README Document for Reproducing Results in "Revealed Preferences with Measurement Error"

Victor H. Aguiar vaguiar@uwo.ca

Nail Kashaev* nkashaev@uwo.ca

August 4, 2020

1. Data Sources

The paper uses two data sets:

- Adams, A., Cherchye, L., De Rock, B., & Verriest, E. (2014). Replication data for: Consume Now or Later? Time Inconsistency, Collective Choice, and Revealed Preference (Version 1) [Data set]. ICPSR - Interuniversity Consortium for Political and Social Research. https://doi.org/10.3886/E112718V1
- Ahn, D., Choi, S., Gale, D., & Kariv, S. (2014). Estimating ambiguity aversion in a portfolio choice experiment. Quantitative Economics, 5(2), 195-223.[Replication files] https://doi.org/10.3982/QE243

Both data sets were transformed to csv format. The first data set was split into two tables. The first table contains information about single-individual households. The second table contains information about couples' households.

2. Software

First Application, Appendices B,E, and F

A version 1.1.1 of the `Julia` programming language was used in coding the analysis files. For details about how to install `Julia` on different platforms and make `Julia` programs exe-

*Department of Economics, University of Western Ontario.

cutable from the command line see https://julialang.org/downloads/platform/. After installation of `Julia 1.1.1`. copy `ReplicationAK/Julia_environment/1.1.1/Manifest.toml` and `ReplicationAK/Julia_environment/1.1.1/Project.toml` files to `.julia/environments/v1.1`. Next run `using Pkg` and `Pkg.instantiate()` in the `Julia` terminal.

Second Application

Our second application (Section 8) requires the following software environment.

(i) Install `R 3.4.4`.

- Install `revealedPrefs` package from CRAN in `R 3.4.4`. (`install.packages("revealedPrefs")`).
- Unzip the `revealedPrefsmod.zip` package in `library` for your `R` installation (next to where the `revealedPrefs` was installed, e.g. `C:/Users/Elvis/Documents/R/R-3.4.4/library`).
- Alternatively, instead of the previous step, you can compile from the source `revealedPrefsmod.tar` file: `install.packages("revealedPrefsmod.tar",repos=NULL,type="source")`. This source file has the code for the functions that we use to simulate the draws of the true consumption and true price. These files have been written on the basis of the files in `revealedPrefs`. The files were modified under the terms of the GNU public license.
- Functions in `revealedPrefsmod.tar`:
  - `simul.cpp` – It simulates prices and quantities from a uniform distribution good by good. Copyright 2014 Julien Boelaert.
  - `simulprice.cpp` – modification of `SimGarp`. It takes prices as given and generates quantities.
  - `simulpricewealth.cpp` – modification of `SimGarp`. It takes prices and expenditures and generates quantities that match them.
  - `simulquantity.cpp` – modification of `SimGarp`. It takes quantities as given and generates prices.
  - `simulquantitywealth.cpp` – modification of `SimGarp`. It takes quantities and prices as given and generates prices.

(ii) Install `Julia 1.1.0`

- After installation of `Julia 1.1.0` copy `ReplicationAK/julia_environment/1.1.0/Manifest.toml` and `ReplicationAK/julia_environment/1.1.0/Project.toml` files to `.julia/environments/v1.1`. Next run `using Pkg` and `Pkg.instantiate()` in the `Julia` terminal.

- `Pkg.instantiate()` must be executed after all `R` packages are installed.
- The Julia package RCall may cause trouble in some systems with more than one `R` installation. Fix the `ENV["R_HOME"]` to the desired folder in those cases. Additional details of this are provided in the main files of the second application.

3. Hardware

Different parts of the code were run on different machines. In the description of the files below we will refer to the machine that was used and the approximate execution times. The following machines were used:

- PC1
  - Intel(R) Core(TM) i7-6700 CPU @ 3.40 GHz, 64 GB of RAM, Windows 10 Enterprise.
  - GPU: NVIDIA GeForce GTX 950, CUDA v10.1.
- PC2
  - Intel(R) Xeon(R) Silver 4110 CPU @ 2.10 GHz (2 processors), 64 GB of RAM, Windows 10 Enterprise.
  - GPU: NVIDIA Quadro P5000, CUDA v10.1.
- PC3
  - Intel(R) Core(TM) i7-9800X CPU @ 3.80 GHz, 128 GB of RAM, Windows 10 Enterprise.
  - GPU: NVIDIA Tesla V100-PCIE-16GB, CUDA v10.1.
- Cluster
  - Niagara – the Compute Canada cluster Niagara by the University of Toronto. Specifications: https://docs.computecanada.ca/wiki/Niagara.

4. Content

- `Appendix_B` – the folder contains the analysis files to replicate the results in Appendix B.
- `Appendix_E` – the folder contains the analysis files to replicate the results in Appendix E.

- `Appendix_F` – the folder contains the analysis files to replicate the results in Appendix F.
- `Data_all` – all data files used in the paper.
- `Deterministic_test` – the folder contains the analysis files to replicate the results of the deterministic RP tests in Sections 7 and 8.
- `FirstApp` – the folder contains the analysis files to replicate the results in Section 7.1 (including footnotes).
- `Julia_environment` – the folder contains toml files with all necessary `Julia` packages.
- `Output_all` – all reproducible outputs reported in the paper.
- `SecondApp` – the folder contains the analysis files to replicate the results in Section 8 (including footnotes).

Below we describe the content of every folder except `Julia_environment` and `Output_all`.

Appendix_B

Rootfiles

`B_tables_all.jl` – this code creates all tables in Appendix B (Tables 2-4).

Appendix_B1

- `B1_dgpX_10k_2000.jl` – the code applies our test to simulated data. The data is generated according to DGP`X`, where `X` equals 1 or 2. These results are used in Table 2.

  Machine/Execution time: PC3 / 72h per DGP.

  Output: `B1_dgpX_chain_(0,10000).sample_2000.csv`.

- `B1_EDdettest_DGP12.jl` – the code applies the deterministic test to simulated data. The data is generated according to DGP1 and DGP2 (See Appendix B1 in the paper). These results are used in Table 2.

  Machine/Execution time: PC3 / 0.5h.

  Output: `B1_EDdettest_rr_dgp1.csv`, `B1_EDdettest_rr_dgp2.csv`, and `B1_EDdettest_arr_dgp12.csv`.

- `dgp12.jl` – this function generates data according to DGP1 and DGP2.

Appendix_B2

- `B2_dgpX_10k_Y.jl` – the code applies our test to simulated data. The data is generated according to DGP`X`, where `X` equals 3 or 4. Sample size`Y` is equal either 2000 or 3000. These results are used in Table 3.

  Machine/Execution time: PC3/ 96h per DGP.

  Output: `B2_dgpX_chain_(0,10000).sample_Y.csv`.

Appendix_B3

- `B3_dgpX_5k_2000.jl` – the code applies our test to simulated data where the MCMC chain has length 5000. The data is generated according to DGP`X`, where `X` equals 2,3, or 4. These results are used in Table 4.

  Machine/Execution time: PC2 / 48h per DGP.

  Output: `B3_dgpX_chain_(0,5000).sample_2000.csv`.

cpufunctions

This folder contains a function used in the Appendix B that does not use CUDA.

- `myfun.jl` – the moment function for testing the ED-model.

cudafunctions

This folder contains functions and modules used in the first application that use CUDA.

- `cuda_chainfun.jl` – this code generates the MCMC chain for testing the ED-model.
- `cuda_fastoptim.jl` – this code constructs the objective function for testing the ED-model.

Appendix_E

Appendix_E1

Rootfiles.–

- `E1_main.jl` – the code tests for s/ED-IU-rationalizability using data from couples' households.

Machine/Execution time: PC3 / 2h.
Output: `E1_TS.csv`.

cpufunctions.– This folder contains functions and modules used in Appendix E1 that do not use CUDA.

- `ED_data_load.jl` – this function loads the data.
- `myfun_IU_meandisc.jl` – the moment function for testing the IU-ED-model.
- `warm_start_searchdelta_justcvex_IU.jl` – this function generates the initial element of the MCMC chain for testing the IU-ED-model.

cudafunctions.– This folder contains functions and modules used in Appendix E1 that use CUDA.

- `cuda_chainfun_IU_meandisc.jl` – this code generates the MCMC chain for testing the IU-ED-model.
- `cuda_fastoptim_IU_counter.jl` – this code constructs the objective function for testing the IU-ED-model.

`Appendix_E2`

Rootfiles.–

- `E2_main.jl` – the code tests for the collective model using data from couples' households.

Machine/Execution time: PC1 / 1.3h.
Output: `E2_TS.csv`.

cpufunctions.– This folder contains functions and modules used in Appendix E2 that do not use CUDA.

- `ED_data_load.jl` – this function loads the data.
- `myfun_collective.jl` – the moment function for testing the collective model.
- `warm_start_searchdelta_justcvexcollective.jl` – this function generates the initial element of the MCMC chain for testing the collective model.

cudafunctions.– This folder contains functions and modules used in Appendix E2 that use CUDA.

- `cuda_chainfun_collective.jl` – this code generates the MCMC chain for testing the collective model.
- `cuda_fastoptim.jl` – this code constructs the objective function for testing the collective model.

Appendix_F

Rootfiles

- `F_figure1.R` – this file generates Figure 1 in Appendix E.
- `F_main_shell.jl` – the code computes the values of the test statistic for testing different counterfactual values. This is an interactive Julia code that allows to test any point over the grid of parameters. We run this file in a Powershell loop – loop.ps1.

Machine/Execution time: PC3 / 48h.
Output:
`F_X._0.975.csv`, where `X` is in $\{1.0, 1.01, 1.02, \ldots, 1.1\}$ is the value of $1 + \kappa$.

cpufunctions

This folder contains functions and modules used in Appendix F that do not use CUDA.

- `myfun_counter.jl` – the moment function for testing the counterfactual model.
- `warm_start_searchdelta_justcvex.jl` – this function generates the initial element of the MCMC.

cudafunctions

This folder contains functions and modules used in Appendix F that use CUDA.

- `cuda_chainfun.jl` – this code generates the MCMC chain.
- `cuda_fastoptim_counter.jl` – this code constructs the objective function for testing the counterfactual model.

Data_all

- `cve.csv` – consumption data for the first application (single-individual households).
- `cvecouple.csv` – consumption data for the first application (couples' households).
- `p.csv` – price data for the first application (single-individual households).
- `pcouple.csv` – price data for the first application (couples' households).
- `rationalitydata3goods.csv` – data for the second application.
- `rv.csv` – interest rate data for the first application (single-individual households).
- `rvcouple.csv` – interest rate data for the first application (couples' households).

FirstApp

couples

This folder contains programs generating the output in Section 7.1 for couples' households (including footnotes).

- `1App_couples_main_X.jl` – the code generates the value of the test statistic using data on couples' households when the discount factor is supported on or inside [`X`,1]. For instance, `1App_couples_main_0.1.jl` generates the value of the test statistic when the discount factor is supported on or inside $[0.1, 1]$. These results are used in the main text and in footnote 50.

Machine/Execution time: PC1 / 1h per file.
Output: `1App_couples_main_X.csv`.

cpufunctions

This folder contains functions and modules used in the first application that do not use CUDA.

- `ED_data_load.jl` – this function loads the data.
- `ED_det_test.jl` – this function computes the rejection rate of the deterministic test of the ED-model.
- `myfun_recoverdelta.jl` – the moment function for constructing the confidence set for the average discount factor.

- `myfun.jl` – the moment function for testing the ED-model.
- `warm_start_searchdelta_justcvex_delta.jl` – this function generates the initial element of the MCMC chain for constructing the confidence set for the average discount factor.
- `warm_start_searchdelta_justcvex.jl` – this function generates the initial element of the MCMC chain for testing the ED-model.

`cudafunctions`

This folder contains functions and modules used in the first application that use CUDA.

- `cuda_chainfun_delta.jl` – this code generates the MCMC chain for constructing the confidence set for the average discount factor.
- `cuda_chainfun.jl` – this code generates the MCMC chain for testing the ED-model.
- `cuda_fastoptim_counter.jl` – this code constructs the objective function for computing the confidence set for the average discount factor.
- `cuda_fastoptim.jl` – this code constructs the objective function for testing the ED-model.

`deterministic_test`

- `1App_dt.jl` – the code applies the deterministic test to the survey data set. These results are used in the main text of Section 7.1.

Machine/Execution time: PC1 / 1 min.
Output: `1App_dt_rr.csv`.

`procedures`

This folder contains main programs used in the first application.

- `1App_main.jl` – generates the value of the test statistic for both single-individual and couples' households for a generic lover bound of the support of the discount factor.
- `1App_singles_ADF.jl` – generates the value of the test statistic for testing whether an average discount factor equals to a given value.

singles

This folder contains programs generating the output in Section 7.1 for single-individual households (including footnotes).

- `1App_singles_ADF_X.jl` – the code computes the value of the test statistic for testing the hypothesis that the average discount factor is equal to `X`. These results are used footnote 48.

  Machine/Execution time: PC1 / 40min per file.

  Output: `1App_singles_ADF_X.csv`.

- `1App_singles_main_X.jl` – the code generates the value of the test statistic using data on single-individual households when the discount factor is supported on or inside [`X`,1]. For instance, `1App_singles_main_0.1.jl` generates the value of the test statistic when the discount factor is supported on or inside $[0.1, 1]$. These results are used in the main text and in footnote 49.

  Machine/Execution time: PC1 / 0.5h per file.

  Output: `1App_singles_main_X.X.csv`.

SecondApp

deterministic_test

- `2App_dt.jl` – the code applies the deterministic test to the experimental data set. These results are used in the main text of Section 7.2.

Machine/Execution time: PC1 / 1 min.
Output: `2App_dt_rr.csv`.

pricemisperception

Programs generating the output when there is price misperception (Section 8).

- `2App_pm_900.jl` – the code generates the value of the test statistic for the model with price misperception. These results are used in the main text of Section 7.2.

Machine/Execution time: Cluster / 12h.
Output: `2App_pm_reps_900.csv`.

secondappfunctions

This folder contains functions and modules used in the second application.

- `guessfun_price.jl` – this code generates the initial draw of the Montecarlo step for price misperception.
- `guessfun_quantity.jl` – this code generates the initial draw of the Montecarlo step for trembling hand.
- `jumpfun_price.jl` – this function will draw new candidates for the Montecarlo. For this particular application this is the same as the guessfun_price.jl.
- `jumpfun_quantity.jl` – this function will draw new candidates for the Montecarlo. For this particular application this is the same as the guessfun_quantity.jl.
- `myfun_pm.jl` – the moment condition for price misperception.
- `myfun_th.jl` – the moment condition for trembling hand.

tremblinghand

Programs generating the output when there is trembling-hand measurement error in consumption (Section 8).

- `2App_th_X.jl` – the code generates the value of the test statistic for testing the model with trembling-hand errors in consumption with `X` draws. For instance, `2App_th_900.jl` computes the value of the test statistic using 900 draws. These results are used in the main text of Section 7.2 and in footnote 57.

  Machine/Execution time: Cluster/ 6h, 11h, and 15h for 580 draws, 900 draws, and 2970 draws, respectively.

  Output: `2App_th_reps_X.csv`.
- `2App_th_main.jl` – this code is used in `2App_th_X.jl`.