j=1}^{m_l} \pi\left(\theta_{(j)}^{(l)}|\theta_{(k)}^{(l)}\right)\left[U_i\left(c\left(\theta_{(j)}^{(l)}\right), \theta_{(j)}^{(l)}\right) + \beta w\left(\theta_{(j)}^{(l)}|\theta_{(j)}^{(l)}\right)\right], \quad \forall l, k \\
& U_i\left(c\left(\theta_{(k)}^{(l)}\right), \theta_{(k)}^{(l)}\right) + \beta w\left(\theta_{(k)}^{(l)}|\theta_{(k)}^{(l)}\right) \geq U_i\left(c\left(\theta_{(p)}^{(q)}\right), \theta_{(k)}^{(l)}\right) + \beta w\left(\theta_{(p)}^{(q)}|\theta_{(k)}^{(l)}\right), \quad \forall l, k, p, q \\
& c\left(\theta_{(k)}^{(l)}\right) \in C, \quad \forall l, k,
\end{aligned} \tag{34}$$

where the extended computational domain $\mathcal{B}$ to draw samples from is given extending Eq. (30) to the multi-agent setting by taking the Cartesian product over all agents and all types.

4.5 A Solution Algorithm for Dynamic Incentive Problems

We now outline how to solve the reformulated dynamic adverse selection models with a standard dynamic programming approach by combining value function iteration with GPs (cf. Sec. 4.2), BAL (cf. Sec. 4.3), and parallel computation (cf. Appendix C).[26]

The algorithm that we propose for computing the optimal decision rules in dynamic incentive problems is summarized in Alg. 1, and proceeds as follows: We instantiate the value function iteration by drawing for every discrete state present in the model (that is, every type θ) a uniform sample of M points $\mathbf{X}_\theta = \{\hat{v}_\theta^{(1)}, \ldots, \hat{v}_\theta^{(M)}\} \subset \mathcal{B}$, the hypercube of promise utilities (cf. Eq. (30)),[27] at each of which we provide an initial guess K_θ^{init}. Jointly, they represent the initial training set $\mathcal{D}_\theta^{init} = \{(\hat{v}_\theta^{(1)}, K_\theta^{1,init}), \ldots, (\hat{v}_\theta^{(M)}, K_\theta^{M,init})\}$, to which we fit a GP per type θ. Having the initial setup ready, the value function iteration algorithm we propose is detailed in Alg. 1: In every iteration step of the value function iteration, we fit a GP $\mathcal{G}_\theta$ to the training set $\mathcal{D}_\theta$. This step is embarrassingly parallelizable and thus can be substantially accelerated by contemporary hardware, a fact that we leverage in our implementation. For more details on the parallelization scheme we employ, see Appendix C. Next, we perform BAL on the points that were deemed feasible to obtain a list of s candidate points per type, out of which the one with the best score is added to the training set. This step is an essential component in alleviating the curse of

[26]The value function iteration algorithm proposed in this section is loosely related to the one proposed by Keane and Wolpin [1994], who use Monte Carlo simulations and linear regression to solve and estimate discrete choice dynamic programming problems, a setting that substantially differs from the one we are targeting here.

[27]We use $\hat{v}_\theta^{(i)}$ as a shorthand notation for $(\hat{v}^{(i)}, \theta)$, that is, the location at which the Bellman equation is evaluated.

Algorithm 1: Value Function Iteration with GPs and BAL.

```
Data: For all $\theta$, draw a (small) initial set of $M$ points $\mathbf{X}_\theta = \{\hat{v}_\theta^{(1)}, \ldots, \hat{v}_\theta^{(M)}\} \subset \mathcal{B}$
        such that it contains the feasible set $\mathcal{V}$. Initial guess for value function $K_\theta^{\text{init}}$.
        Approximation accuracy $\bar{\epsilon}$.
Result: For each $\theta$: GPs over $\mathcal{B}$ for the value functions $K_\theta^*$ and the $\kappa$ policy
        functions $\chi_\theta^*$ per type.
For each type $\theta$: Generate initial training set
 $\mathcal{D}_\theta^{init} = \{(\hat{v}_\theta^{(1)}, K_\theta^{1,init}), \ldots, (\hat{v}_\theta^{(m)}, K_\theta^{m,init})\}$.
For each type $\theta$: $\mathcal{G}_\theta$ over $\mathcal{D}_\theta^{init}$.
Set iteration step $j = 1$.
while $\epsilon > \bar{\epsilon}$ do
    for $\theta \in \Theta$ do
        Set $G_{\theta,j-1} = G_\theta$.
        Use the predictive mean $\tilde{\mu}_{\theta,j-1}$ of $\mathcal{G}_{\theta,j-1}$ as interpolator of $K_{\theta,j-1}$.
        Use the predictive variance predictive $\tilde{\sigma}_{\theta,j-1}$ of $\mathcal{G}_\theta, j-1$ as point-wise
          measure for the uncertainty around $K_{\theta,j-1}$.
        Given $\tilde{\mu}_{\theta,,j-1}$ and $\tilde{\sigma}_{\theta,j-1}$, perform BAL on the approximate feasible points
          contained in $\mathbf{X}_\theta$ to obtain a list of $s$ candidate points $y^{(1)}, \ldots, y^{(s)}$.
        Add the point $y^{(k)}$ with the largest score (cf. Eq. (27)) to $\mathbf{X}_\theta$.
        $M = M + 1$.
        for $\hat{v}_\theta^{(i)} \in \mathbf{X}_\theta$ do
            Solve the optimization problem given by the relevant Bellman equation
              (see, e.g., Eqs. (31) and (34)) at the coordinate $\hat{v}_\theta^{(i)}$ to obtain $K_\theta^j$ (and the
              respective policies if needed), and where the interpolation on the
              right-side of the Bellman operator is performed by using the
              predictive mean from the various GPs from the iteration $j-1$.
        end
        Define the training set at iteration $j$: $\mathcal{D}_{\theta,j} = \{(\hat{v}_\theta^{(1)}, K_\theta^{1,j}), \ldots, (\hat{v}_\theta^{(1)}, K_\theta^{m,j})\}$.
        Fit a GP $\mathcal{G}_\theta$ over $\mathcal{D}_{\theta,j}$.
        Compute an error measure, e.g.: $\epsilon_\theta = \|K_\theta^j - K_\theta^{j-1}\|_2$.
    end
    Set $j = j + 1$.
    $\epsilon = \max(\epsilon_{\theta_1}, \ldots, \epsilon_{\theta_D})$.
end
$\tau = j - 1$.
$K^* = \{K_{\theta_1}^\tau, \ldots, K_{\theta_m}^\tau\}$,        $\chi^* = \{\chi_{\theta_1,1}, \cdots, \chi_{\theta_1,\kappa}, \cdots, \chi_{\theta_m,1}, \cdots, \chi_{\theta_m,\kappa}\}$.
```

dimensionality, as it focuses the computational resources where needed most. One way to perform BAL in the presented context is to simulate the model for s steps within every value function iteration step j to generate candidate points–in analogy to the second error measure outlined in Sec. 4.4. This approach, however, comes at the extra computational cost that in every iteration step j, not only value but also policy functions need to be approximated via GPs. As outlined before, we use Eq. (27) to score the said candidate points. We then add one or multiple points with the highest score to the set of training points. Then, we solve the model-implied Bellman equations (see, e.g., Eqs. (31)and (34)) at each of the training points to obtain new training targets $\{K^1_\theta, \dots, K^M_\theta\}$, over which we fit GPs.[28] This process is repeated until numerical convergence at some iteration step τ is reached (cf. Eq. (18)). Once the value function iteration has been terminated, we also evaluate our second error measure given by expression (33). If the error along the simulated path is also low, and all points along the simulation are deemed feasible, we have a numerical solution of acceptable accuracy for the approximate value functions $K^* = \{K^\tau_{\theta_1}, ..., K^\tau_{\theta_m}\}$, and the corresponding κ policy functions per state, that is, $\chi^* = \{\chi_{\theta_1,1}, \cdots, \chi_{\theta_1,\kappa}, \cdots, \chi_{\theta_m,1}, \cdots, \chi_{\theta_m,\kappa}\}$, all given by the predictive mean of an individual GP. If, however, this second error is still too large, or some of the points simulated are deemed infeasible, we re-launch the value function iteration until the second error criterion is also satisfied. For further implementation details, we refer the reader to the code hosted under the following URL: `https://github.com/GaussianProcessesForDynamicEcon/DynamicIncentiveProblems`.

Finally, note that one of the defining features of GPR is that it is a grid-free method for constructing a function approximator, that is, it allows the modeler to steer the content of the training set $\mathcal{D}$ closely and thus to construct interpolators on irregularly shaped geometries. This has two significant practical advantages when addressing dynamic incentive models numerically. First, suppose an individual optimization problem at some particular point x_i does not converge. In that case, one does not need to deal with tuning the optimizer until it converges at this location of the state space. Instead, this training input (and the corresponding nonsensical training target) can be discarded, that is, it is not added to the training set. This is in stark contrast to grid-based methods such as Smolyak [see, e.g., Krueger and Kubler, 2004a, Judd et al., 2014] or adaptive sparse grids [see, e.g., Brumm and Scheidegger, 2017, Brumm et al., 2021], where the construction of the interpolator breaks down if some of the optimization problems required by the algorithm cannot be solved. Second, computing solutions solely on a domain of relevance allows one to carry out value function iteration on complex, high-dimensional geometries without suffering from massive inefficiencies, as the computational resources are concentrated where needed and not wasted in parts of the state space where the quality of the GP interpolant is already high. Particularly in high-dimensional settings, this can substantially speed up the time-to-solution process, as the feasible set might have a negligibly small volume

[28]In our numerical experiments below (see Secs. 4.6 and 5), at every single training point, the individual optimization problems are solved either with `Ipopt` [see, e.g., Waechter and Biegler, 2006] (`http://www.coin-or.org/Ipopt/`) or the constrained optimization toolbox by Scipy (`https://docs.scipy.org/doc/scipy/reference/optimize.html`. Note that both solvers yield the same results up to numerical precision, providing an additional sanity check for our numerical solutions.

compared to the computational domain that standard approximation methods require.

4.6 Verification of the Solution Algorithm

To demonstrate the scalability and versatility of the computational framework introduced in this paper in the simplest possible fashion, we now solve the dynamic adverse selection problem as presented in Sec. 3.1. We use the original model calibration by Fernandes and Phelan [2000] as a basic verification test for our method, as for their two-dimensional baseline setting, the approximate numerical solutions are known, and the results can be visualized. We start by briefly summarizing the baseline model and its parameterization. Second, we report on the performance of our proposed dynamic programming method. Finally, we briefly discuss the numerical solution to this dynamic incentive problem.

The environment by Fernandes and Phelan [2000] consists of a risk-neutral principal that *minimizes* her cost, and a risk-averse agent. We choose the agent's utility over consumption c to be

$$U(c) = \sqrt{c}, \tag{35}$$

where c is restricted to be on a compact interval. There are two types in the model, that is, a "low" (state 1; also denoted as L) and a "high" (state 2; also denoted as H) type. The respective endowments are given by h_L and h_H. The agent "learns" his private type in each period and then reports it to the principal. Subsequently, the principal transfers consumption to the agent dependent on what the agent reported. Since the problem depends on the full history of reports, we use the recursive reformulation that we introduced (cf. Sec. 4.4 and Eq. (31)) to avoid the previously mentioned numerical difficulties. Furthermore, we follow Fernandes and Phelan [2000] and assume that the agent cannot claim to be a higher type than he is.

Since this model consists of two persistent types, the resulting state space of promised utilities is 2-dimensional. The Markov process governing the endowments is chosen such that the agent has a 90 percent chance of receiving the endowment he received in the previous period. This yields the following transition probabilities across the different types:

$$\Pi = \begin{bmatrix} \pi(L|L) & \pi(H|L) \\ \pi(L|H) & \pi(H|H) \end{bmatrix} = \begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix}. \tag{36}$$

The remaining model parameterization is reported in Tab. 1.

To solve this benchmark model, we start the dynamic programming algorithm 1 by uniformly drawing 100 samples for each type θ from the domain $\mathcal{B} = [0, 10]^2$. Next, we provide an initial guess for the value function at each of the sample points, that is,

$$K_\theta^{init} = \pi(L|\theta)(h_L - U^{-1}(\hat{v}_\theta(L)(1-\beta))) + \pi(H|\theta)(h_H - U^{-1}(\hat{v}_\theta(H)(1-\beta)))/(1-\beta), \tag{37}$$

Parameter	Value
β	0.9
h_H	0.35
h_L	0.1
$[\underline{c}, \overline{c}]$	$[0,1]$
$\mathcal{B}$	$[0,10]^2$

Table 1: Parameterization of the privately observed endowment model by Fernandes and Phelan [2000].

Error type	L_2 [%]	L_∞ [%]
global error	$1.7 \cdot 10^{-6}$	$8.2 \cdot 10^{-6}$
error along a simulated path	$2.8 \cdot 10^{-8}$	$2.6 \cdot 10^{-6}$

Table 2: Average and maximum percentage errors at convergence. The first error type, 'global error,' was computed by evaluating Eq. (33) at uniformly drawing 1, 000 samples on $\mathcal{B}$, whereas 'error along a simulated path' was computed by generating the same number of observations along a simulated path and that was launched at the point in the state space given by $\arg\max_{\hat{v}(\theta),\theta} K(\hat{v}(\theta), \theta)$.

where $\theta \in \{L, H\}$, and over which we fit GPs–using piece-wise polynomial kernels with $q = 1$–over the initial training set. Thereafter, we iterate until we reach a 'global error' smaller than $\epsilon = 1 \cdot 10^{-4}$% in the L_2-norm (cf. Tab. 2). At this level of accuracy, the value function and policies do not change anymore, even if the error is further decreased. To solve this model, we perform BAL by generating candidate points along a simulation path of 2, 000 steps that is started from the maximizer. Since the procedure is numerically somewhat expensive, as was pointed out in Sec. 4.5, we add one additional point per type to the training set after every five value function iteration steps, resulting in sets $\mathcal{D}_\theta$ containing 205 samples per type at convergence. Fig. 5 depicts how BAL sequentially adds samples to the training set where they matter the most, that is, in a subset of the feasible region where a typical simulation of the model tends to go. This figure makes it clear that BAL is a crucial ingredient for the overall efficiency of our algorithm, as it helps to alleviate the curse of dimensionality by focusing the GP approximation on the equilibrium path, which is a portion of the state space that is much smaller than the entire feasible set (cf. Fig. 6 below). Note that one iteration step takes about 125 seconds to execute when running on an ordinary desktop, using 12 MPI processes of an AMD Ryzen 9 processor. However, it could be reduced by at least another order of magnitude by simply using proportionally more CPUs on a compute cluster (see Appendix C for more details regarding the parallelization).

Once the value function iteration has reached its termination criterion, we evaluate the error measure (33) along a simulation path, which shows very low values of $L_2 = 2.8 \cdot 10^{-8}$ % and $L_\infty = 2.6 \cdot 10^{-6}$%, respectively (see Tab. 2). Comparing the solutions found here to

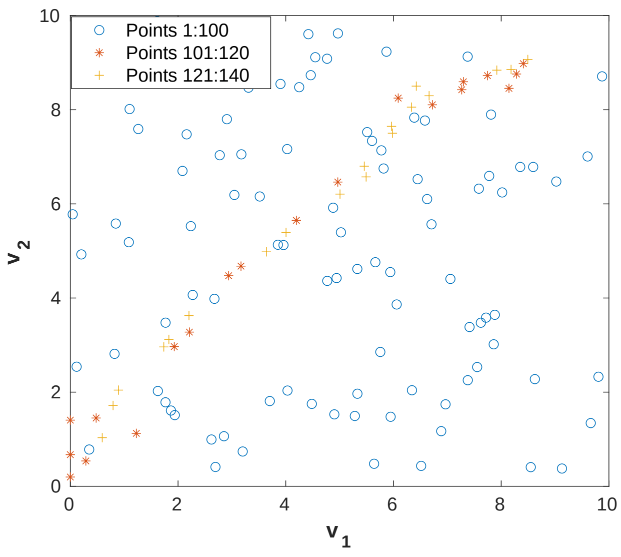

Figure 5: This figure displays the initial set $\mathbf{X}_{h_L}$ of 100 points (blue dots), uniformly drawn from $\mathcal{B}$, which then are systematically enhanced via BAL. The first 20 points being added to this set are depicted as red stars, whereas the subsequently added 20 points are labeled as a yellow plus. Notice that the points are added where needed most, that is, in the region where the simulation of the model tends to move along.

model solutions with errors that are about an order of magnitude worse no longer showed a change in the value and policy functions. Thus, we conclude that the value function iteration has converged.

In the left panel of Fig. 6, we depict the value function of state 1, the 'low' type. To do so, we uniformly draw 3,000 points from $\mathcal{B}$ at which we evaluate the corresponding predictive mean of the GP. In the example here, we distinguish the points that are approximately feasible or infeasible by looking at a lower bound of the value function, $K_{min} = (h_L - \bar{c})/(1-\beta)$, that is, the minimal payout minus maximal cost in perpetuity. The samples below this cutoff are labeled as infeasible.

The right panel of Fig. 6 displays a projection of the approximate feasible set for the 'low' type. Notice that our numerical characterization of the approximate feasible set may not be seen as highly accurate as for instance in Fernandes and Phelan [2000]. This has merely to do with the fact that we draw a large but finite sample, on which we then evaluate and visualize a GP. If the sample chosen was much larger, the boundaries between feasible and infeasible would be much more distinct. However, in the applications at hand, determining the exact boundary is not of key relevance. What matters in the application at hand is that the approximate feasible set contains the true feasible set and that the approximate value and policy functions are of high quality where the model 'lives,' that is, along the simulated path (cf. Fernandes and Phelan [2000, p. 236]).

In Fig. 7, we depict a simulation of the optimal policy. The latter was launched at the point of the state space with the highest value function value and then was evolved

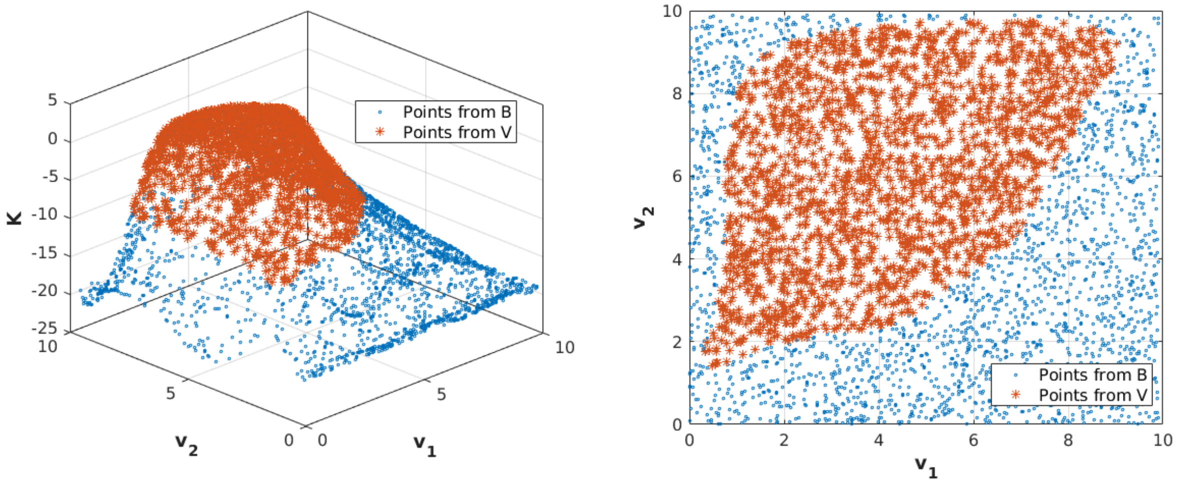

Figure 6: The left panel depicts the value function as a function of $\hat{v}(L)$ and $\hat{v}(H)$ (denoted as $\mathbf{v_1}$ and $\mathbf{v_2}$) for the low type. The (orange) stars depict the draws for which the value function is feasible, that is, from $\mathcal{V}$, whereas the (blue) dots represent points from $\mathcal{B}$ that were deemed infeasible. Recall that the state variables consist of the promise utilities. Thus, one cannot simply assume a certain shape for the resulting value function. Furthermore, since we have a continuous relaxation of the problem, we know that once we approach the boundary of the feasible set, the value function will decrease, as can be seen in the figure. The right panel displays–in the same color-coding as before–the approximate feasible set at convergence as a function of the promise utilities projected to the low state.

forward by sequentially drawing 1,000 random shocks that follow the distribution given by Π (cf. Eq. (36)). In the left panel of Fig. 7, we display the simulation trajectory within the feasible set for the low type, starting in the lower left and then moving quickly up to the top right of the feasible set. This figure demonstrates that the remainder of state space is irrelevant in this model setting, as the optimal policy does not traverse it, thereby clearly demonstrating that the BAL indeed puts the high resolution where needed most (cf. Fig. 5). On the right panel of Fig. 7 we show a simulation of the utility promise to the truth-telling agent. The (blue) line represents the utility promise, whereas the (black) dots indicate the type in a given simulation step. The 0 represents the low state L, whereas 1 denotes the high state H. The (blue) line suggests that the agent accumulates a utility promise by sacrificing the current utility for a higher baseline utility in the long run. Concretely, the agent starts with a low utility promise, which then rises for the first 17 periods of the simulation. After that, it oscillates conditional on whether the agent is in the high or low state, thereby receiving a higher payout in the high than in the low state. As a result, the payout is higher than what he could have achieved on his own, that is, it is higher than h_L and h_H, respectively. Loosely speaking, the contract behaves like a state-dependent retirement account, where we first accumulate wealth and then live off of the returns.

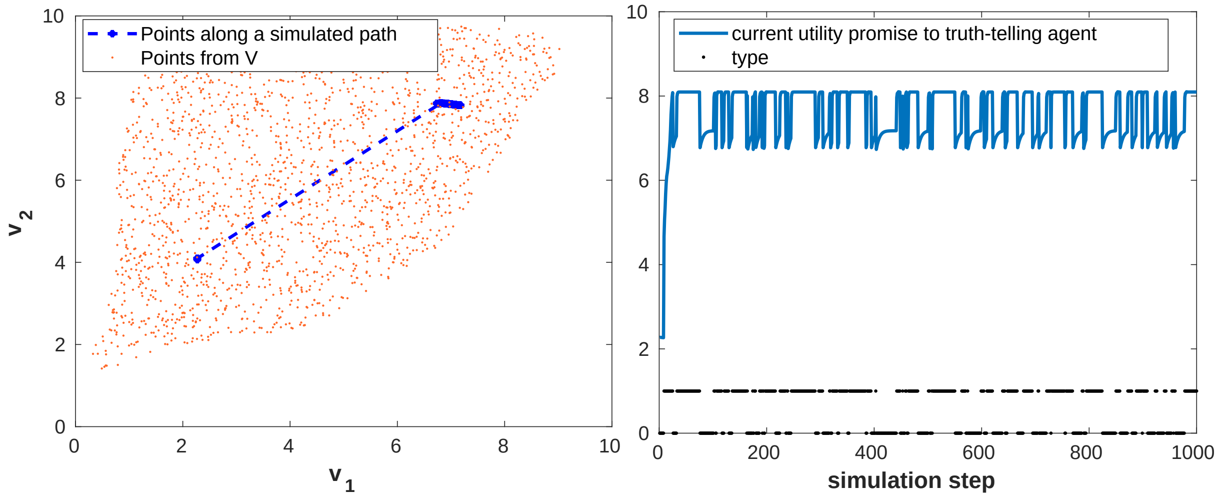

Figure 7: The left panel depicts the path of a simulation (dashed-starred blue line) of the optimal policy as a function of $\hat{v}(L)$ and $\hat{v}(H)$ (denoted as $\mathbf{v_1}$ and $\mathbf{v_2}$) for the low type within the approximate feasible set $\mathcal{V}$. The right panel shows the promise to the truth-telling agent as a function of the simulation step t and its respective current type. The low type is encoded by a value 0, whereas the high type is represented by a 1 on the y-axis.

Comparing our results to those reported by Fernandes and Phelan [2000], we can (by visual inspection) state that we replicate their findings.[29] In summary, our sequence of figures and performance numbers verify that our proposed dynamic programming method can successfully solve dynamic incentive problems. Thus, we are now prepared to address in Sec. 5 dynamic incentive models with heterogeneous agents.

5 A Dynamic Incentive Model with Heterogeneous Agents

To study how heterogeneity in risk across agents affects adverse selection in insurance markets, we present now the solutions to a stylized heterogeneous agents model with multiple persistent types (cf. Sec. 3.2). We start by summarizing in Sec. 5.1 its parameterization and the convergence of the numerical solution. After that, we discuss in Sec. 5.2 the results and their implications.

5.1 Parameterization and Numerical Convergence

We consider again a discrete-time economy that consists of a risk-neutral principal that *minimizes* her cost, and $n = 2$ risk-averse agents of $m_i = 2$ different, persistent shocks per agent i. Consequently, the resulting state space of promised utilities is four-dimensional

[29]Note that Fernandes and Phelan [2000] do not report any convergence numbers, policies, or run-times. Thus, we cannot perform a more detailed comparison beyond a visual inspection of the value functions and approximate feasible sets.

(cf. Sec. 3.2). The agents are further differentiated by their risk profile. For both agents, we choose the utility over consumption c_i to be

$$U_i(c_i) = \sqrt{c_i}, \tag{38}$$

where c_i is restricted to the compact interval $[0, 1]$. On an individual level, both agents are exposed to a "low" shock (state 1; also denoted as L), and a "high" shock (state 2; also denoted as H). The corresponding endowments are the same for both agents, and are denoted by h_L and h_H (see Tab. 3 for their numerical values). As above in the single-agent setting (cf. Sec. 4.6), the agents "learn" their private type in each period, and thereafter report it to the principal. Subsequently, she transfers consumption to every agent conditional on what the particular agent reported. Since this problem depends on the full history of reports, we need to use the recursive reformulation that was introduced before (cf. Sec. 4.4) to avoid the previously mentioned numerical difficulties. Furthermore, we entertain again the assumption by Fernandes and Phelan [2000] that agents cannot claim to be a higher type than they are.

Heterogeneity across the two agents is introduced by equipping them with different risk profiles. We model the first agent as the one in the single-agent benchmark (cf. Sec. 4.6). In contrast, we choose the parameterization for the second agent in such a way that he has a lower risk of dropping from the high state H into the low state L compared to the first agent. The Markov process governing the endowments of the first agent is given by the following transition probabilities across the different types:

$$\mathbf{\Pi^{(1)}} = \begin{bmatrix} \pi^{(1)}(L|L) & \pi^{(1)}(H|L) \\ \pi^{(1)}(L|H) & \pi^{(1)}(H|H) \end{bmatrix} = \begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix}, \tag{39}$$

whereas the Markov chain describing the second agent is chosen as

$$\mathbf{\Pi^{(2)}} = \begin{bmatrix} \pi^{(2)}(L|L) & \pi^{(2)}(H|L) \\ \pi^{(2)}(L|H) & \pi^{(2)}(H|H) \end{bmatrix} = \begin{bmatrix} 0.9 & 0.1 \\ 0.0667 & 0.9333 \end{bmatrix}. \tag{40}$$

It is now straightforward to generalize those two components into a stylized heterogeneous agents setting by assembling the transition matrices $\mathbf{\Pi^{(i)}}$ from the individual agents into a single one by following Eq. (14), that is,

$$\mathbf{\Pi} = \begin{bmatrix} \mathbf{\Pi^{(1)}} & \mathbf{0} \\ \mathbf{0} & \mathbf{\Pi^{(2)}} \end{bmatrix}, \tag{41}$$

where we assume to model two distinct agents, and where a transition from one to the other is not possible. Recall that the stationary distribution of a Markov chain with a transition matrix P is given by some vector, $\hat{\chi}$, such that $\hat{\chi}P = \hat{\chi}$. In other words, over the long run, no matter what the starting state was, the proportion of time the chain spends in state k is $\hat{\chi}_k$ for all k. In a two-dimensional problem, the stationary distribution $\hat{\chi}$ for a stochastic matrix $P = (P_{ij})$ (such as $\mathbf{\Pi^{(1)}}$ and $\mathbf{\Pi^{(2)}}$) is given by $(P_{21}/(P_{12} + P_{21}), 1 - P_{21}/(P_{12} + P_{21}))$.

Parameter	Value
β	0.9
h_H	0.35
h_L	0.1
$[\underline{c}, \overline{c}]$	$[0, 1]$
$\mathcal{B}$	$[0, 10]^4$

Table 3: Parameterization of the heterogeneous agent model.

Thus, it follows that agent 1 spends 50% of the time in either of the two states, whereas agent 2 is 40% of the time in the low state, and 60% in the high state. Furthermore, starting from a high state, agent 2 has a 33% lower risk of dropping into the low state compared to agent 1. The rationale behind choosing this setup is that we intend to model a marked, but not too significant difference, between the two agents. If, by contrast, the risk profile across the two types would differ significantly, such information might be public in some secondary metric, such as in the case when one compares insuring young and old agents, thus rendering the "hidden information" modeling approach invalid. The remaining parameterization of the model is summarized in Tab. 3.

We next proceed to solve the heterogeneous agent model by using algorithm 1. As for the two-dimensional benchmark model (cf. Sec. 4.6), we need to provide an initial guess for the value function at each of the four-dimensional sample points $\hat{v} = (\hat{v}^1_\theta, \hat{v}^2_\theta)$, that is,

$$K^{i,init}_\theta = (\pi^{(i)}(L|\theta)(h_L - U^{-1}(\hat{v}^i_\theta(L)(1-\beta))) + \pi^{(i)}(H|\theta)(h_H - U^{-1}(\hat{v}^i_\theta(H)(1-\beta))))/(1-\beta), \quad (42)$$

where $\theta \in \{L, H\}$. We start the value function iteration by drawing 128 points for each type and each agent from the domain $\mathcal{B} = [0, 10]^4$. We iterate until we reach a 'global error' smaller than $\epsilon = 1 \cdot 10^{-3}\%$ in the L_2-norm (cf. Tab. 4). Our numerical experiments show that in the heterogeneous agents setting too, at this level of accuracy, the value function and policies do not change anymore, even if the error is further decreased. During the value function iteration procedure, we perform BAL by again generating candidate points along a simulation path of $2,000$ steps every five iteration steps until the training sets $\mathcal{D}^i_\theta$ contain about 370 samples per agent and type. To approximate the value and policy functions, we use a piece-wise polynomial kernel with $q = 0$.

Solving this four-dimensional model requires substantially more computing time than the two-dimensional benchmark model studied in Sec. 4.6. Using two compute nodes with Intel Ice Lake processors that are each equipped with 64 cores and that are installed at the research computing cluster of Lancaster University, our numerical experiments show that at the beginning of the value function procedure, one iteration step consumes less than 2 minutes. In contrast, at convergence, when many additional samples have been added to the training set via BAL, an iteration step takes around 8 minutes. The increase in compute time compared to the baseline model discussed in Sec. 4.6 is a consequence of the fact that the optimizer requires more time to solve the individual problems (cf.

Error type	L_2 [%]	L_∞ [%]
global error	$7.7 \cdot 10^{-4}$	$4.0 \cdot 10^{-3}$
error along a simulated path	$7.4 \cdot 10^{-8}$	$5.5 \cdot 10^{-6}$

Table 4: Average and maximum percentage errors at convergence. The first error type, the 'global error,' was computed by evaluating Eq. (33) at uniformly drawn 1, 000 samples on $\mathcal{B}$, whereas the 'error along a simulated path' was computed by generating the same number of observations along a simulated path for each agent. The simulation was started at the maximizer of the associated value function.

maximal payoff to principal	Low type L	High type H
One agent		
high-risk agent	1.193	1.616
low-risk agent	1.199	1.774
Two agents		
high-risk agent	1.161	1.607
low-risk agent	1.193	1.774

Table 5: Optimal payoff to the principal for all model combinations.

Eq. (34)) than in the simpler model. In addition, the time-to-solution suffers from the cubic algorithmic complexity $\mathcal{O}(N^3)$ of standard GPs with respect to the sample size N: Since every $\mathcal{D}_\theta^i$ consists of about twice as many data points as in the benchmark model, the runtime increases by about an additional order of magnitude per value function iteration step. However, the runtime could again substantially be reduced by simply using more CPUs (cf. Appendix C), and also by applying scalable GPs (cf. Appendix B, and references therein).

Table 4 reports the errors once the value function iteration has converged. As before, we find very low values for the different error measures such as $L_2 = 7.4 \cdot 10^{-8}\%$ and $L_\infty = 5.5 \cdot 10^{-6}\%$ along a simulated path. Thus, reaching such low errors allows us next to study the implications of the model solutions.

5.2 Results

The main issue of hidden heterogeneity in insurance markets, such as unknown risk profiles, is that it can impede the proper functioning of markets by imposing significant costs on the insurers, rendering it unprofitable to offer insurance [Akerlof, 1970]. Thus, to obtain a notion of how additional hidden information affects the payoff to the principal, one needs to find the optimal value for the contract in the case of a model that consists of multiple agents with different risks, and compare them to the situation where one studies these agents in isolation. Formally, the payoff is specified by the quantity $K\left(\tilde{v}\left(\theta\right), \theta\right)$, where $\theta \in \{L, H\}$, and $\tilde{v}(\theta) \in \arg\max_v K\left(v, \theta\right)$. Monitoring this metric provides a first hint on how costly adverse selection is, thereby allowing us to answer questions such as

	Mean payoff
One agent	
high-risk agent	-3.12
low-risk agent	-1.67
Two agents	
high-risk agent	-6.77
low-risk agent	-4.55

Table 6: Average payoff to the principal for all model combinations along a simulation path of length $1,000$. All the simulations were launched at the maximizer of the associated value function.

whether adding high-risk individuals to the pool of customers decreases payoffs for the insurance company, that is, the principal.

To establish a baseline, we first consider the payoffs from the two-dimensional benchmark models of the low and high-risk agent we introduced in Sec. 5.1 separately. The results are summarized in the top two rows of Tab. 5 and are denoted as "One agent". As shown there, the payoffs to the insurance for solely insuring the high-risk agent are 1.193 and 1.616 when we assume that the agent is either in the low state L or the high state H, respectively. In contrast, the payoffs to the principal for solely insuring the low-risk agent are 1.199 and 1.774 if we assume again that the agent is in state L or H, respectively.

Next, we consider the heterogeneous agents model, where both the low and high-risk agents are present simultaneously. In this setting, the principal offers four contracts; two for the high-risk agent and two for the low-risk agent. These contracts are conditional on the state the agent is currently in. The computed payoffs are summarized in the lower two rows of Tab. 5 and are denoted as "Two agents". In this joint problem, the principal's optimal values are 1.161 and 1.607 for the high-risk agent, and 1.193 and 1.774 for the low-risk agent. We can see that the payoffs in the model with heterogeneous agents are up to 2.5% lower than in the benchmark case. This finding is unsurprising since, in the heterogeneous agents model, the principal has to incentivize the agents to report truthfully for four possible reports compared to two in the benchmark case. This result is due to the fact that there are more constraints in the heterogeneous agent model. Consequently, the contract's payoff should be less or equal than in the model with solely one agent.

To complement this first finding on how heterogeneity in risk affects adverse selection, we provide several results from simulations, as they help us describe the long-run behavior of the contract. Tab. 6 reports the principal's mean payoff along a simulation for the two different single-agent benchmark models (denoted as "One agent"), as well as the heterogeneous agents model (denoted as "Two agents"). This table shows that for the heterogeneous agents' case, the average payoff is -6.77 and -4.77 for high and low-risk agents, whereas for the single agent models, those values yield -3.12 and -1.67, respectively. We can see that the average payoff to the principal along a simulation is markedly lower in the heterogeneous agents' case compared to the single-agent models.

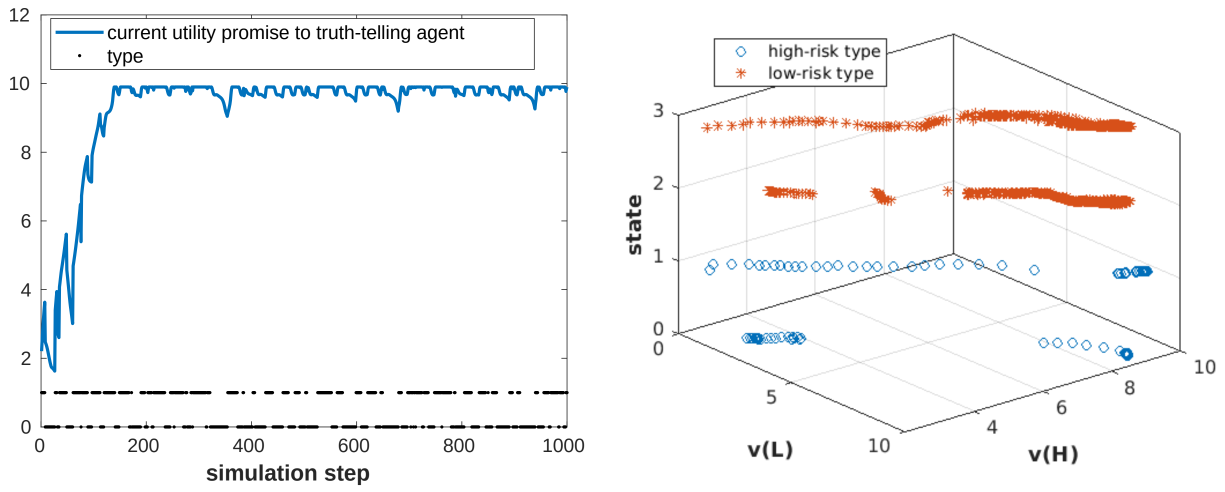

Figure 8: The left panel shows the promise to the high-risk, truth-telling agent as a function of the simulation step t and his respective current type. The low type is encoded by 0, whereas the high type is represented by a 1 on the y-axis. The right panel displays projections of utility promises along 1,000 simulation steps for the agent with the high-risk type (blue dots) and the low-risk type (orange stars). The z-axis, labeled as "state", indicates the discrete state the simulation is in. The numbers 0 and 1 encode the L and H state for the high-risk agent, whereas the numbers 2 and 3 represent the low-risk agent. $v(L)$ and $v(H)$ encode the promise to the agent when he previously reported L or H, respectively.

To further investigate the discrepancy between the findings reported in Tab. 5 and Tab. 6, we take a closer look at the simulated behavior of the contract. The left panel of Fig. 8 displays the utility promise to the truth-telling high-risk agent along a simulation of 1,000 steps length, and the corresponding type he is in. We can see that for the first 130 simulation steps, the utility promises steadily increase to a maximum level of just below 10. After these 130 periods, the contract behavior becomes almost static, as can be seen by the fact that it alternates between a narrow range of values, depending on the current reported shock. Comparing these simulation results to those from the benchmark model (cf. the right panel of Fig. 7), we can see that the promises to the truth-telling agent are higher in the heterogeneous agents model than in the original problem, which implies that the payoff of the principal is lower. The latter finding is also reflected by the values reported in Tab. 6.

The right panel of Fig. 8 depicts projections of utility promises for two simulations, one for each risk type, along 1,000 steps. The z-axis encodes the type the simulation was in. The values 0 and 1 represent the low and high state for the high-risk agent, whereas the levels 2 and 3 represent the low and high type of the low-risk agent, respectively. Furthermore, the $v(L)$-axis and the $v(H)$-axis represent the utility promises to the respective agent if he reported L or H. The overall behavior we observe here for the heterogeneous agents case resembles that previously observed for the one-agent benchmark model (cf.

the left panel of Fig. 7): The simulation starts somewhere in the lower left corner of the set of promised utilities, but then quickly ends up in a relatively focused region in the top right corner in the space of utility promises.

How do the differences between the simulations of the baseline model and those presented in this section mesh with the fact that the value of the contract for the principal is almost identical, no matter whether we consider a setting with one or two agents? This question can be answered by the fact that in the particular model setting we are using (cf. Fernandes and Phelan [2000]), the relatively small discount factor $\beta = 0.9$ eliminates the influence of the long-run behavior on the contract at time zero, and therefore enables the principal to postpone the payments to the agents into the future. To put a concrete number on this problem: at a simulation step 100, the payoff is discounted with a value as small as $\beta^{100} \approx 2.7 \cdot 10^{-5}$ at time zero.

In summary, the numerical results of our stylized dynamic adverse selection model with heterogeneous agents and persistent shocks provide two main insights. First, our findings imply that it is unsurprising that observing adverse selection in real data is a difficult task (see, e.g., Spinnewijn [2017], and references therein). For the infinite horizon model under consideration, the contracting behavior changes during its lifetime (cf. the left panel of Fig. 8). In particular, the price of adverse selection only shows up at certain times in the contract. Therefore, it is unclear how such behavior could be extracted from real data, as every insurance company typically has customers at different stages of their contract. Second, one of the central arguments for government intervention in insurance markets is that heterogeneity in risk causes a market failure due to its impact on the profitability of the insurance contract [Akerlof, 1970]. In contrast, our findings suggest that this impact is too small to affect profitability to a significant extent, thereby confirming the findings of the empirical literature, which failed to find adverse selection in several markets (see, e.g., Cardon and Hendel [2001]).

Finally, note that in order to quantify our findings more precisely, a carefully calibrated model would be required. However, this is beyond the scope of this article. The main objective of the work presented here is to lay the foundations for making dynamic adverse selection models with heterogeneous agents and persistent shocks numerically tractable.

6 Conclusion

This article makes a series of contributions that enable the study of dynamic incentive models with heterogeneous agents and persistent shocks, that is, for high-dimensional models where the feasible set is not known ex ante. First, we introduce a penalty-based reformulation of dynamic incentive problems that is numerically easier to handle than the standard recursive formulation commonly used in the literature, as it allows us to bypass the explicit computation of the feasible set. Second, we provide formal proof that this reformulation converges to the same solution as the original model. Third, we propose a scalable and flexible value function iteration algorithm that combines Gaussian process regression with Bayesian active learning for solving a wide range of high-dimensional dynamic programming problems, including–but not limited–to those studied in this paper. Fourth, to demonstrate the capabilities of our framework, we apply it to the dynamic adverse selection model by Fernandes and Phelan [2000] as a verification test for our method, as the approximate numerical solutions are known for their two-dimensional baseline setting, and the results can be compared. Furthermore, since the models we study no longer have analytical solutions, but have to be determined iteratively, the final ingredients for our framework are measures to assess the credibility and correctness of our computational results. To do so, we follow the best practices in a sub-field of computational sciences called *validate, verify, and uncertainty quantification*, and propose to use two particular error criteria jointly. Fifth, we use our modeling framework to study to what extent risk heterogeneity causes adverse selection in insurance markets, a question where the theoretical and empirical literature are at odds, and that could not be answered using existing techniques. We find that considering multiple agents with a hidden risk profile only has a minor effect on the overall value of the contract. This observation suggests that, at least in our model setting, heterogeneity in risks fails to explain adverse selection, thereby confirming the findings of the empirical literature [Cardon and Hendel, 2001] in a numerical setting. Our results thus imply that the scope for government intervention due to adverse selection is limited by the considered contracting environment and that care has to be taken when extrapolating findings from simplified models. In addition, all our methodological developments are supplemented by a Python-based toolbox that can be found under the following URL: `https://github.com/GaussianProcessesForDynamicEcon/DynamicIncentiveProblems`.

In summary, this all suggests that our proposed framework will enable researchers to study dynamic incentive problems of a greater richness than was possible prior to this work, as they no longer need to drastically restrict their modeling choices from the outset.

Moreover, we emphasize that while the focus of the work presented in this paper lies in solving dynamic adverse selection problems with discrete, persistent shocks, the methods proposed here, being generic, have a far broader scope: They can prove useful in solving models where one or several of the following features occur: the state space is endogenous, of irregular geometry, or multi-dimensional. These situations can arise in problems such as:

- Dynamic adverse selection with a continuum of hidden states. If one applies the first-order approach to tackle such models, it becomes necessary to track the utility promise and the marginal utility promise as part of the state space. Furthermore, not all promises are feasible.
- Dynamic moral hazard problems with multiple agents. Such models require keeping track of the utility promises to every agent. If the principal is constrained, for example, via a budget, then the state space becomes endogenous, as not all states will be affordable. That is, there are promises that cannot be paid with current cash on hand.
- Dynamic games. In these models, one needs to keep track of the history in order for the players to punish or reward the past behavior of their opponents. This history

dependence also has a recursive formulation using promise utilities as an endogenous state space.

- Problems with debt and endogenous default. These models typically require that the researcher endogenously determines the borrowing limits as part of the model.

A Machine Learning Glossary

In this appendix, we provide a short glossary of terms that we use in this paper that are common in the machine learning literature. In addition, we try to link this terminology to the terms commonly used in economics. For a more complete overview, see, for example, Goodfellow et al. [2016], Bishop [2006], and `https://developers.google.com/machine-learning/glossary`, as well as Igami [2020], who clarifies the connections between certain algorithms to develop artificial intelligence and the econometrics of dynamic structural models.

- **Machine learning:** Mitchell [1997] provides a succinct definition: *A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.* In simple language, machine learning is a field in which human-made algorithms (such as linear regression or GPR) have the ability to *learn* by themselves and make predictions for unseen data. In the context of our proposed algorithm, E corresponds to the training set $\mathcal{D}$ that contains solutions to the Bellman equation at various locations of the state space at a given iteration step j, whereas the term prediction in our context refers to interpolating the value and policy functions.
- **Model:** A machine learning model is a data structure that stores a representation of a dataset, for example, a GP and its hyperparameters that are used to fit a value function based on a collection of Bellman equations solved at various feasible locations in the state space in a given iteration step of the value function iteration algorithm.
- **Training set:** A set of observations used to generate machine learning models. In our concrete case, the training set contains a collection of solutions to the Bellman operator, solved at a particular point in the state space at each iteration step of the value function procedure.
- **Hyperparameters:** They are higher-level properties of a model, such as how fast it can learn, or the complexity of a model. In the context of GPs, the hyperparameters are, for instance, the characteristic lengthscale and variability defining the SE kernel (cf. Eq. (22)).
- **Training:** Training a model such as a GP means learning, that is, determining good values for all the hyperparameters, for example, via maximizing the likelihood.
- **Supervised machine learning:** Training a model using a *labeled* dataset. A label refers to an *answer* portion of an observation in supervised learning. For example, in our case, we have labels to classify observations into *feasible* and *infeasible* (cf. Sec. 3.1).
  - **Regression:** Predicting a continuous output. In application, prediction refers to interpolate value or policy functions.
  - **Classification:** Predicting a categorical output, such as *feasible* and *infeasible*.
- **Unsupervised learning:** Training a model to find patterns in an unlabeled dataset. An example of unsupervised machine learning popular in economics is principal component analysis (PCA).

- **Reinforcement learning:** A branch of machine learning that, among other things, is concerned with solving dynamic programming problems. Reinforcement learning is a data-driven approach to solving dynamic programming problems, where the data can be created artificially via simulations or by real-life observations. See Sutton and Barto [2018] for a thorough introduction.
- **Active learning:** A training approach in which the algorithm chooses some of the data it learns from. Active learning is particularly valuable when labeled examples are scarce or expensive to obtain, such as solving constrained optimization problems within every step of the value function iteration. Instead of blindly seeking a diverse range of labeled examples, an active learning algorithm selectively seeks the particular range of examples it needs for learning. In the algorithm we propose in this paper, we use BAL to enrich the training set so that the Bellman equations are solved at the locations of the state space where they improve the GP function approximation the most (cf. Sec. 4.3).

B Computational bottlenecks

Let N be the number of observations in the training set $\mathcal{D}$ (cf. Sec. 4.2). The computational cost of standard GPR is dominated by the need to perform a Cholesky decomposition of the $N \times N$ covariance matrix at each iteration step of Alg. 1, which scales as $O(N^3)$ [Rasmussen and Williams, 2005]. Thus, using standard implementations of GPs will cause problems when N is in the order of $10,000$. In such cases, one has to resort to fast GP approximations. In our applications, we use GPs based on Blackbox Matrix-Matrix multiplication (BBMM) [Gardner et al., 2018]. BBMM inference uses a modified batched version of the conjugate gradients algorithm to derive all terms for training and inference in a single call. BBMM reduces the asymptotic complexity of exact GP inference from $O(N^3)$ to $O(N^2)$. Moreover, there are other recent fast methods available such as KISS-GP [Wilson and Nickisch, 2015] or Deep Kernel Learning [Wilson et al., 2016] that allow GPs to scale up to millions of observations if one prescribes some structure in the prior covariance kernels. Note that we have implemented and tested those kernels in the Python code accompanying this paper. However, in the context of our models, we achieved the best overall performance with BBMM. For a more detailed discussion, see Murphy [2022, Ch. 18.5], and references therein.

C Parallelization

In order to solve "large" problems in a reasonably short time, we use parallel computation. There is one key location where the value function iteration algorithm described in Sec. 4.5 can trivially exploit the availability of parallel computing: the evaluation of the Bellman operator. At every iteration step j of the value function procedure, there are $m \cdot N$ *independent* optimization problems that need to be solved and are all independent from each other. Recall that m is the number of discrete states (resulting, for instance, from the number of persistent, discrete shocks in the model) in the model, and N is the number of observations per discrete state. Consequently, the generation of the training data for the GPs is embarrassingly parallel. We now outline the basic steps in our parallelization using the Message Passing Interface (MPI; see, e.g., Skjellum et al. [1999]). For simplicity, we assume that we have n_{cpu} computational cores available, which we refer to as *workers* or *processes* (that correspond to individual MPI processes) from this point on. At each iteration step j of the value function iteration algorithm, we broadcast the current value function to all processes such that every process can evaluate the Bellman operator independently. The communication cost required to perform this operation is negligibly small. Then, the collection of the N training inputs per discrete state θ (see Alg. 1) becomes embarrassingly parallelizable. In consequence, each worker simply evaluates the Bellman operator at a fraction of the test points, that is, and every worker is assigned with a fractional workload equal to solving $(m \cdot N)/n_{\text{cpu}}$ times. This is where most of the computational time is spent. Subsequently, all the workers gather the distributed data. This operation also has a negligible communication cost. Furthermore, the fitting of the GP hyperparameters is also parallelized across MPI workers, thereby also being moderately accelerated.

D Why use Gaussian processes?

Apart from GPs, also other methods of machine learning, such as deep neural networks (see, e.g., Goodfellow et al. [2016]) have also recently piqued the interest of computational economists to solve and estimate dynamic models (see, e.g. Azinovic et al. [2022], Villa and Valaitis [2018], Duarte [2018], Fernández-Villaverde et al. [2019], Maliar et al. [2021], Didisheim et al. [2020], Chen et al. [2021], Ebrahimi Kahou et al. [2021]. However, in our setting, we deem GPs the more suitable choice as, among many reasons, their expressiveness allows to obtain excellent predictions with considerably fewer observations. In addition, GPs provide an estimate of uncertainty or confidence in the predictions through the predictive variance. While the predictive mean is often used as the best guess of the output, that is, the interpolation value, the full distribution can be used in a meaningful way. For example, we can estimate a 95% confidence bound for the predictions, which can be used to measure control performance (cf. Sec. 4.3). Moreover, GPs allow to include prior knowledge of the system behavior by defining priors on the hyperparameters or by constructing a particular structure of the covariance function. This feature enables incorporating domain knowledge into the GP model to improve its accuracy. For more reasons on when to apply GPs instead of neural networks, see Murphy [2022, Ch. 18.1].

References

Arpad Abraham and Nicola Pavoni. Efficient allocations with moral hazard and hidden borrowing and lending: A recursive formulation. *Review of Economic Dynamics*, 11(4): 781 – 803, 2008. ISSN 1094-2025. doi: http://dx.doi.org/10.1016/j.red.2008.05.001. URL http://www.sciencedirect.com/science/article/pii/S1094202508000197.

Dilip Abreu and Yuliy Sannikov. An algorithm for two-player repeated games with perfect monitoring. *Theoretical Economics*, 9(2):313–338, 2014. ISSN 1555-7561. doi: 10.3982/TE1302. URL http://dx.doi.org/10.3982/TE1302.

Dilip Abreu, David Pearce, and Ennio Stacchetti. Optimal cartel equilibria with imperfect monitoring. *Journal of Economic Theory*, 39(1):251 – 269, 1986. ISSN 0022-0531. doi: http://dx.doi.org/10.1016/0022-0531(86)90028-1. URL http://www.sciencedirect.com/science/article/pii/0022053186900281.

Dilip Abreu, David Pearce, and Ennio Stacchetti. Toward a Theory of Discounted Repeated Games with Imperfect Monitoring. *Econometrica*, 58(5):1041–1063, September 1990. URL https://ideas.repec.org/a/ecm/emetrp/v58y1990i5p1041-63.html.

Dilip Abreu, Benjamin Brooks, and Yuliy Sannikov. Algorithms for stochastic games with perfect monitoring. *Econometrica*, 88(4):1661–1695, 2020. doi: https://doi.org/10.3982/ECTA14357. URL https://onlinelibrary.wiley.com/doi/abs/10.3982/ECTA14357.

George A. Akerlof. The Market for "Lemons": Quality Uncertainty and the Market Mechanism. *The Quarterly Journal of Economics*, 84(3):488–500, 1970. ISSN 0033-5533. doi: 10.2307/1879431. URL https://www.jstor.org/stable/1879431. Publisher: Oxford University Press.

Stefania Albanesi and Christopher Sleet. Dynamic optimal taxation with private information. *The Review of Economic Studies*, 73(1):1–30, 2006. ISSN 00346527, 1467937X. URL http://www.jstor.org/stable/3700615.

Eduardo M. Azevedo and Daniel Gottlieb. Perfect Competition in Markets With Adverse Selection. *Econometrica*, 85(1):67–105, 2017. ISSN 1468-0262. doi: 10.3982/ECTA13434. URL https://onlinelibrary.wiley.com/doi/abs/10.3982/ECTA13434. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.3982/ECTA13434.

Marlon Azinovic, Luca Gaegauf, and Simon Scheidegger. Deep equilibrium nets. *International Economic Review*, 2022. doi: https://doi.org/10.1111/iere.12575. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/iere.12575.

Marco Battaglini and Rohit Lamba. Optimal dynamic contracting: The first-order approach and beyond. *Theoretical Economics*, 14(4):1435–1482, 2019. ISSN 1555-7561. doi: 10.3982/TE2355. URL https://onlinelibrary.wiley.com/doi/abs/10.3982/TE2355. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.3982/TE2355.

R. Bellman. *Adaptive Control Processes: A Guided Tour*. 'Rand Corporation. Research studies. Princeton University Press, 1961. URL http://books.google.ch/books?id=POAmAAAAMAAJ.

Felix Berkenkamp, Angela P Schoellig, and Andreas Krause. Safe controller optimization for quadrotors with gaussian processes. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 491–496. IEEE, 2016.

Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 2nd edition, 2000. ISBN 1886529094.

Ilias Bilionis, Nicholas Zabaras, Bledar Konomi, and Guang Lin. Multi-output separable gaussian process: Towards an efficient, fully bayesian paradigm for uncertainty quantification. *Journal of Computational Physics*, 241:212–239, 05 2013. doi: 10.1016/j.jcp.2013.01.011.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

Tobias Broer, Marek Kapicka, and Paul Klein. Consumption risk sharing with private information and limited enforcement. *Review of Economic Dynamics*, 23:170 – 190, 2017. ISSN 1094-2025. doi: http://dx.doi.org/10.1016/j.red.2016.10.001. URL http://www.sciencedirect.com/science/article/pii/S1094202516300333.

Johannes Brumm and Simon Scheidegger. Using adaptive sparse grids to solve high-dimensional dynamic models. *Econometrica*, 85(5):1575–1612, 2017. ISSN 1468-0262. doi: 10.3982/ECTA12216. URL http://dx.doi.org/10.3982/ECTA12216.

Johannes Brumm, Christopher Krause, Andreas Schaab, and Simon Scheidegger. Sparse grids for dynamic economic models. *Available at SSRN 3979412*, 2021.

Johannes Brumm, Christopher Krause, Andreas Schaab, and Simon Scheidegger. Sparse grids for dynamic economic models. *Available at SSRN 3979412*, 2022.

Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, Inc., USA, 1st edition, 2010. ISBN 1439821089.

Yongyang Cai and Kenneth L Judd. Advances in numerical dynamic programming and new applications. *Handbook of computational economics*, 3, 2014.

James Cardon and Igal Hendel. Asymmetric Information in Health Insurance: Evidence from the National Medical Expenditure Survey. *RAND Journal of Economics*, 32(3):408–27, 2001. ISSN 0741-6261. URL https://econpapers.repec.org/article/rjerandje/v_3a32_3ay_3a2001_3ai_3a3_3ap_3a408-27.htm. Publisher: The RAND Corporation.

Kathryn Chaloner and Isabella Verdinelli. Bayesian experimental design: A review. *Statist. Sci.*, 10(3):273–304, 08 1995. doi: 10.1214/ss/1177009939. URL https://doi.org/10.1214/ss/1177009939.

Hui Chen, Antoine Didisheim, and Simon Scheidegger. Deep structural estimation: With an application to option pricing. *Available at SSRN 3782722*, 2021.

Steve Cicala, David Hémous, and Morten G Olsen. Adverse selection as a policy instrument: Unraveling climate change. Working Paper 30283, National Bureau of Economic Research, July 2022. URL http://www.nber.org/papers/w30283.

Alma Cohen and Liran Einav. Estimating risk preferences from deductible choice. 97(3):745–788, 2007. ISSN 0002-8282. doi: 10.1257/aer.97.3.745. URL https://www.aeaweb.org/articles?id=10.1257/aer.97.3.745.

Harold L. Cole and Narayana R. Kocherlakota. Efficient allocations with hidden income and hidden storage. *The Review of Economic Studies*, 68(3):523–542, 2001. doi: 10.1111/1467-937X.00179. URL +http://dx.doi.org/10.1111/1467-937X.00179.

Marc Peter Deisenroth, Carl Edward Rasmussen, and Jan Peters. Gaussian process dynamic programming. *Neurocomputing*, 72(7):1508–1524, 2009.

Peter M. DeMarzo and Yuliy Sannikov. Optimal security design and dynamic capital structure in a continuous-time agency model. *The Journal of Finance*, 61(6):2681–2724, 2006. ISSN 1540-6261. doi: 10.1111/j.1540-6261.2006.01002.x. URL http://dx.doi.org/10.1111/j.1540-6261.2006.01002.x.

Antoine Didisheim, Dimitrios Karyampas, and Simon Scheidegger. Implied risk aversion smile. *Available at SSRN 3533089*, 2020.

Matthias Doepke and Robert M. Townsend. Dynamic mechanism design with hidden income and hidden actions. *Journal of Economic Theory*, 126(1):235 – 285, 2006. ISSN 0022-0531. doi: http://dx.doi.org/10.1016/j.jet.2004.07.008. URL http://www.sciencedirect.com/science/article/pii/S0022053104001954.

Victor Duarte. Machine learning for continuous-time economics. 2018. Working paper.

Mahdi Ebrahimi Kahou, Jesús Fernández-Villaverde, Jesse Perla, and Arnav Sood. Exploiting symmetry in high-dimensional dynamic programming. Working Paper 28981, National Bureau of Economic Research, July 2021. URL http://www.nber.org/papers/w28981.

Aryan Eftekhari and Simon Scheidegger. High-dimensional dynamic stochastic model representation. *SIAM Journal on Scientific Computing*, 44(3):C210–C236, 2022. doi: 10.1137/21M1392231. URL https://doi.org/10.1137/21M1392231.

Yaakov Engel, Shie Mannor, and Ron Meir. Bayes Meets Bellman: The Gaussian Process Approach to Temporal Difference Learning. In *Proceedings of the 20th International Conference on Machine Learning*, pages 154–161. AAAI Press, 2003. ISBN 1-57735-189-4.

Hanming Fang, Michael P. Keane, and Dan Silverman. Sources of Advantageous Selection: Evidence from the Medigap Insurance Market. *Journal of Political Economy*, 116(2):303–350, April 2008. ISSN 0022-3808. doi: 10.1086/587623. URL https://www.journals.uchicago.edu/doi/10.1086/587623. Publisher: The University of Chicago Press.

Ana Fernandes and Christopher Phelan. A recursive formulation for repeated agency with history dependence. *Journal of Economic Theory*, 91(2):223 – 247, 2000. ISSN 0022-0531. doi: http://dx.doi.org/10.1006/jeth.1999.2619. URL http://www.sciencedirect.com/science/article/pii/S0022053199926194.

J. Fernández-Villaverde, S. Hurtado, and G. Nu no. Financial frictions and the wealth distribution. Working paper, 2019.

Jesús Fernández-Villaverde, Grey Gordon, Pablo Guerrón-Quintana, and Juan F Rubio-Ramirez. Nonlinear adventures at the zero lower bound. *Journal of Economic Dynamics and Control*, 57:182–204, 2015.

Amy Finkelstein and Kathleen McGarry. Multiple Dimensions of Private Information: Evidence from the Long-Term Care Insurance Market. *American Economic Review*, 96 (4):938–958, September 2006. ISSN 0002-8282. doi: 10.1257/aer.96.4.938. URL https://www.aeaweb.org/articles?id=10.1257/aer.96.4.938.

Jacob Gardner, Geoff Pleiss, Kilian Q Weinberger, David Bindel, and Andrew G Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. *Advances in neural information processing systems*, 31, 2018.

Alan Genz. Testing multidimensional integration routines. In *Proc. Of International Conference on Tools, Methods and Languages for Scientific and Engineering Computation*, pages 81–94, New York, NY, USA, 1984. Elsevier North-Holland, Inc. ISBN 0-444-87570-0. URL http://dl.acm.org/citation.cfm?id=2837.2842.

Mikhail Golosov, Aleh Tsyvinski, and Nicolas Werquin. Recursive contracts and endogenously incomplete markets. *Handbook of Macroeconomics*, 2:725–841, 2016.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Wouter J. Den Haan, Kenneth L. Judd, and Michel Juillard. Computational suite of models with heterogeneous agents ii: Multi-country real business cycle models. *Journal of Economic Dynamics and Control*, 35(2):175 – 177, 2011. ISSN 0165-1889. doi: 10.1016/j.jedc.2010.09.010. URL http://www.sciencedirect.com/science/article/pii/S0165188910002149. <ce:title>Computational Suite of Models with Heterogeneous Agents II: Multi-Country Real Business Cycle Models</ce:title>.

J. H. Halton. Algorithm 247: Radical-inverse quasi-random point sequence. *Commun. ACM*, 7(12):701–702, December 1964. ISSN 0001-0782. doi: 10.1145/355588.365104. URL http://doi.acm.org/10.1145/355588.365104.

Zhiguo He, Bin Wei, Jianfeng Yu, and Feng Gao. Optimal long-term contracting with learning. *The Review of Financial Studies*, 30(6):2006–2065, 2017. doi: 10.1093/rfs/hhx007.

Mitsuru Igami. Artificial intelligence as structural estimation: Deep Blue, Bonanza, and AlphaGo. *The Econometrics Journal*, 23(3):S1–S24, 03 2020. ISSN 1368-4221. doi: 10.1093/ectj/utaa005. URL https://doi.org/10.1093/ectj/utaa005.

Kenneth Judd, Sevin Yeltekin, and James Conklin. Computing supergame equilibria. *Econometrica*, 71(4):1239–1254, 2003. ISSN 00129682, 14680262. URL http://www.jstor.org/stable/1555496.

Kenneth L Judd. Projection methods for solving aggregate growth models. *Journal of Economic Theory*, 58(2):410–452, 1992.

Kenneth L Judd. *Numerical methods in economics*. The MIT press, 1998.

Kenneth L Judd, Lilia Maliar, Serguei Maliar, and Rafael Valero. Smolyak method for solving dynamic economic models: Lagrange interpolation, anisotropic grid and adaptive domain. *Journal of Economic Dynamics and Control*, 44:92–123, 2014.

Marek Kapička. Efficient Allocations in Dynamic Private Information Economies with Persistent Shocks: A First-Order Approach. *The Review of Economic Studies*, 80(3):1027–1054, 01 2013. ISSN 0034-6527. doi: 10.1093/restud/rds045. URL https://doi.org/10.1093/restud/rds045.

Michael P. Keane and Kenneth I. Wolpin. 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, 1994. ISSN 00346535, 15309142. URL http://www.jstor.org/stable/2109768.

Narayana R. Kocherlakota. Zero expected wealth taxes: A mirrlees approach to dynamic optimal taxation. *Econometrica*, 73(5):1587–1621, 2005. ISSN 1468-0262. doi: 10.1111/j.1468-0262.2005.00630.x. URL http://dx.doi.org/10.1111/j.1468-0262.2005.00630.x.

Laurence Kotlikoff, Felix Kubler, Andrey Polbin, and Simon Scheidegger. Pareto-improving carbon-risk taxation. *Economic Policy*, 36(107):551–589, 02 2021. ISSN 0266-4658. doi: 10.1093/epolic/eiab008. URL https://doi.org/10.1093/epolic/eiab008.

Andreas Krause and Carlos Guestrin. Nonmyopic active learning of gaussian processes: An exploration-exploitation approach. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, page 449–456, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595937933. doi: 10.1145/1273496.1273553. URL https://doi.org/10.1145/1273496.1273553.

Andreas Krause, Ajit Singh, and Carlos Guestrin. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *J. Mach. Learn. Res.*, 9:235–284, June 2008. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1390681.1390689.

Dirk Krueger and Felix Kubler. Computing equilibrium in OLG models with stochastic production. *Journal of Economic Dynamics and Control*, 28(7):1411 – 1436, 2004a. ISSN 0165-1889. doi: http://dx.doi.org/10.1016/S0165-1889(03)00111-8. URL http://www.sciencedirect.com/science/article/pii/S0165188903001118.

Dirk Krueger and Felix Kubler. Computing equilibrium in olg models with stochastic production. *Journal of Economic Dynamics and Control*, 28(7):1411 – 1436, 2004b. ISSN 0165-1889. doi: https://doi.org/10.1016/S0165-1889(03)00111-8. URL http://www.sciencedirect.com/science/article/pii/S0165188903001118.

Richard A. Lambert. Long-term contracts and moral hazard. *Bell Journal of Economics*, 14 (2):441–452, 1983.

L. Ljungqvist and T.J. Sargent. *Recursive macroeconomic theory*. Mit Press, 2000. ISBN 9780262194518. URL http://books.google.ch/books?id=3LmXQgAACAAJ.

David G. Luenberger and Yinyu Ye. *Linear and Nonlinear Programming*. International Series in Operations Research & Management Science. Springer US, 3 edition, 2008. ISBN 978-0-387-74502-2. doi: 10.1007/978-0-387-74503-9. URL https://www.springer.com/gp/book/9780387745022.

D. J. C. MacKay. Information-based objective functions for active data selection. *Neural Computation*, 4(4):590–604, July 1992. doi: 10.1162/neco.1992.4.4.590.

George J. Mailath, Ichiro Obara, and Tadashi Sekiguchi. The maximum efficient equilibrium payoff in the repeated prisoners' dilemma. *Games and Economic Behavior*, 40(1): 99 – 122, 2002. ISSN 0899-8256. doi: https://doi.org/10.1006/game.2001.0901. URL http://www.sciencedirect.com/science/article/pii/S0899825601909017.

Anastasia Makarova, Huibin Shen, Valerio Perrone, Aaron Klein, Jean Baptiste Faddoul, Andreas Krause, Matthias Seeger, and Cedric Archambeau. Automatic termination for hyperparameter optimization. Technical report, 2022. URL https://openreview.net/forum?id=2NqIV8dzR7N.

Lilia Maliar and Serguei Maliar. Numerical methods for large-scale dynamic economic models. In *Handbook of computational economics*, volume 3, pages 325–477. Elsevier, 2014.

Lilia Maliar, Serguei Maliar, and Pablo Winant. Deep learning for solving dynamic economic models. *Journal of Monetary Economics*, 122:76–101, 2021. ISSN 0304-3932. doi: https://doi.org/10.1016/j.jmoneco.2021.07.004. URL https://www.sciencedirect.com/science/article/pii/S0304393221000799.

Albert Marcet and Ramon Marimon. Recursive contracts. *Econometrica*, 87(5):1589–1631, 2019. doi: https://doi.org/10.3982/ECTA9902. URL https://onlinelibrary.wiley.com/doi/abs/10.3982/ECTA9902.

Laurent Mathevet, David Pearce, and Ennio Stacchetti. Reputation for a degree of honesty. Technical report, Working Paper, 2022.

Costas Meghir and Luigi Pistaferri. Income variance dynamics and heterogeneity. *Econometrica*, 72(1):1–32, 2004. ISSN 1468-0262. doi: 10.1111/j.1468-0262.2004.00476.x. URL http://dx.doi.org/10.1111/j.1468-0262.2004.00476.x.

Antonio Mele. Repeated moral hazard and recursive lagrangeans. *Journal of Economic Dynamics and Control*, 42:69 – 85, 2014. ISSN 0165-1889. doi: https://doi.org/10.1016/j.jedc.2014.03.007. URL http://www.sciencedirect.com/science/article/pii/S0165188914000669.

Charles A. Micchelli, Yuesheng Xu, and Haizhang Zhang. Universal kernels. *J. Mach. Learn. Res.*, 7:2651–2667, dec 2006. ISSN 1532-4435.

T.M. Mitchell. *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill, 1997. ISBN 9780071154673. URL https://books.google.ch/books?id=EoYBngEACAAJ.

Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN 9780262018029.

Kevin P. Murphy. *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2022. URL probml.ai.

Harald Niederreiter. *Random Number Generation and quasi-Monte Carlo Methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992. ISBN 0-89871-295-5.

William L. Oberkampf and Christopher J. Roy. *Verification and Validation in Scientific Computing*. Cambridge University Press, 2010. doi: 10.1017/CBO9780511760396.

J. Park and I. W. Sandberg. Universal approximation using radial-basis-function networks. *Neural Computation*, 3(2):246–257, June 1991. ISSN 0899-7667. doi: 10.1162/neco.1991.3.2.246.

Alessandro Pavan, Ilya Segal, and Juuso Toikka. Dynamic mechanism design: A myersonian approach. *Econometrica*, 82(2):601–653, 2014.

Nicola Pavoni, Christopher Sleet, and Matthias Messner. The dual approach to recursive optimization: Theory and examples. *Forthcoming, Econometrica*, 2017.

T. Poggio and F. Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497, Sep 1990. ISSN 0018-9219. doi: 10.1109/5.58326.

Carl Edward Rasmussen and Hannes Nickisch. Gaussian processes for machine learning (gpml) toolbox. *Journal of Machine Learning Research*, 11(100):3011–3015, 2010. URL http://jmlr.org/papers/v11/rasmussen10a.html.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. ISBN 026218253X.

William P. Rogerson. Repeated moral hazard. *Econometrica*, 53(1):69–76, 1985.

John Philip Rust. Numerical dynamic programming in economics. In H. M. Amman, D. A. Kendrick, and J. Rust, editors, *Handbook of Computational Economics*, volume 1, chapter 14, pages 619–729. Elsevier, 1 edition, 1996. URL http://EconPapers.repec.org/RePEc:eee:hecchp:1-14.

Alvaro Sandroni and Francesco Squintani. Overconfidence, insurance, and paternalism. 97(5):1994–2004, 2007. ISSN 0002-8282. doi: 10.1257/aer.97.5.1994. URL https://www.aeaweb.org/articles?id=10.1257/aer.97.5.1994.

Yuliy Sannikov. A continuous- time version of the principal: Agent problem. *The Review of Economic Studies*, 75(3):957–984, 2008. ISSN 00346527, 1467937X. URL http://www.jstor.org/stable/20185061.

Yuliy Sannikov. Breaking the curse of dimensionality. Technical report, 2022.

Simon Scheidegger and Ilias Bilionis. Machine learning for high-dimensional dynamic stochastic economies. *Journal of Computational Science*, 33:68 – 82, 2019. ISSN 1877-7503. doi: https://doi.org/10.1016/j.jocs.2019.03.004. URL http://www.sciencedirect.com/science/article/pii/S1877750318306161.

Anthony Skjellum, William Gropp, and Ewing Lusk. *Using MPI*. MIT Press, 1999.

Christopher Sleet and Sevin Yeltekin. On the computation of value correspondences for dynamic games. *Dynamic Games and Applications*, 6(2):174–186, Jun 2016. ISSN 2153-0793. doi: 10.1007/s13235-015-0139-1. URL https://doi.org/10.1007/s13235-015-0139-1.

Stephen E. Spear and Sanjay Srivastava. On repeated moral hazard with discounting. *The Review of Economic Studies*, 54(4):599–617, 1987. doi: 10.2307/2297484.

Johannes Spinnewijn. Heterogeneity, Demand for Insurance, and Adverse Selection. *American Economic Journal: Economic Policy*, 9(1):308–343, February 2017. ISSN 1945-7731. doi: 10.1257/pol.20140254. URL https://www.aeaweb.org/articles?id=10.1257/pol.20140254.

Stefanie Stantcheva. Optimal taxation and human capital policies over the life cycle. 125(6): 1931–1990, 2017. ISSN 0022-3808. doi: 10.1086/694291. URL https://www.journals.uchicago.edu/doi/full/10.1086/694291. Publisher: The University of Chicago Press.

Nancy Stokey, Robert Lucas, and Edward Prescott. Recursive methods in economic dynamics. *Cambridge MA*, 1989a.

Nancy Stokey, Robert Jr Lucas, and Edward Prescott. *Recursive Methods in Economic Dynamics*. Harvard University Press, Cambridge, MA, 1989b.

Kjetil Storesletten, Christopher Telmer, and Amir Yaron. Consumption and risk sharing over the life cycle. *Journal of Monetary Economics*, 51(3):609 – 633, 2004. ISSN 0304-3932. doi: http://dx.doi.org/10.1016/j.jmoneco.2003.06.005. URL http://www.sciencedirect.com/science/article/pii/S0304393204000182.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Alessandro Tenzin Villa and Vytautas Valaitis. Machine learning projection methods for macro-finance models. Working paper, 2018.

Andreas Waechter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math. Program.*, 106 (1):25–57, May 2006. ISSN 0025-5610. doi: 10.1007/s10107-004-0559-y. URL http://dx.doi.org/10.1007/s10107-004-0559-y.

Cheng Wang. Dynamic Insurance with Private Information and Balanced Budgets. *The Review of Economic Studies*, 62(4):577–595, 1995. ISSN 0034-6527. doi: 10.2307/2298078. URL http://www.jstor.org/stable/2298078.

Ivan Werning. Optimal unemployment insurance with unobservable savings, 2002.

Noah Williams. On dynamic principal-agent problems in continuous time. 2009.

Noah Williams. Persistent private information. *Econometrica*, 79(4):1233–1275, 2011. ISSN 1468-0262. doi: 10.3982/ECTA7706. URL http://dx.doi.org/10.3982/ECTA7706.

Andrew Wilson and Hannes Nickisch. Kernel interpolation for scalable structured gaussian processes (kiss-gp). In *International conference on machine learning*, pages 1775–1784. PMLR, 2015.

Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Artificial intelligence and statistics*, pages 370–378. PMLR, 2016.

Yue Wu, Hui Wang, Biaobiao Zhang, and K.-L. Du. Using radial basis function networks for function approximation and classification. *ISRN Applied Mathematics*, 2012. doi: 10.5402/2012/324194. URL https://www.hindawi.com/journals/isrn/2012/324194/cta/.

Sevin Yeltekin, Yongyang Cai, and Kenneth L. Judd. Computing equilibria of dynamic games. *Operations Research*, 65(2):337–356, 2017. doi: 10.1287/opre.2016.1572.