# Replication Package for "Identification and Estimation of Continuous Time Dynamic Discrete Choice Games"

Jason R. Blevins

## Overview

This package contains the replication code for "Identification and Estimation of Continuous Time Dynamic Discrete Choice Games" by Jason R. Blevins. The code implements structural estimation and Monte Carlo experiments for two continuous-time dynamic discrete choice models analyzed in the paper. Replication instructions below are organized into two main sections corresponding to these two models.

The first model (`mc1p` directory) is a continuous-time single-agent renewal model inspired by Rust (1987) that analyzes optimal bus engine replacement decisions. There are two main programs: `rustct` carries out the estimation of the model with real data while `mc1p` carries out the Monte Carlo experiments.

The second model (`mcnp` directory) is a continuous-time quality ladder model based on Ericson and Pakes (1995) that analyzes oligopoly dynamics with entry, investment, and exit decisions. A single program, `mcnp`, carries out the Monte Carlo experiments.

## Data Availability

The Monte Carlo experiments in this paper do not involve analysis of external data. The data used are generated via simulation in the code itself.

The NFXP data used in the empirical example is from Rust (1987). It is included in this replication package and is freely available under the MIT License.

### NFXP Data

The data for the Nested Fixed Point Algorithm (NFXP) (i.e., bus engine replacement data) used in the single-agent renewal model (`mc1p`) consists of odometer readings and dates of bus engine replacements for 162 buses in the fleet of the Madison Metropolitan Bus Company that were in operation during the period December 1974 to May 1985. This data was originally collected and used by Rust (1987). The data files represent different bus models/vintages and are provided in ASCII format. Each file is named by bus model (e.g., `d309.asc` for Davidson model 309 buses, `g870.asc` for Grumman model 870 buses).

The original data and detailed documentation can be downloaded from John Rust's website at https://editorialexpress.com/jrust/nfxp.html or from the Zenodo repository at https://doi.org/10.5281/zenodo.3374587. The data files are also included in this replication package for convenience.

Data files: `mc1p/data/*.asc`

---

# Computational Requirements

## Software Requirements

- Fortran 2008 compiler: GNU Fortran (gfortran) or Intel Fortran Compiler
- LAPACK and BLAS libraries (or Intel MKL as alternative)
- OpenMP support (included with above compilers)
- GNU Make
- Bash shell
- R (for statistical calculations in Table 2 generation)

The Makefiles in this project will use GNU Fortran by default. To use the Intel Fortran compiler, prefix the usual `make` command with `SYSTEM=intel`:

```
SYSTEM=intel make
```

## Installing Dependencies

On Debian/Ubuntu Linux:

```
sudo apt-get install gfortran liblapack-dev libblas-dev r-base
```

On CentOS/RHEL Linux:

```
sudo yum install gcc-gfortran lapack-devel blas-devel R
```

On macOS with Homebrew:

```
brew install gcc r
```

macOS provides BLAS and LAPACK through the Accelerate framework.

## Controlled Randomness

For reproducibility, random number generator seeds for data generation are set within the Monte Carlo experiment control programs: `mc1p/mc1p.f90` (line 130) and `mcnp/mcnp.f90` (line 128).

## Expected Numerical Variability

Small numerical differences may occur across different computing environments due to differences in compilers (e.g., GNU vs. Intel), compiler versions, CPU architecture (e.g., Intel x86 vs. ARM), and numerical libraries (e.g., BLAS and LAPACK). Even with fixed random number generator seeds, floating-point arithmetic and compiler optimizations can result in small variations across platforms in practice.

- Table 1 should replicate nearly exactly, as it consists of sample characteristics from fixed data.
- Table 2 may show small differences due to numerical optimization.
- Tables 3 and 5 contain Monte Carlo results that involve both random number generation and numerical optimization and are aggregated over many replications. These may exhibit small differences across computing platforms. Differences larger than 0.05 in either table would be unexpected.
- Table 4 should replicate exactly except for the "Obtain V" timing column, which will vary by system and is not intended to be replicated.

# List of Tables and Programs

The provided code reproduces all tables in the paper. Figures in the paper do not require code. The following table summarizes the complete list of tables and programs. Please see the detailed replication instructions in the table-specific sections that follow.

| Table | Replication Script | Description | Hardware Used | Runtime |
|---|---|---|---|---|
| Table 1 | `table_1_2.sh` | Rust (1987) Sample Characteristics | MacBook Pro (12 core M2 Max) | 30 sec |
| Table 2 | `table_1_2.sh` | Single Agent Estimates with Rust (1987) Data | MacBook Pro (12 core M2 Max) | 30 sec |
| Table 3 | `table_3.sh` | Single Agent Monte Carlo Results | MacBook Pro (12 core M2 Max) | 10 min |
| Table 4 | `table_4.sh` | Quality Ladder Monte Carlo Specifications | Mac Pro (28 core Intel Xeon W) | 20 hr |
| Table 5 | `table_5.sh --partial` | Quality Ladder Monte Carlo (Partial Replication) | HPC Cluster (48 core Intel Xeon) | 10 hr |
| Table 5 | `table_5.sh --full` | Quality Ladder Monte Carlo (Full Replication) | HPC Cluster (48 core Intel Xeon) | 5 days |

Notes:

- Runtimes in the table above are for replicating complete tables, taking advantage of parallelism on the hardware shown.
- Actual runtime may vary significantly by Fortran compiler used, hardware capabilities, and the level of multiprocessing used.
- Table 5 is very computationally intensive to fully replicate. Please review the detailed replication instructions below for details and recommendations.

---

# Single-Agent Renewal Model (Tables 1-3)

## Description of Code

All code for the single-agent renewal model results resides in the `mc1p/` subdirectory.

Main Programs:

- `rustct.f90`: Structural estimation using bus engine replacement data.
- `mc1p.f90`: Monte Carlo experiment control program.

Core Modules:

- `rust_model.f90`: Single-agent continuous-time dynamic discrete choice model.
- `rust_data.f90`: Data loading and processing for bus fleet data.
- `dataset.f90`: Underlying data structure.

Libraries:

- `lbfgsb/`: L-BFGS-B optimization routines.
- `expokit/`: Expokit for dense Matrix exponential calculations.

Control Files:

- `control/mc-<delta>-<nm>.ctl`: Monte Carlo experiment control files.

Automated replication scripts:

- `table_1_2.sh`: Replicates Tables 1 and 2.
- `table_3.sh`: Replicates Table 3.

## Tables 1 and 2: Single-Agent Model Empirical Results

- **Estimated time:** 30 seconds on a modern laptop.
- **Hardware used:** 2023 MacBook Pro, 12-core M2 Max processor.
- **Software used:** MacOS 15.7.1 and GNU Fortran 15.2.0.
- **Outputs:** `mc1p/results/table_1.tex` and `mc1p/results/table_2.tex`.

To automatically generate both Tables 1 and 2:

```
cd mc1p
./table_1_2.sh
```

This script:

- Builds binaries for three model variants using conditional compilation:
  - `rustct-abbe`: Fixed $\lambda$ specification
  - `rustct-homogeneous`: Homogeneous $\lambda$ specification
  - `rustct-heterogeneous`: Heterogeneous $\lambda$ specification
- Runs each model variant using the Rust (1987) bus data and saves log files:
  - `./rustct-abbe > logs/rustct-abbe.log`
  - `./rustct-homogeneous > logs/rustct-homogeneous.log`
  - `./rustct-heterogeneous > logs/rustct-heterogeneous.log`
- Extracts appropriate results from the log files
- Computes the LR statistics and p-values.
  - This step requires `Rscript`. If it is not available, p-values will be omitted from the table.
- Generates `mc1p/results/table_1.tex` (sample characteristics)
- Generates `mc1p/results/table_2.tex` (parameter estimates and heterogeneity tests)

You may wish to manually compile the binaries (`rustct-abbe`, `rustct-homogeneous`, and `rustct-heterogeneous`) used to produce these results (for example, to change the Fortran compiler used). In that case, you can use `make` to build the programs directly before running the automated replication script:

```
cd mc1p

# Default settings (GNU Fortran)
make rustct-abbe rustct-homogeneous rustct-heterogeneous

# Use Intel Fortran compiler
SYSTEM=intel make rustct-abbe rustct-homogeneous rustct-heterogeneous
```

## Table 3: Single-Agent Model Monte Carlo Experiments

- **Estimated time:** 10 minutes on a modern laptop.
- **Hardware used:** 2023 MacBook Pro, 12-core M2 Max processor.
- **Software used:** MacOS 15.7.1 and GNU Fortran 15.2.0.
- **Output:** `mc1p/results/table_3.tex`.

To execute all the single-agent model Monte Carlo experiments reported in Table 3, you can use the `table_3.sh` script:

```
cd mc1p
./table_3.sh
```

This script:

- Compiles the `mc1p` binary used for the Monte Carlo experiments.
- Runs each of the specifications reported in the paper and saves individual log files:
  - `./mc1p control/mc-<delta>-<nm>.ctl > logs/mc-<delta>-<nm>.log`
  - For each specification, by default all cores on your machine will be used to carry out Monte Carlo replications in parallel.

- Collects results from each specification and produces Table 3.
- Saves the complete LaTeX table to `mc1p/results/table_3.tex`
- Displays the results to the console.

Alternatively, you can manually compile the program and run individual specifications by using the appropriate control file. Individual control files are named as `mc-<delta>-<nm>.ctl` where `delta` is the observation time interval (`0.00` for continuous time) and `nm` is the number of markets simulated. For example:

```
cd mc1p
make mc1p
./mc1p control/mc-1.00-3200.ctl
```

The `mc1p` program supports OpenMP parallelization, executing individual Monte Carlo replications in parallel across threads. To explicitly set the number of threads, set the `OMP_NUM_THREADS` environment variable before running either `table_3.sh` or `mc1p`:

```
export OMP_NUM_THREADS=8   # Adjust number of cores for your system
./table_3.sh
```

---

# Quality Ladder Model (Tables 4-5)

The quality ladder model results are much more computationally demanding to reproduce than those for the single-agent model. In particular, the Monte Carlo experiments involve repeatedly solving for the equilibrium of a complex dynamic game with many firms and computing the matrix exponential for a large matrix to calculate the log likelihood function for each trial parameter value during the optimization. This becomes feasible due to the computational benefits of continuous time games as well as the use of high performance, vectorized Fortran code.

## Description of Code

All code for the quality ladder model results resides in the `mcnp/` subdirectory.

Main Program:

- `mcnp.f90`: Monte Carlo experiment control program.

Core Modules:

- `model.f90`: Continuous-time dynamic oligopoly (quality ladder) model with entry, exit, and investment.
- `encoding.f90`: Efficient state space encoding/decoding.
- `dataset.f90`: Data structure for storing simulated observations.
- `sparse.f90`: Sparse matrix operations for computational efficiency.

Libraries:

- `lbfgsb/`: L-BFGS-B optimization routines.

Control Files:

- `control/example.ctl`: A small-scale example control file for testing on a standard laptop or desktop computer.
- `control/mc-<nn>-<delta>.ctl`: Monte Carlo simulation control files for different numbers of firms `nn` and data sampling interval `delta`.
- `control/time-<nn>.ctl`: Timing control files for different numbers of firms `nn`.

Automated replication scripts:

- `table_4.sh`: Replicates Table 4.

- `table_5.sh`: Replicates Table 5.
- `make_timing_table.sh`: Helper script for Table 4 that collects results and produces the LaTeX table.
- `run_parallel.sh`: Helper script for Table 5 that handles automatic distribution of Monte Carlo trials across parallel processes.
- `table_5_save.sh`: Helper script for Table 5 that collects results from Monte Carlo replications and creates LaTeX table.

## Quality Ladder Model: Table 4 (Timing Results)

- **Estimated time:** 20 hours on a modern, multicore workstation.
- **Hardware used:** 2019 Mac Pro with 28-Core Intel Xeon W 2.5 GHz CPU, 96 GB RAM.
- **Software used:** macOS 12.6 and GNU Fortran 12.2.0.
- **Output:** `mcnp/results/table_4.tex`.
- **Note:** The "Obtain V" column reports wall clock times and is not intended to be replicated exactly: timing will depend on system characteristics.

To produce the complete Table 4 automatically:

```
cd mcnp
./table_4.sh
```

This script builds the `mcnp` binary, runs timing experiments for N=2 through N=30 firms using the corresponding control files (e.g., `./mcnp control/time-02.ctl`), saves log files for each specification, extracts model size and timing information from the output, and produces Table 4 in LaTeX format at `mcnp/results/table_4.tex`.

## Quality Ladder Model: Table 5 (Monte Carlo Results)

Table 5 is the most computationally intensive to replicate. The results in the paper were carried out in parallel on a high performance computing (HPC) cluster over approximately 5 days. The cluster node used has 48 Intel Xeon Platinum 8260 2.40GHz CPU cores and 192 GB memory.

Due to the high computational requirements, we provide three distinct replication approaches depending on your computational resources and time constraints:

1. **Full Replication**: Complete reproduction with 100 Monte Carlo trials.
2. **Incremental Replication**: Run selected specifications individually.
3. **Partial Replication**: Faster validation with 10 trials instead of 100.

## Multithreading and Parallel Processing

To reduce the computational time required, we recommend carrying out these experiments using nested parallelism:

- Run multiple parallel Monte Carlo experiments (*processes*)
- Each experiment uses multi-threading (*threads*)
- Total number of cores used concurrently: $total = processes \times threads$

At the inner level of nesting, the `mcnp` program uses OpenMP with multiple *threads* within each simulation to accelerate computation of the vectorized value function, log likelihood function, etc.

At the outer level of nesting, if you have many cores available the `table_5.sh` script will automatically distribute Monte Carlo trials across multiple *processes*, based on the recommended number of Threads Per Trial in the table below. Each specification below corresponds to one row of Table 5.

| N | Sampling | Threads Per Trial | Time Per Trial |
|---|---|---|---|
| 2 | Continuous | 2 | 6 sec |
| | $\Delta = 1.0$ | 2 | 12 sec |

| N | Sampling | Threads Per Trial | Time Per Trial |
|---|---|---|---|
| 4 | Continuous | 4 | 2 min |
| | $\Delta = 1.0$ | 4 | 4 min |
| 6 | Continuous | 4 | 16 min |
| | $\Delta = 1.0$ | 4 | 32 min |
| 8 | Continuous | 8 | 1 hr |
| | $\Delta = 1.0$ | 8 | 4 hr |

*Example:* On the HPC cluster node we used, the final table row with $N = 8$ and $\Delta = 1.0$ required approximately 4 hours per Monte Carlo trial and a full replication requires 100 trials. With 48 cores at our disposal, we executed trials concurrently using 6 parallel *processes* each with 8 *threads* for parallel processing within a trial.

**Approach 1: Full Replication (100 Monte Carlo Trials Per Specification)**

- **Estimated time:** 5 days on a 48-core HPC cluster node.
- **Hardware used:** HPC cluster node with Intel Xeon Platinum 8260 2.40 GHz CPUs (48-cores) and 192 GB memory.
- **Software used:** RHEL 8.10 and Intel Fortran compiler 2024.2.0.
- **Output:** `mcnp/results/table_5.tex`.

This approach reproduces all results as reported in the paper:

```
cd mcnp

# Run all specifications with 100 Monte Carlo trials each
./table_5.sh --full
```

The script automatically:

- Builds the `mcnp` binary used for the Monte Carlo specifications.
- Runs all 8 specifications (2, 4, 6, 8 firms with both continuous and discrete sampling).
  - Carries out each specification by running `mcnp` with the appropriate control files and saves log files.
  - Carries out 100 Monte Carlo trials per specification.
  - Uses parallel execution of trials across multiple processes with recommended thread allocation per process (handled internally by `run_parallel.sh`).
  - Each parallel process will use its own log file.
  - Skips already-completed specifications (resumes after interruption).
- Collects all results and generates the LaTeX table at `results/table_5.tex`.

To check status of completed specifications:

```
./table_5.sh --full --status
```

To resume interrupted runs:

```
# Resumes automatically, skipping completed specifications
./table_5.sh --full
```

By default, the number of CPU cores in your system is automatically detected by `table_5.sh`. To override this, set the `OMP_NUM_PROCS` environment variable before running the script. Use your physical core count, not hyperthreaded logical cores. To check: `lscpu` on Linux or `sysctl -n hw.physicalcpu` on macOS.

```
# Optional: Set number of physical cores (auto-detected if not set)
export OMP_NUM_PROCS=48
```

Finally, we note that the `table_5.sh` script will automatically compile the `mcnp` binary using GNU Fortran, but if you wish to manually build it you can use `make` like so. For example, on our HPC cluster, we used the Intel Fortran compiler:

```
cd mcnp
make clean
SYSTEM=intel make
```

**Approach 2: Incremental Replication (Selected Specifications)**

- **Requirements:** HPC cluster or high-end, multicore workstation with at least 32 GB RAM
- **Estimated time:** Variable, depending on hardware and selection

The `table_5.sh` script provides an `--experiment` command line option to specify an individual experiment specification to run. Adding `--full` as above ensures that all 100 Monte Carlo replications will be completed.

This approach allows you to, for example, validate that the code works correctly using smaller specifications. You can also use this approach to distribute computation across multiple machines or sessions and collect the results at the end.

```
cd mcnp

# Choose which specific specifications to run individually.
# For convenience, all 8 specifications are listed here:
./table_5.sh --full --experiment mc-02-0.0
./table_5.sh --full --experiment mc-02-1.0
./table_5.sh --full --experiment mc-04-0.0
./table_5.sh --full --experiment mc-04-1.0
./table_5.sh --full --experiment mc-06-0.0
./table_5.sh --full --experiment mc-06-1.0
./table_5.sh --full --experiment mc-08-0.0
./table_5.sh --full --experiment mc-08-1.0

# Generate LaTeX table from completed specifications only
./table_5.sh --full --save
```

**Approach 3: Partial Replication (10 Monte Carlo Trials Per Specification)**

- **Requirements:** HPC cluster or high-end, multicore workstation with at least 32 GB RAM
- **Estimated time:** 10 hours on 48-core HPC cluster node
- **Hardware used:** HPC cluster node with Intel Xeon Platinum 8260 2.40 GHz CPUs (48-cores) and 192 GB memory.
- **Software used:** RHEL 8.10 and Intel Fortran compiler 2024.2.0.
- **Output:** `mcnp/results/table_5_partial.tex`.

This uses separate control files (`control/mc-*-partial.ctl`) which carry out only the first 10 Monte Carlo trials instead of the full 100 trials reported in the paper.

```
cd mcnp

# Run all specifications with only 10 trials each
./table_5.sh --partial
```

The following table displays the expected partial replication results corresponding to Table 5 in the paper. It shows the means and standard deviations of the parameter estimates for only the first 10 of the full 100 Monte Carlo trials reported in the paper.

**Table: Quality Ladder Model Monte Carlo Results (Partial Replication)**

| N | K | Sampling | | $\lambda_L$ | $\lambda_H$ | $\gamma$ | $\kappa$ | $\eta$ | $\mu$ |
|---|---|---|---|---|---|---|---|---|---|
| | | DGP | True | 1.000 | 1.200 | 0.400 | 0.800 | 4.000 | 0.900 |
| 2 | 56 | Continuous | Mean | 1.003 | 1.210 | 0.401 | 0.809 | 4.035 | 0.899 |
| | | | S.D. | 0.015 | 0.015 | 0.009 | 0.029 | 0.093 | 0.021 |
| | | $\Delta = 1.0$ | Mean | 1.120 | 1.320 | 0.399 | 0.939 | 4.398 | 0.950 |
| | | | S.D. | 0.198 | 0.205 | 0.007 | 0.276 | 0.723 | 0.073 |
| 4 | 840 | Continuous | Mean | 1.004 | 1.205 | 0.405 | 0.802 | 4.067 | 0.900 |
| | | | S.D. | 0.017 | 0.022 | 0.014 | 0.052 | 0.258 | 0.033 |
| | | $\Delta = 1.0$ | Mean | 0.952 | 1.147 | 0.400 | 0.692 | 3.796 | 0.891 |
| | | | S.D. | 0.095 | 0.092 | 0.006 | 0.151 | 0.372 | 0.034 |
| 6 | 5,544 | Continuous | Mean | 1.003 | 1.198 | 0.409 | 0.791 | 4.017 | 0.900 |
| | | | S.D. | 0.015 | 0.023 | 0.024 | 0.034 | 0.200 | 0.022 |
| | | $\Delta = 1.0$ | Mean | 0.990 | 1.191 | 0.398 | 0.780 | 3.934 | 0.898 |
| | | | S.D. | 0.084 | 0.086 | 0.006 | 0.143 | 0.335 | 0.025 |
| 8 | 24,024 | Continuous | Mean | 1.001 | 1.207 | 0.412 | 0.805 | 4.013 | 0.898 |
| | | | S.D. | 0.015 | 0.021 | 0.019 | 0.024 | 0.179 | 0.021 |
| | | $\Delta = 1.0$ | Mean | 1.003 | 1.204 | 0.399 | 0.792 | 3.937 | 0.905 |
| | | | S.D. | 0.131 | 0.131 | 0.004 | 0.172 | 0.377 | 0.052 |

---

## All Tables: Automated Full Replication

For users with access to HPC resources, a `main.sh` script is provided that automates the complete replication of all tables:

```
./main.sh
```

This script runs each table in order (Tables 1-5), with parallelization within each table as appropriate.

*Important:* Given the computational requirements, particularly for Table 5, this script should only be used on HPC systems with substantial resources (32+ GB RAM, 40+ cores).

For most users, we recommend running the individual table scripts separately as described above, which provides more flexibility and control over the replication process.

## References


- Byrd, R. H., P. Lu, and J. Nocedal (1995). A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific and Statistical Computing* 16, 1190–1208.
- Ericson, R. and A. Pakes (1995). Markov-Perfect Industry Dynamics: A Framework for Empirical Work. *Review of Economic Studies* 62, 53–82.
- Rust, J. (1987). Optimal Replacement of GMC Bus Engines: An Empirical Model of Harold Zurcher. *Econometrica* 55, 999–1033.
- Sidje, R. B. (1998). Expokit: A software package for computing matrix exponentials. *ACM Transactions on Mathematical Software* 24, 130–156.
- Zhu, C., R. H. Byrd, P. Lu, and J. Nocedal (1997). Algorithm 778: L-BFGS-B, FORTRAN routines for large scale bound constrained optimization. *ACM Transactions on Mathematical Software* 23, 550-560.


## License

The L-BFGS-B license is provided in the `mc1p/lbfgsb/` directory: License.txt.

The Expokit license is provided in the `mc1p/expokit/` directory: LICENSE.