respy
A prototypical finite-horizon discrete choice dynamic programming model

Janos Gabler[1] and Tobias Raabe[2]

[1] Economics, Universität Bonn, Germany
`janos.gabler@uni-bonn.de`
[2] Economics, Universität Bonn, Germany
`tobiasraabe@uni-bonn.de`

Abstract

One of the most challenging tasks in economics is to forecast the effects of policies that have not yet been implemented. To achieve this goal economists rely on structural estimation. This approach requires the researcher to build a theoretical model of the observed behavior. Then, the parameters of the model are estimated on empirical data. By modeling the full decision process of economic agents, structural models are well suited to predict behavior even after changes to the environment.

Because of the computational challenges, previous work in this research area has relied on Fortran to implement the models. Here we present `respy`, a prototypical finite-horizon discrete choice dynamic programming model to analyze the economics behind an individuals occupational and educational choice. It is implemented in both, Fortran and Python, and achieves comparable runtime in both versions.

1 Introduction

Policy evaluation is at the center of modern economics. Possible policy evaluation tasks can be categorized as follows: Firstly, the evaluation of a policy that has been implemented in the past. Secondly, predicting how a policy implemented in one time or place would affect behavior and outcomes in a different time and/or place. Thirdly, predicting the effect of a policy that has never been implemented. An example of the third type would be the valuation of the effect of tuition fees in Germany on unemployment before the introduction in 2006/07.

To predict the impact of policies that never have been implemented before, economists often rely on structural estimation. Structural estimation consists of three steps. First, researchers formulate a theoretical economic model to explain the observed behavior. Then, the model parameters are estimated on a suitable data set. Lastly, the policy parameter (in the example above the the amount of the tuition fees) is changed and simulations are run with the modified model to make predictions.

The estimation of such models is computationally expensive and their implementation usually requires advanced programming skills and knowledge of numerical optimization. One important class of structural models are finite-horizon discrete choice dynamic programming (DCDP) models. These are models where agents are able to perform an action from a discrete number of options at a number of discrete points in time. As an example, individuals might choose whether to continue education, be unemployed or work in one of two sectors every year from 15 to 65. In the field of labor economics, DCDP models have been frequently used to address a series of important issues such as the evaluation of policies aimed at increasing education, the transition from unemployment to employment, or the interrelation between fertility, marriage, and work decisions over the life-cycle.

In this study we present a prototypical model to study the economics behind agents' educational and occupational choices over their life cycle using the framework of a finite-horizon DCDP model. By fitting the model to data we are able to investigate how agents form their occupational and education decisions over their life cycle. Furthermore, the model allows us to study the impact of a hypothetical policy which subsidizes education or provides higher unemployment benefits. We will discuss the computational challenges of this modeling decision and compare the Fortran against the Python implementation.

The remainder of the paper is structured as follows: In Section 2, we present the basic economic model and derive its solution. We outline the estimation of the model with data. Section 3 shows details of the model using a simulated data set. We look at the returns to experience and decisions over the life cycle and show the effects of a new policy. Section 4 illustrates the computational challenges in the implementation of `respy` and shows a speed comparison between Fortran and Python. In section 5, we draw some conclusions and present possible extensions to the model.

2 Basic Setup

In the following, we present the model of [10] and state their assumptions about functional forms and the distributions of unobservables. Then, we turn to the model solution and briefly outline the estimation approach.

2.1 Economic Model

In the model, an agent decides among K possible alternatives in each of T (finite) discrete periods of time [10]. Alternatives are defined to be mutually exclusive and $d_k(t) = 1$ indicates that alternative k is chosen at time t and $d_k(t) = 0$ indicates otherwise. Associated with each choice is an immediate reward $R_k(S(t))$ that is known to the agent at time t but partly unknown from the perspective of periods prior to t. The state space $S(t)$ encompasses all the information available to the agent at time t that affects immediate and future rewards.

At the beginning of each period t, the agent fully learns about all immediate rewards, chooses one of the alternatives and receives the corresponding payoffs. The state space is then updated according to the agent's state experience and the process is repeated in $t+1$. Agents are forward looking. Thus, they do not simply choose the alternative with the highest immediate rewards each period. Instead, their objective at any time τ is to maximize the expected rewards over the remaining time horizon:

$$\max_{\{d_k(t)\}_{k\in K}} E\left[\sum_{\tau=t}^{T} \delta^{\tau-t} \sum_{k\in K} R_k(\tau) d_k(\tau) \middle| S(t)\right]. \tag{1}$$

The discount factor $0 > \delta > 1$ captures the agent's preference for immediate over future rewards. Agents maximize Equation 1 by choosing the optimal sequence of alternatives $\{d_k(t)\}_{k\in K}$ for $t = \tau, \ldots, T$.

Within this more general model framework, [10] then impose additional functional form and distributional assumptions that define their prototypical model of occupational choice. Agents live for a total of 40 periods and are risk neutral. Each period, agents choose to work in either of two occupations (k = 1, 2), to attend school (k = 3), or to remain at home (k = 4). The immediate reward functions are given by:

$$
\begin{aligned}
R_1(t) &= w_{1t} = \exp\{\alpha_{10} + \alpha_{11}s_t + \alpha_{12}x_{1t} - \alpha_{13}x_{1t}^2 + \alpha_{14}x_{2t} - \alpha_{15}x_{2t}^2 + \epsilon_{1t}\} \\
R_2(t) &= w_{2t} = \exp\{\alpha_{20} + \alpha_{21}s_t + \alpha_{22}x_{2t} - \alpha_{23}x_{2t}^2 + \alpha_{24}x_{1t} - \alpha_{25}x_{1t}^2 + \epsilon_{2t}\} \\
R_3(t) &= \beta_0 - \beta_1 I(s_t \geq 12) - \beta_2(1 - d_3(t-1)) + \epsilon_{3t} \\
R_4(t) &= \gamma_0 + \epsilon_{4t},
\end{aligned}
$$

where s_t is the number of periods of schooling obtained by the beginning of period t, x_{1t} is the number of periods that the agent worked in occupation one by the beginning of period t, x_{2t} is the analogously defined level of experience in occupation two, α_1 and α_2 are parameter vectors associated with the wage functions, β_0 is the consumption value of schooling, β_1 is the post-secondary tuition cost of schooling, with I as an indicator function equal to one if the agent completed high school and zero otherwise, β_2 is an adjustment cost associated with returning to school, γ_0 is the (mean) value of the non-market alternative. The ϵ_{kt}'s are alternative-specific shocks, to occupational productivity, to the consumption value of schooling, and to the value of non-market time. The productivity and taste shocks follow a four-dimensional multivariate normal distribution with mean zero and covariance matrix $\Sigma = [\sigma_{ij}]$. They collect the parameterization of the reward functions in $\theta = \{\alpha_1, \alpha_2, \beta, \gamma, \Sigma\}$.

Given the structure of the reward functions and the agent's objective, the state space at time t is $S(t) = \{s_t, x_{1t}, x_{2t}, d_3(t-1), \epsilon_{1t}, \epsilon_{2t}, \epsilon_{3t}, \epsilon_{4t}\}$. It is convenient to denote its observable elements as $\bar{S}(t)$. The elements of $S(t)$ evolve according to:

$$
\begin{aligned}
x_{1,t+1} &= x_{1t} + d_1(t) \\
x_{2,t+1} &= x_{2t} + d_2(t) \\
s_{t+1} &= s_t \ + d_3(t) \\
f(\epsilon_{t+1} \mid S(t), d_k(t)) &= f(\epsilon_{t+1} \mid \bar{S}(t), d_k(t)),
\end{aligned}
$$

where the last equation reflects the fact that the ϵ_{kt}'s are serially independent. They set the initial conditions as $x_{1t} = x_{2t} = 0$ and $s_0 = 10$. Agents cannot attain more than ten additional years of schooling. Note that all agents start out identically, different choices over the life cycle are simply the cumulative effects of different shocks.

2.2 Solution

From a mathematical perspective, the model is a finite-horizon dynamic programming (DP) problem under uncertainty that can be solved by backward induction. It is useful to define the value function $V(S(t), t)$ as a shorthand for Equation 1. $V(S(t), t)$ depends on the state space at t and on t itself due to the finiteness of the time horizon and can be written as:

$$
V(S(t), t) = \max_{k \in K}\{V_k(S(t), t)\},
$$

with $V_k(S(t), t)$ as the alternative-specific value function. $V_k(S(t), t)$ obeys the Bellman equation [2] and is thus amenable to a backward induction.

$$
V_k(S(t), t) = \begin{cases} R_k(S(t)) + \delta E\left[V(S(t+1), t+1) \mid S(t), d_k(t) = 1\right] & \text{if } t < T \\ R_k(S(t)) & \text{if } t = T \end{cases}
$$

Assuming continued optimal behavior, the expected future value of state $S(t+1)$ for all K alternatives given today's state $S(t)$ and choice $d_k(t) = 1$, $E\max(S(t+1))$ for short, can be calculated:

$$E\max(S(t+1)) = E\left[V(S(t+1), t+1) \mid S(t), d_k(t) = 1\right].$$

This requires the evaluation of a four-dimensional integral as future rewards are partly uncertain due to the unknown realizations of the shocks. With all ingredients at hand, the solution of the model by backward induction is straightforward.

2.3 Estimation

The estimation of the model is standard and based on maximum simulated likelihood [8, 13]. For example, we simulate the choice probabilities to evaluate the sample log-likelihood. With only a finite number of draws, there is always the risk of simulating zero probability for an agent's observed decision. So we use the logit-smoothed accept-reject simulator as suggested by [14]. The scale parameter is set to 500 as in [10].

3 Application

To study the model in more detail, we simulate a sample of agents from the first of three different model parameterizations of [10]. There are four choices in the model, occupations A and B, schooling and home.

In Figure 1 (a), wages in occupation A are displayed in relation to experience in both sectors. Skills learned in occupation B are not transferable to occupation A and do not raise the wage. In contrast to that, some general skill learned in occupation A is also useful in occupation B and raises the wage. This can be seen in Figure 1 (b). In general, wages in occupation B tend to be much higher.

Figure 2 displays the differential effect of education on the returns to being in occupation A or B. While the initial wage in occupation B is lower, it increases faster with schooling compared to occupation A.

Simulating a sample of 1,000 agents from the model allows us to investigate how these features interact in determining an agent's decisions over their life cycle. Note that all agents start out identically. Different choices are simply the cumulative effects of different shocks. Initially, 50% of agents increase their level of schooling but the share of agents enrolled in school declines sharply over time. The share working in occupation A hovers around 40% at first, but then decreases to 21%. Occupation B continuously gains in popularity, initially only 11% work in occupation B but its share increases to about 77%. Around 1.5% stay at home each period. We visualize this choice pattern in detail below.

We start out with the large majority of agents working in occupation A. Eventually, however, most agents end up working in occupation B. The returns to education are higher for occupation B and previous work experience is transferable. Occupation B gets more and more attractive as agents increase their level of schooling and gain experience in the labor market.

For Figure 4, we assume that the cost of education in high school and college increase from 4,000 to 10,000 per year. Again, we simulate a sample of 1,000 agents and look at the distribution of choices over the life cycle. As the cost of education increases, fewer people re-enroll into high school or college. This drives down returns in occupation B which is still

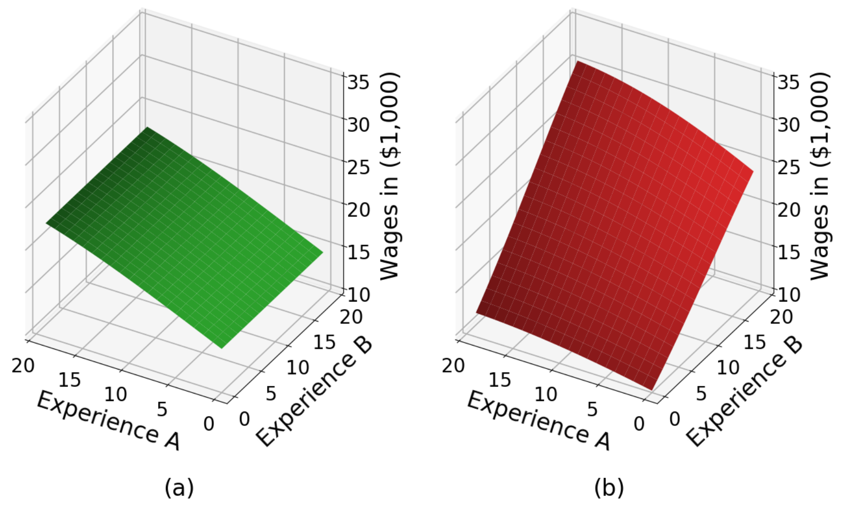

Figure 1: Returns to experience in occupations
Note: Years of schooling are held constant at ten years. All other parameters of the reward function are set to zero.

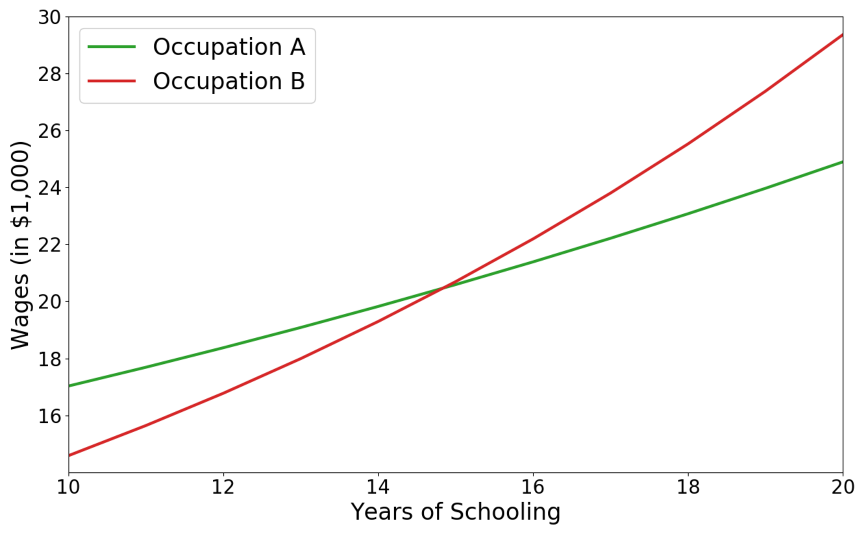

Figure 2: Returns to schooling
Note: For this graph, years of experience in both occupations are held constant at five years.

the choice of the majority, but now the stable share is 65% instead of 80%. In addition, more people stay at home as education is too costly to obtain and thus they never earn wages high enough to surpass the returns from home production.

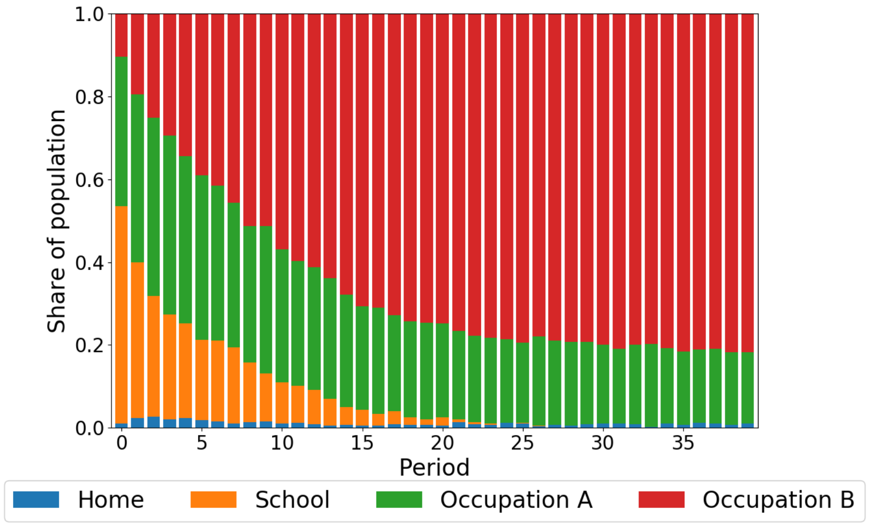

Figure 3: Agents' decision over their life cycle

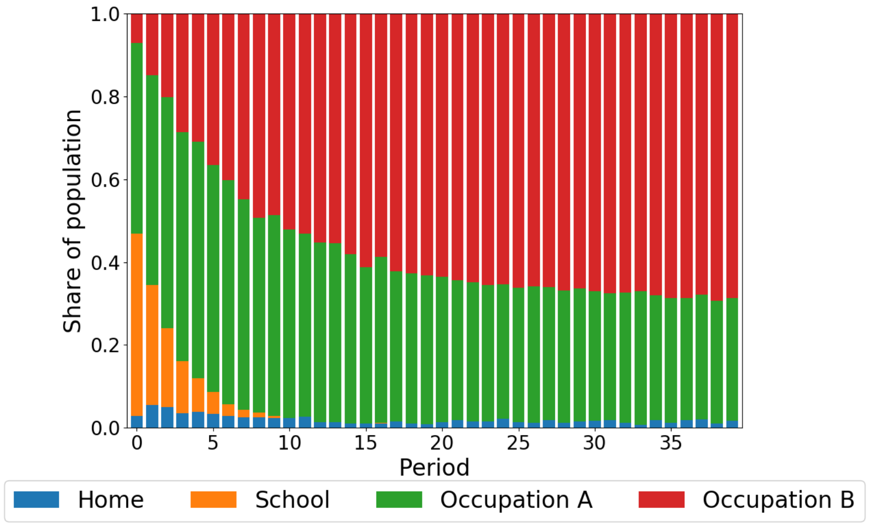

Figure 4: Agent's decision over their life cycle. Cost of education increased.

4 Challenges and Implementation

This section lays out the computational challenges inherent to the model. We will discuss how `respy` [7][1] helps researchers to overcome these challenges by offering two implementations. First, the Fortran implementation is presented. Secondly, we show that Python has become a

[1]The project is publicly available under `https://github.com/OpenSourceEconomics/respy` and the documentation under `https://respy.readthedocs.io/en/latest/`.

suitable alternative to low-level programming languages and is able to provide many benefits while also mitigating disadvantages.

The computational challenges of this model are exacerbated by two factors. The first factor is that the state space of the model increases exponentially. This problem is well known as the curse of dimensionality [5] and is illustrated in Figure 5. The graph shows the number of all admissible states which consist of a combination of five elements, period, experiences in both occupations, years of schooling and the lagged choice. This impacts runtime significantly as the solution of the model requires to determine optimal behavior in every state. The second factor is the estimation of the true parameters of model with the data. The model has to be solved and the probability of observing the agent's decision has to be calculated for each candidate solution. Considering that the optimization involves several hundreds evaluations and robustness checks demand for multiple optimization processes with different initial parameter vectors, performance is critical.

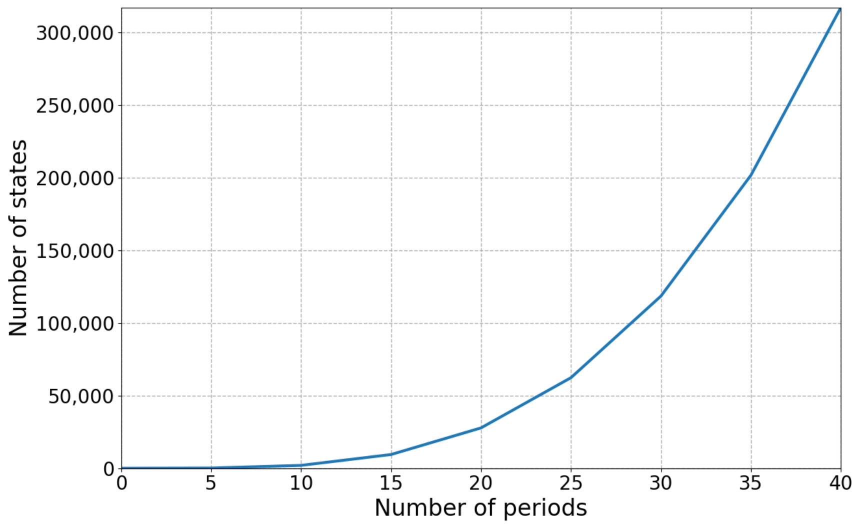

Figure 5: The exponentially increasing number of states per period.

The first critical component of the model is the backward induction. The optimal behavior in future periods determines the optimal decision in previous periods. Thus, the complete process cannot be parallelized without much effort, but it is possible to parallelize the process within each period. This leads directly to the second computationally expensive components in the model, two Monte Carlo integrations. The first Monte Carlo integration evaluates the $E\max$, the expected maximum value, for all states in the state space. Secondly, the probability of observing the agent's choice is also calculated with Monte Carlo integration. In both settings the researcher has to define the number of points where the integral is evaluated and thus is faced with a trade-off between numerical precision and runtime.

The original model by [10] was implemented in Fortran to shoulder the computational burden in 1994. Fortran is the first higher programming language with a long tradition in research areas with numerical problems. Due to its specialization to algorithmic problems Fortran has a clear syntax and compilers which generate highly optimized machine code. Advances in computing power over the last 25 years potentially mask the need for Fortran in 1994. For comparison, [6] mentions that the exact solution of the model took ca. 50 minutes on a CRAY 2 supercomputer in 1994. In 2016 using a single process of the Acropolis cluster from the University of Chicago the solution time dropped to nine minutes.

The Fortran version of `respy` does not build upon the original codes written in FORTRAN 77. The implementation is written in Fortran 90 and leverages OpenMP [3] and MPI [9] to achieve major performance boosts. OpenMP is used to accelerate the Monte Carlo integrations. MPI is used to parallelize the evaluation of sample likelihood across individuals.

Python was released in 1991 by Guido van Rossum to replace ABC, a programming language for teaching and prototyping. Because of that, Python has many features which make it a perfect language for newcomers or researchers with no background in computer science. The language is interpreted, interactive, dynamically typed and has a simple syntax which allow for rapid development. It has a rich standard library and an even more versatile ecosystem with many libraries supporting scientific applications. Python can be used as a glue language to tie components written in different languages together. For instance, `respy`'s interface is solely written in Python, but can call routines written in Fortran via `F2PY` [15].

The main disadvantage of Python is that it is not as performant as Fortran due to dynamic typing. Using scientific packages like NumPy which use typed arrays and call routines written in low-level languages, partially address this problem. However, to achieve performance similar to Fortran, we rely on Numba [12] - a JIT-compiler for Python functions. Upon calling the function, Numba will infer the types of the inputs and translate the function to optimized machine code. Numba incorporates two important implementation decisions. First, Numba does not require the function to be written in another language. It can be as easy as writing a function in Python and extending it with a decorator, a wrapper around the function containing instructions for Numba. However, Numba only supports a subset of Python's and NumPy's capabilities. In addition, writing vector operations as loops, using if-statements and creating intermediate variables may significantly increase the performance of the compiled function. In Python, these operations are notoriously slow. Secondly, Numba targets only functions and not the whole program. Thus, it forces the programmer to think about the design of the implementation, to isolate critical components which are then reimplemented with Numba. In `respy`, we use Numba mainly for the Monte Carlo integrations and other functions which are called often.

The following scalability exercise in Figure 6 shows 1,000 evaluations of the likelihood of the model for the Fortran and the Python version. Fortran is displayed with the blue line and parallelizes the Monte Carlo integration with OpenMP. The Python implementation is shown in orange and restricted in all its calls to the number of threads indicated in the figure. Regardless of the number of threads, the Python version is up to ten times faster than the Fortran implementation. This result is remarkable in the sense that for `respy` it is sufficient to target some bottlenecks in the implementation to get performance way better than implementations in low-level languages.

However, the result is not without caveats. The first is that the Python implementation avoids some computations by reusing pre-computed values which the Fortran version does not. These changes have not made it into the stable version, but the preliminary results indicate a ten-fold increase in performance. The second caveat is that the Fortran version also supports parallelization by MPI which is not used here to keep the results similar.

Besides that the Python implementation is now ten times faster than Fortran, we have two other reasons to favor Python over Fortran. First, the entry barrier to Python is significantly lower. Thus, the project opens up to students, beginners in programming and the scientific audience and has the potential to be iteratively improved by contributions from many sources.

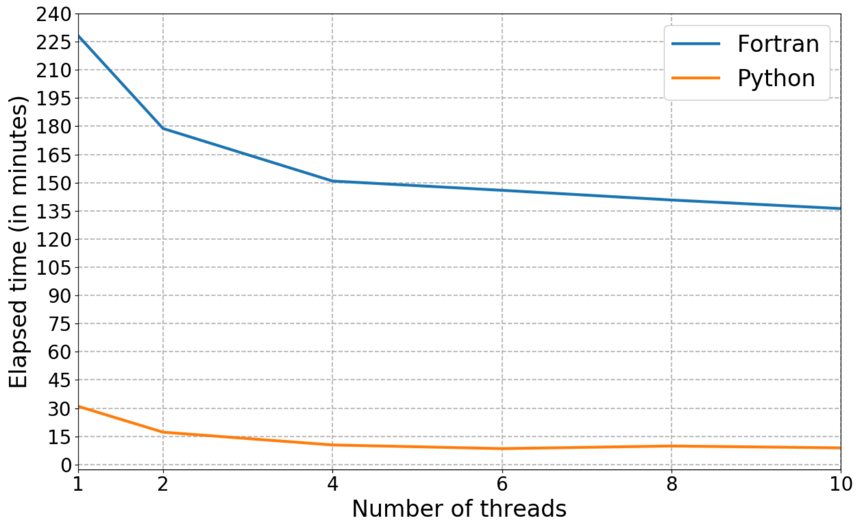

Figure 6: Computation time for 1,000 evaluations of the model likelihood.

This is amplified by the modular structure of `respy` which facilitates improving isolated components. Furthermore, insights generated in this project can be transfered to other research project and initiate progress. For instance, `respy` is part of Open Source Economics[2] where it serves as a reference implementation for others.

Secondly, Python offers many solutions to ensure the quality of the implementation. The most important tool in this category for `respy` is `pytest` [11], a scalable framework for writing tests. In combination with our comprehensive test suite it ensures that changes do not lead to regressions. Other tools keep the code base and the documentation well formatted or raise warnings if changes do not adhere to the style guide. In addition, these tools integrate well with a number of continuous integration services which continuously monitor the implementation.

5 Conclusion and future work

The key take-away is that depending on the application, Python offers ways to overcome the inherent limitations on an interpreted language. By using Numba one is able to achieve performance in components similar to C and Fortran. Thus, researchers are not forced to learn a new language, but can stay inside the versatile Python ecosystem which enables faster development. The gain in time due to higher productivity can be more valuable than reduced runtime. Particularly with regard to the continuing progress of computer hardware, runtime may become less of a problem.

Regarding the implementation, the `respy` project serves not only as a prototypical finite-horizon DCDP model, but also as a teaching tool and a reference implementation for future research. There are some other projects under development which derive insights from `respy` and implement them in their setup.

For the future, we plan to enable more parallelization by using tools like `mpi4py` [4]. In addition, we want to implement other estimation techniques like simulated method of moments

[2]See `https://github.com/OpenSourceEconomics` for an overview of related projects.

[14] or conditional choice probability estimation [1]. To assess the robustness of the optimization, we want to review existing frameworks and algorithms for the numerical optimization of the model parameters.

As the model sits in the point of intersection between mathematics, computer science, and economics, we always welcome contributions from other disciplines and researchers related to Open Source Economics offer supervision as part of a theses project.

References

[1] Peter Arcidiacono and Robert A Miller. Conditional choice probability estimation of dynamic discrete choice models with unobserved heterogeneity. *Econometrica*, 79(6):1823–1867, 2011.

[2] Richard Bellman. Dynamic programming. *Princeton*, 1957.

[3] Leonardo Dagum and Ramesh Menon. Openmp: An industry-standard api for shared-memory programming. *Computing in Science & Engineering*, (1):46–55, 1998.

[4] Lisandro Dalcín, Rodrigo Paz, and Mario Storti. Mpi for python. *Journal of Parallel and Distributed Computing*, 65(9):1108–1115, 2005.

[5] Stuart E Dreyfus and Richard Ernest Bellman. *Applied dynamic programming.* 1962.

[6] Philipp Eisenhauer. The approximate solution of finite-horizon discrete-choice dynamic programming models. *Journal of Applied Econometrics*, 34(1):149–154, 2019.

[7] Philipp Eisenhauer. respy - a python package for the simulation and estimation of a prototypical discrete choice dynamic programming model. 2019.

[8] Ronald A Fisher. On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 222(594-604):309–368, 1922.

[9] William D Gropp, William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: portable parallel programming with the message-passing interface*, volume 1. MIT press, 1999.

[10] 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*, pages 648–672, 1994.

[11] Holger Krekel, Bruno Oliveira, Ronny Pfannschmidt, Floris Bruynooghe, Brianna Laugher, and Florian Bruhin. pytest 4.0, 2004.

[12] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15, pages 7:1–7:6, New York, NY, USA, 2015. ACM.

[13] Charles F Manski and Steven R Lerman. The estimation of choice probabilities from choice based samples. *Econometrica: Journal of the Econometric Society*, pages 1977–1988, 1977.

[14] Daniel McFadden. A method of simulated moments for estimation of discrete response models without numerical integration. *Econometrica: Journal of the Econometric Society*, pages 995–1026, 1989.

[15] Pearu Peterson. F2py: a tool for connecting fortran and python programs. *International Journal of Computational Science and Engineering*, 4(4):296–305, 2009.