# respy
# A prototypical finite-horizon discrete choice dynamic programming model

Janos Gabler Tobias Raabe

CSCUBS 2019

## Table of Contents

Introduction

Economic Model

Application

Challenges and Implementation

Conclusion and Future Work

Bibliography

## Introduction

## Introduction

- Policy evaluation is at the center of economics.
- Predicting the impact of policies that have never been implemented before is done via structural estimation.
- Structural models are computationally expensive and require advanced programming skills.
- `respy` is a finite-horizon discrete choice dynamic programming model for modeling occupational choices.

## Economic Model

## Economic Model

- Agents are faced with four choices every period (occupation A and B, schooling or home production)
- Agents are forward-looking and thus, the optimal decision in the first period depends on the optimal decision in following states until the last period.
- The solution of the model obeys the Bellman equation [1], it can be solved with backward induction.
- As agents face uncertainty in future periods, the value of being in future states is calculated by Monte Carlo integration.

## Application

## Returns to experience

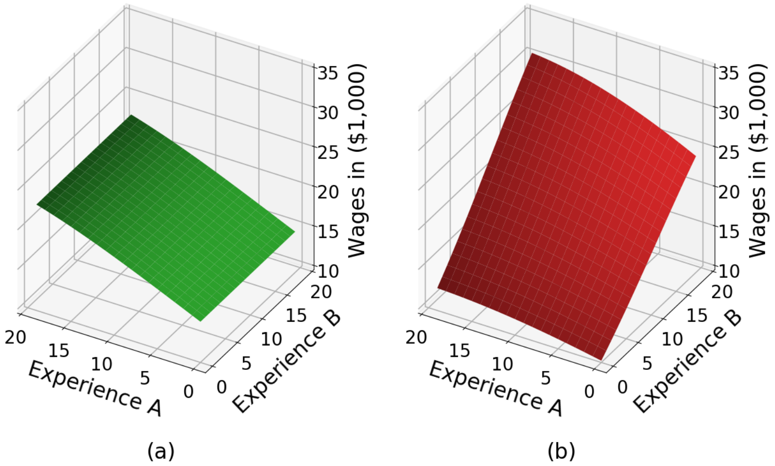

Figure: Returns to experience

## Returns to schooling

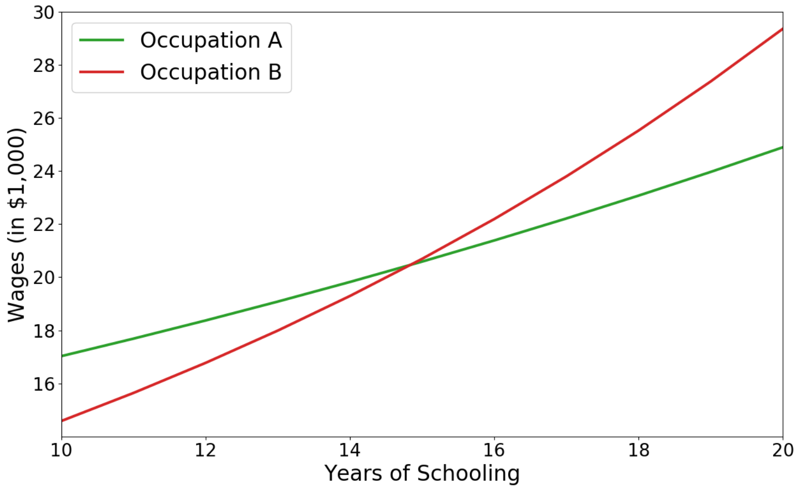

Figure: Returns to schooling

# Choices over the life cycle

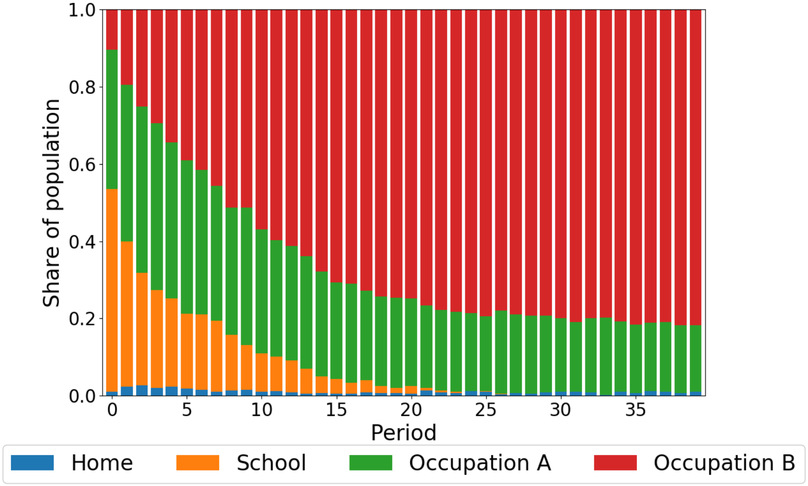

Figure: Choices over the life cycle

# Choices over the life cycle

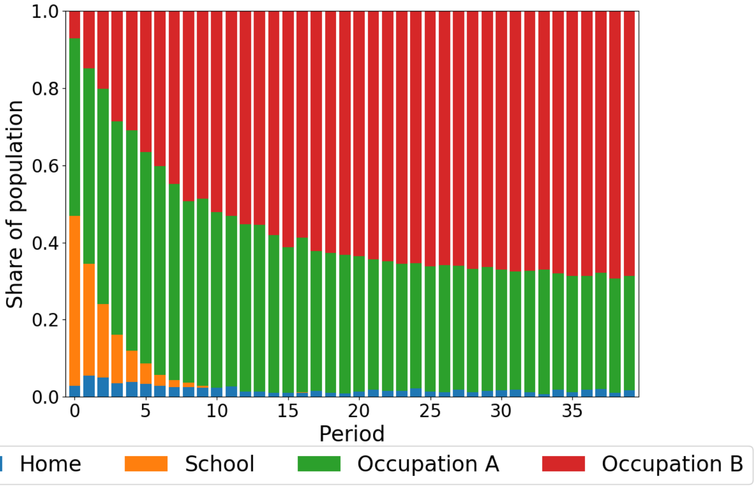

Figure: Choices over the life cycle with increased costs for schooling

## Challenges and Implementation

## Challenges

Computational challenges of the model are exacerbated by two factors:

1. The state space increases exponentially.
2. While finding the optimal parameters of the model, the model has to be solved and the probability of the data given the model calculated for each candidate solution.

## Challenges

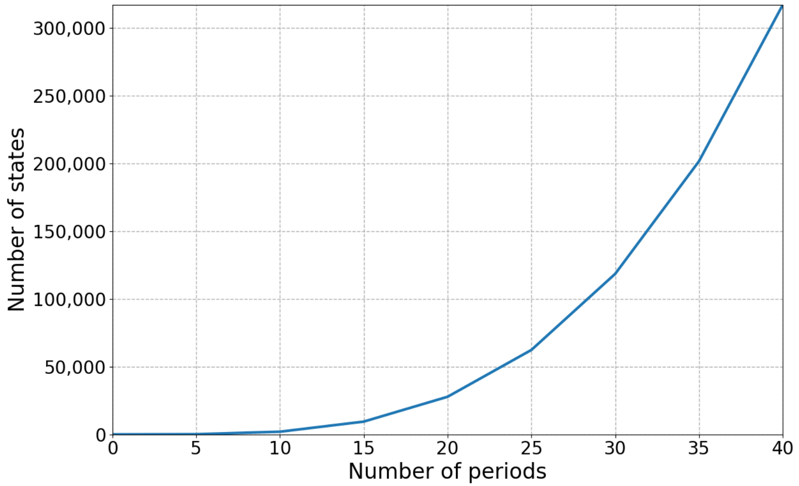

Figure: Number of states per period

## Challenges

Computational challenges of the model:

1. The backward induction is a sequential process and can only be easily parallelized across states in a period.
2. There is a trade-off between precision and runtime governing the number of points where the integrand is evaluated.

## Implementation

Fortran

- Fortran 90.
- OpenMP support to parallelize the Monte Carlo integration.

Python

- Python is normally slow as it is dynamically typed.
- Numba[3] can reach performance similar compared to Fortran, C, C++ by JIT-compiling type-specialized functions.

## Implementation

```
@guvectorize(
    ["f8[:], f8[:], f8[:], f8[:, :], f8, b1, f8[:, :]"],
    "(m), (n), (n), (p, n), (), () -> (n, p)",
    nopython=True,
    target="cpu",
)
def get_continuation_value(
    wages, rewards_systematic, emaxs, draws, delta, max_education, cont_value
):
    num_draws, num_choices = draws.shape
    num_wages = wages.shape[0]

    for i in range(num_draws):
        for j in range(num_choices):
            if j < num_wages:
                rew_ex = wages[j] * draws[i, j] + rewards_systematic[j] - wages[j]
            else:
                rew_ex = rewards_systematic[j] + draws[i, j]

            cont_value_ = rew_ex + delta * emaxs[j]

            if j == 2 and max_education:
                cont_value_ += INADMISSIBILITY_PENALTY

            cont_value[j, i] = cont_value_
```

## Implementation

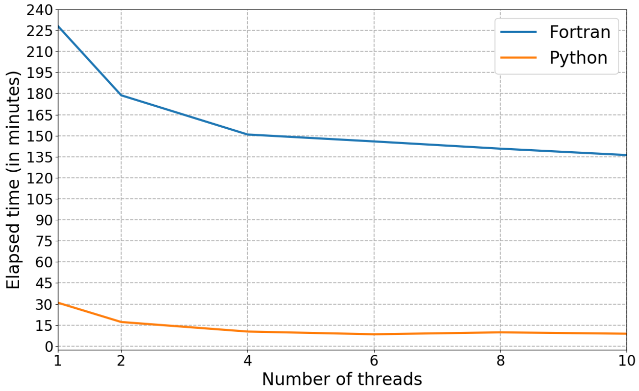

Figure: Scalability of Fortran and Python implementation

*The Fortran implementation uses OpenMP, Python Numba to parallelize the Monte Carlo integration.*

## Conclusion and Future Work

## Conclusion and Future Work

- Depending on the application, Python offers ways to overcome the inherent limitations of a dynamically typed language.
- Numba allows to have performance similar to Fortran, C, C++ in functions.
- The main bottleneck is development time not runtime.
- You can contribute to projects under `https://github.com/OpenSourceEconomics`.

## Bibliography

## References

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

[2] 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.

[3] 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.