Data and Computer Codes for

"The Spending and Debt Response to Minimum Wage Hikes"

Àpril 2012

1) The programs and datasets to replicate the CEX, CPS, SIPP and credit card results are located in the subdirectory "unix_files"

* AER_FINAL_PROGRAMS.zip contains all of the programs
    * The file run_programs.do in that folder will call all of the programs in the order they need to be run. The files that are commented out will not run but are included for completeness (the files that create our CEX extract from the CEX data cds and the files used to clean and analyze the credit card data).
* The main data extracts are the following:
    * ces_int_82_08.dta.gz
    * ogr_select.dta.gz
    * rep_sipp1.dta.gz
    * The programs to create the extracts are included, and presume that the rawer forms of the data have already been created.
* Path names may need to be changed.

2) The programs and datasets to replicate the Dynamic Programming are in the subdirectory "minwage"

* See below for step-by-step DP replication instructions

* If you copy this research.zip file directly to your C: drive and unzip (i.e. get C:/research/), all path names and file references for the dynamic programming part of the code should work without modification.

Dynamic Programming Code

1. Attached are the computer codes used to generate the dynamic programming model results in "The Consumption Response to Minimum Wage Hikes".

2. Because the programs in this package call each other and pass files between themselves, the programs will work only if directories are properly specified. If you copy *research.zip* to your C: drive and unzip, the references should all work. After unzipping, the first file to run in the dynamic programming problem (*minwageDP_1.gau*) should be located at C:/research/minwage/*minwageDP_1.gau*.

3. We use SCF data to generate the initial distribution of the state variables when simulating the model. Codes for merging the SCF data together are written in SAS, and these data are processed in SAS and Stata.

4. The codes solving for the decisions at preset points of the state variables are written in the C++ language. We use GAUSS to call the C programs, send them the necessary inputs - including parameter values and initial values of the state variables - and then run the simulations in GAUSS.

5. If you want to consider some of the alternative specifications described in the paper (Table 8), or create your own, consult the file "initializations.gau" along with the "Dynamic Programming" section below.

Calculating Initial Distribution For Simulation

We use SCF data to set the initial distribution of the state variables for the simulation. There are multiple steps to retrieving the raw data (see Appendix 2), but the final step is running *scf.do* in C:/research/minwage/calculatingInitDistForSim, which creates *scf.out ; scf.out*
gets read in during the simulation (see *minwage_Sim_Graph_f.gau*; this file contains the simulation procedure and gets called by *minwageSimmerGrapher.gau*).

Dynamic Programming

The codes for the dynamic programming, simulation, and final output are in the C:\research\minwage directory. Steps for running them are outlined in the next page. Further detail can be found in the Appendix of this file or in comments within the code files themselves.

Step1: Calculating Optimal Decision Rules

We simulate minimum wage hikes at 3 different ages: 22, 34, and 50, and also simulate the model with no minimum wage hike. Since we find decision rules for each of the ages, and for the no hike case, any set of parameters takes 4 runs to implement. Parameters (including utility, technology, and grid discretization parameters) are set in *initializations.gau*. *MinwageDP_1.gau, minwageDP_2.gau, minwageDP_3.gau, minwageDP_4.gau* read in the settings described in *initializations.gau,* then calculate decision rules for the case of minimum wage hikes at age 22, 34, 50, and for no minimum wage hike, respectively: you need to run all four files. *minwageSimmerGrapher.gau* simulates the model for the case of minimum wage hikes at age 22, 34, 50, and for no minimum wage hike (you need to run the file four times), and graphs the simulated data.

In order to keep track of the parameters in different specifications of the model and also to keep track of the output of these specifications, there is a parameter initialized in *MinwageDP_1.gau, minwageDP_2.gau, minwageDP_3.gau, minwageDP_4.gau* called *settingsSwitch* (of the form ##.1, ##.2, or ##.3. For the baseline specification it is 67.1, 67.2, or 67.3 for each age of the wage hike respectively. For the adjustment cost specification, it is 65.1, 65.2, 65.3. See **Table C**). *settingsSwitch* refers to a given set of parameters in a specification (including utility, technology, and grid discretization parameters), defined in *initializations.gau*. The files (*minwageDP_1.gau,..,minwageDP_4.gau)* call the procedure `set_paramsAndGrids` which is also in the file *initializations.gau*. The two key parameters that *MinwageDP_1.gau-minwageDP_4.gau* pass into `set_paramsAndGrids` are `settingsSwitch` and `hikeSwitch` (`hikeSwitch` indicates whether we turn on the hike). `set_paramsAndGrids` then passes back the key utility, technology, and grid size parameters to the decision rules code (*minwageDP_1.gau,..,minwageDP_4.gau)* (see the line of code with "`set_paramsAndGrids(settingsSwitch,hikeSwitch)`"). Note: The only parameters that differ between *settingsSwitch* =##.1, *settingsSwitch* =##.2, *settingsSwitch* =##.3 are *hikeStart* (the time of the hike in number of qtrs after age 18) and *wageIncr* (the income growth rate caused by the hike – calibrated to give $250 [for each age receiving a wage hike]).

For example: in the baseline case *settingsSwitch* =67.1 and *hikeSwitch=1* in the file m*inwageDP_1.gau; settingsSwitch* =67.2 and *hikeSwitch=1* in the file m*inwageDP_2.gau; settingsSwitch* =67.3 and *hikeSwitch=1* in the file m*inwageDP_3.gau; settingsSwitch* =67.1 and *hikeSwitch*=0 in the file m*inwageDP_4.gau.* If you were to change to the no borrowing against durables case (ie, downpaymentR=1), all the "67"s would be changed to "72", so *settingsSwitch* =72.1 and *hikeSwitch=1* in the file m*inwageDP_1.gau; settingsSwitch* =72.2 and *hikeSwitch=1* in the file m*inwageDP_2.gau; settingsSwitch* =72.3 and *hikeSwitch=1* in the file *minwageDP_3.gau; settingsSwitch* =72.1 and *hikeSwitch*=0 in the file *minwageDP_4.gau.*

Step2: Move output to a new folder

MinwageDP_1.gau outputs decisions rules to C:/research/minwage/results/**newResults_tmp_1** by default [*minwageDP_2.gau* outputs decisions rules to C:/research/minwage/results/**newResults_tmp_2**, etc]. After *minwageDP_1.gau* finishes running, you should create a new folder in C:/research/minwage/results, and copy all output from C:/research/minwage/results/**newResults_tmp_1** into it. This way, results don't get overwritten next time you run *minwageDP_1*. Specifically, for the baseline case, you should create a folder called

C:/research/minwage/results/**Results_671_hike**. This folder should include two subfolders named called "sims" and "graphs" to hold the simulations' matrix output and graph equivalents respectively. Do the same for the other 3 runs of the specification (e.g. for the baseline: newResults_tmp_2 → Resuilts_672_hike, newResults_tmp_3 → Results_673_hike, newResults_tmp_4 → Results_671_nohike).

In naming the folder [and the *settingsSwitch*], please follow the conventions in **Table A** so that subsequent programs (*minwageSimmerGrapher.gau, printMatrix.gau, printMedians.gau,* and *SS_all_regs.do*) run properly.

Table A

Naming Conventions for a new run number, ##

Program	*SettingsSwitch*	*hikeSwitch*	*hikeStart*	equivalent Age	Folder Name	Folder Name for baseline specification
minwageDP_1.gau	##.1	1	16	22	Results_##1_hike	C:/research/minwage/results/Results_671_hike
minwageDP_2.gau	##.2	1	64	34	Results_##2_hike	C:/research/minwage/results/Results_672_hike
minwageDP_3.gau	##.3	1	128	50	Results_##3_hike	C:/research/minwage/results/Results_673_hike
minwageDP_4.gau	##.1	0	[no hike]		Results_##1_nohike	C:/research/minwage/results/Results_671_nohike

Step 3: Simulate the model

Simulations are run by *minwageSimmerGrapher.gau*. Profiles are output to subfolders [graphs & sims] of the folders created in step 2.

Step 4: Estimate the effects

Table B on the next page clarifies the steps.

Table B	**Program**	**Run Time**	**Description**	**Actions**
get Optimal Decision Rules	Initializations.gau (located in C:\research\minwage\)	(never run, just updated)	Initializes all the parameters.	1. This step can be skipped if only replicating paper runs. To create a set of runs with new parameters: Copy the last elseif section in initializations.gau; change desired parameters; change *settingSwitch* to a ##.1 that a hasn't been used yet & *hikeStart* to 16; copy your new section and paste 2 times, changing *hikeStart* to 64 [128] & *settingSwitch* to ##.2 [##.3]. In each section, also initialize *initialPI* & *wageIncr* according to "targeting the income hike" above. You'll have 3 new sections total. (You do NOT need a separate section for nohike (*hikeSwitch* not in this program)).
	minwageDP_1.gau, minwageDP_2.gau, minwageDP_3.gau, minwageDP_4.gau (located in C:\research\minwage\)	2-6 hours each x4 (up to 36 hours, depending on grid size) (can usually run 3 DP codes at a time overnight).	Multiple files, identical but for the folder that results are output to, as well as *settingsSwitch* & *hikeSwitch*. Intended for running several files at once	2. Run 4 files, with: SettingsSwitch=##.1 hikeSwitch=1 SettingsSwitch=##.2 hikeSwitch=1 SettingsSwitch=##.3 hikeSwitch=1 SettingsSwitch=##.1 hikeSwitch=0 When hikeSwitch=1, hike kicks in at the age specified in the corresponding *settingsSwitch* block in *initializations.gau*. All 4 runs should be completed before doing step 4 below.
	Results from running the files above will be stored in the directories C:\research\minwage\results\newResults_tmp_1, C:\research\minwage\results\newResults_tmp_2, C:\research\minwage\results\newResults_tmp_3, C:\research\minwage\results\newResults_tmp_4		Directories that collect output from the minwageDP_1, minwageDP_2, minwageDP_3, minwageDP_4 programs respectively	3. For each "tmp" directory, create a new directory in C:/research/minwage/results***, named according to Table A in this file. (e.g. Results_671_nohike). 4. Copy output into your newly created permanent directories
Simulate	minwageSimmerGrapher.gau (located in C:\research\minwage\)	15min x 4 times	Does the simulation & creates graphs	5. Run 4 times, as follows: Set *basepath* =Results_##1_nohike for all 4 runs (where ##=67 for baseline case), and Set *innovationpath*= each of the following: {Results_##1_nohike, Results_##1_hike, Results_##2_hike, Results_##3_hike}.
	printMatrix.gau (located in C:\research\minwage\)	5min	Makes consM.txt, which is a matrix of simulated peoples' nondurables consum, income, & durables consum	Run Once. 6. [If not running baseline] Change the version number to "##1" before running

Estimate	SS_all_regs.do (located in C:\research\minwage\)	1min	1) Calls "convert_consM.do" 4 times for each run to convert consM.txt to .dta so you can run the regressions 2) Calls "sims_reg_fe.do" 3 times to run the regressions for each of the 3 ages separately 3) Calls "pooled_regs_fe.do" once, to run the pooled regression containing all ages	7. [If not running baseline] Change the version number (local macro v) in the file to ##1. 8. Run once in Stata 9. Copy output to "Version Table.xlsx".
	printMedians.gau (located in C:\research\minwage\)	1min	Finds quartiles for buffer, income, durables stock, assets, & cash on hand	Run Once. 10. [If not running baseline] change the version number to "##1" 11. Run & Copy output to "Version Table.xlsx".

*** Subsequent programs assume that these permanent directories are created inside the C:/research/minwage/results, however if you need to store them elsewhere, expect to change multiple pathnames in all the files listed below the blue bar in this table.

Note: Programs were run on a computer with the following specs:

Processor: Intel® Core™ i7-2600 CPU @ 3.40 GHz (8 cores)
Installed memory (RAM): 16.0 GB RAM
System Type: 64-bit Operating System

If you run into trouble running the paper's specifications, try running with coarser grids. E.g. In *initializations.gau*, replace the existing grid code lines with the following:

```
Astates = setGrid(-180|-35|-10|10|30|150, 10|.75|.5|3|10);
Sstates = setGrid(.3|2|7|25|50|90|220, .3|.4|.4|1|6|10);
Cdec = setGrid(Cfloor|2|4|18|27, .1|.1|.1|1);
Idec = setGrid(0|2|4|7|14|20, .1|.1|.1|.8|1);
```

NonDP – Analytic Solution

The analytic solution to the model is described in Section III and Appendix C of the paper. The model is solved in Gauss using *MPC_MPI_calc_graph.gau*, found in the "C:\research\minwage\nonDP – Analytic Solution" directory. Simply run the file.

Replicating Paper Figures 5 – 9 and Table 7-8

Figures 5 and 7 use direct output from *pooled income and spending graphs.gau*, (located in C:\research\minwage\). Currently the file is set up to run the baseline case: the variable *version*="671" (i.e. settingsSwitch loops over 67.1 to 67.3). For other specifications, you must change *version* (e.g., change from *version*="671" to *version*="721" for the case where $\pi = 1.0$) Version 67 is our Baseline model.

> The file, *pooled income and spending graphs.gau*, finds the average difference in {income, nondurable/durable consumption, and assets) resulting from a minimum wage hike, across 5000 simulated people, at each time period from 10 qtrs before to 10 qtrs after a minimum wage hike, simulated at 3 different ages.

Figure 6 is direct output from C:\research\minwage\NonDP - Analytic Solution*MPC_MPI_calc_graph.gau.* Specifications used for the paper solution can be found inside the file.

Figure 8 re-uses the same results (total, durables and nondurables spending) that were found for figures 6 & 7 – Analytic and Baseline. In addition, Debt for the baseline is negative assets change, output in *pooled income and sending graphs.gau*. For the analytic solution, $Debt_t = -[Y_{t-1} + rA_{t-1} - (C_{t-1}+I_{t-1})]$. The quantity $(Y_{t-1} + rA_{t-1})$ for the baseline is also output from *pooled income and spending graphs.gau* (when *version*="671"), and can be used for the Analytic model.

Figure 9 reports the output of quantile regressions on the Baseline (v67) and Baseline with adjustment costs, beta=.91^1/4 (This is v66). (We use beta=.91^1/4 instead of .93^1/4 because the former leads to buffer quartiles that more closely resemble those of the baseline). The quantile regressions can be run by un-commenting the last section of *SS_all_regs.do*.

Table 7 reports the Baseline specifications, which were set in settingsSwitch=67.1 to 67.3 in *initializations.gau*

Table 8 reports the results from *SS_all_regs.do* and *printmedians.gau* on the following runs respectively:

Table C

Run #	Description
67	Baseline, with extra fine grids
72	π = 1.0, with extra fine grids
73	π = 1.0, β=0.95^1/4
74	σ2e = 0, β=0.95^1/4
75	σ2e = 0.002, σ2u = 0.0, β=0.95^1/4
76	Adjustment cost = 0.05
77	Adjustment cost = 0.05, β=0.91^1/4
80	β=1.01^1/4, σ2e = 0, no borrowing constraints
81	β=1.01^1/4, σ2e = 0, adjustment cost = 0.05, no borrowing constraints

See *initializations.gau* for full specifications.

Appendix 1:

This appendix contains explanations of the key GAUSS files in the C:/research/minwage directory. Functions and their arguments are in bold.

List of Files:

minwageDP_1.gau (minwageDP_2.gau, minwageDP_3.gau, minwageDP_4.gau)
initializations.gau
minwage_sim_Graph_f.gau
minwageSimmerGrapher.gau
minwage_functions.gau

Explanations:

minwageDP_1.gau: This is the decision rule solver.
minwageDP_1 first calls set_paramsAndGrids(), from "initializations.gau," which has various built in settings that can be specified by changing the arguments passed in to set_paramsAndGrids.
The next main point is the "dllcall". getDecisionRulesGAUSS is called; this is the only function GAUSS can see in the DLL. It is the interface for GAUSS. It calls getDecisionRules() in C++, which actually does the work.
Of all the arguments passed in, which include all the parameters and the grids, only VM, bestCM, and bestIM are changed, since they are the value function and the two decision arrays. They are filled in as linear vectors, meaning they start off as nx1 matrices in GAUSS, where n=numAstates*numSstates*numPIstates*(numTstates-1). They are reshaped to be 4 dimensional arrays so we can access the data conveniently.
Next, gauss saves all the data and variables (parameters, grid settings, etc) to "outpath" which is set at the top of minwageDP.
[It is useful to save everything; occasionally it is later needed.]

---

minwageDP_2: minwageDP_2 is a copy of minwageDP_1 that does the same thing; we use it to solve two decision rules simultaneously, because GAUSS will only run one at a time, and a second GAUSS process can't run a file that's already being run.

You can typically run three minwageDP gauss files at a time.

Initializations.gau:
Only contains the function set_paramsAndGrids(settingsSwitch, hikeSwitch).

set_paramsAndGrids(settingsSwitch, hikeSwitch):
Basically, this is the function where you “store” settings that you want to use repeatedly for the simulation. It is two if-blocks; one does all the settings except for the wagehike, and the other turns the wagehike on or off. The main if-block sets the length and size of the wagehike, if it is on; the second simply turns it on or off.
Arguments:
settingsSwitch is just any number, associated with a group of settings you have stored in a block of the main if-statement.
hikeSwitch is 1 if you want the hike on, 0 if off.

---

minwage_Sim_Graph_f.gau:
This contains only sim_graph().

sim_graph(loadpath, innovationpath, savepathsims, savetempdirgraphs, rndSwitch, numsims, innovationTime):
sim_graph() does simulations and graphs the profiles (ie the averages of the data over all the sims) for the state and decision variables (A,S,C,I,PI) and for S/C, the

ratio of stock to consumption (which should theoretically stay constant),
and now for the buffer (a-afloor), and any other basic graphs that
should be produced as summary for every simulation.
sim_graph() is called by minwageSimmerGrapher.

The arguments:

loadpath: sim_graph reads in GAUSS (.fmt) matrix files which have the decision rules as well as all the parameters it needs to run the sim (i.e., depreciationrate (called delta), etc). loadpath is the directory which it opens and loads data from to start. The directory specified by the path should already exist(GAUSS won't create it).

innovationpath: sim_graph has the possibility of having an "information innovation" i.e. having the decision rules change mid-run. The first set of parameters are loaded from the "loadpath" directory (above). The second are from innovationpath. If they are the same path, then no innovation occurs, since the same files will be loaded the second time.

savepathsims: savepathsims is where the sim output is saved. The directory specified by the path should already exist (GAUSS won't create it). Everything having to do with the simulation should be saved here. That includes the profiles, the simulation settings (initasset levels, innovationtime, etc), as well as the simulation log file (which includes the numbers that generate the graphs).

savetempdirgraphs: This is used as part of a path for saving the graphs. $+ means "string concatenation."

rndSwitch: used to fix the shocks the sims receive. if rndSwitch is passed a 0 then there is a fixed seed that the random number generator will start at, meaning all the random numbers drawn will be identical each time that a 0 is passed in.

numsims: Used to vary the number of "sims" (agents) that we see.

innovationTime: time Value (not index) at which the information innovation occurs (i.e. when find out about the wage hike).

---

minwageSimmerGrapher.gau:
This is the file that calls sim_graph(); it is where you choose the basic simulation parameters you want to run on a given (already produced) set of decision rules, to produce the standard Profiles (durables investment, income, etc.).

---

minwage_functions.gau:
General library file. The functions are listed below but for more explanation see the comments in the code; if you want to know one of them you probably need all of them.

output_paramsAndGrids(settingsSwitch, hikeSwitch, Cdec, Idec, Astates, Sstates, PIstates, TSvals,transitionM, TSprobabilities, wageHike, hikeStart, R, fixedDurableCost, delta, downpaymentR, PIgrowthR, beta, rho, theta, CadjustmentLow, CadjustmentHigh, IadjustmentLow, IadjustmentHigh, numTstates, AfloorFnVersion):
Output_paramsAndGrids prints everything that's passed into it and prints a few useful pieces of information (ie an approximation of the number of operations that will be needed to solve the DP, so a rough estimate of the length of time it will take).

```
getFDC(Sval, FDC_multiplier):

getIncome(theTime, assets, stock, permIncome, PIstates, R, PIgrowthR, wageHike,
hikeStart, PIgrowthChangeTime);

getAfloor(stockState, downpaymentR, theTime, periods, fnVersion);

matchArray(baseArray, newArray)

binaryLocate( Xarray,  x)

tauch(stdi,rho,m,utail,ltail)

and a few others.
----------------------------------
```

Appendix 2: Generating *scf.out from* SCF source data

We use SCF data to set the initial distribution of the state variables, to be read in in *minwage_Sim_Graph_f.gau* (search for *scf.out*). The files in the SCF directory and CalculatingInitDistForSim directory together accomplish this task.

1. Download raw SCF data for the newest release and for all earlier years (there are changes/updates all the time). Specifically:
    a. Download the "COPY/EXPORT version" of the data from the Federal Reserve's Survey of Consumer Finances web pages.
    b. Modify & run the *TRANSPORT.sas* file inside the SCF folder to convert the data to .sas7bdat form.
    c. Place the resulting datasets in the folder named "raw"
2. Download the bulletin (*bulletin.macro.sas*) from the SCF website
3. Create directories nomYEAR and realYEAR, where "YEAR" is the date of your most recent data pull. (*bulletin.macro.sas* will spit out datasets into these directories.)
4. Change the libnames at the top of *bulletin.macro.sas* & add the x- variables we want (Search inside past custom bulletin files: "Additional variables for Minwage paper").
5. Save as *bulletin.macro.customYR.sas*, and run it once; it will build some datasets and summary statistics (populates the nomYEAR and realYEAR directories).
6. Then run each individual year file *scfregdataYEAR.sas*, which customizes the data for our use
    a. Also be sure to update libnames in all of these files
7. If adding another year, create a *scfregdataYEAR.sas* file for the latest year of data. Specifically:
    a. Copy the most recent *scfregdataYEAR.sas* file & rename
    b. Update libnames and pathnames (search for "yr" where yr is 07 or whatever year's file you're starting from)
    c. Update pceconvert (PCE Chain Price Index, or jcbm@USECON from haver) and federal minimum wage values
    d. Run
8. Update *readinstata.do* as necessary, and run it. It appends all the csv files from the different years into one dataset
9. *The output of all the above steps is* C:/research/minwage/SCF/*scfregdata8907.dta*
10. Finally, run C:/research/minwage/calculatingInitDistForSim/**scf.do**, which reads in *scfregdata8907.dta* and outputs C:/research/minwage/calculatingInitDistForSim/***scf.out.*** ***scf.out*** gets read in during the simulation (see *minwage_Sim_Graph_f.gau*). **scf.do** also generates statistics reported in appendix table A4 of the paper.

Appendix 3: Targeting the income hike caused by the minimum wage hike

To simulate patterns found in the micro data: We want to use an *initialPI* setting that makes quarterly income in the year before the hike an average $4500 across the 3 different ages that people are modeled to get a minimum wage hike. We also want to set *wageIncr* for each minimum wage hike age to be the growth rate that makes earnings jump $250 in the first period of the hike. We calibrate these values using the following programs:

1) *test_inits.gau*: for a set of parameters, change *initialPI*, trial and error, until the program returns an average income of about $4500.

Once you've found an *initialPI* that gives us $4500

2) *test_inits.gau*: for a set of parameters, start changing *wageIncr* for each of the different ages of the wage hike (n==2 thru ==4). Then run *test_inits.gau*.
3) Run *pooled_regs_fe.do*, which converts gauss output from *test_inits.gau* to .dta format and runs the fixed effect income regression. See if the coefficients on minwagefe for each age equal .250 (i.e. $250). If they don't, adjust *wageIncr* values for each age in *test_inits.gau* as appropriate and repeat steps 2 & 3.

The values of *initialPI* and *wageIncr* for each of the wage hike ages will then be put into *initializations.gau* for the appropriate runs.

The values of *initialPI* and *wageIncr* for each of the paper runs (see paper Table 8) are summarized in the file "Version Table.xls"

Appendix 4: Compiling minwageDP_DLL.dll using C++ for windows

Compile the C++ code to a single DLL file using your favorite compiler.
The underlying files are located in \research\minwage\C\:

Instructions are provided here for Microsoft Visual Studio 2010
(mostly based on MSDN's online instructions here: http://msdn.microsoft.com/en-US/library/ms235636%28v=VS.80%29.aspx]

1. Open Microsoft Visual Studio. From the **File** menu, select **New** and then **Project**
2. From the **Project types** pane, under **Visual C++**, select **Win32**.
3. From the **Templates** pane, select **Win32 Console Application**.
4. [Navigate to …\research\minwage.] Enter **minwageDP_DLL** in the **Name** field. Uncheck **Create directory for solution**
5. Press **OK** to start the Win32 application wizard. From the **Overview** page of the **Win32 Application Wizard** dialog, press **Next**.
6. From the **Application Settings** page of the **Win32 Application Wizard**, under **Application type**, select **DLL** if it is available or **Console application** if **DLL** is not available. Some versions of Visual Studio do not support creating a DLL project using wizards. You can change this later to make your project compile into a DLL.
7. From the **Application Settings** page of the **Win32 Application Wizard**, under **Additional options**, select **Empty project**.
8. Press **Finish** to create the project.
9. From the **Project** menu, select **Add Existing Item**
10. Browse to …\research\minwage\C and select all of the .cpp and .h files (press shift & click). Click **Add**
11. From the **File** menu, select **Save All**
12. To build the project into a DLL, from the **Project** menu, select **minwageDP_DLL Properties…**. From the left pane, under **Configuration Properties**, select **General**. From the

right pane, change the **Configuration Type** to **Dynamic Library (.dll)**. Press **OK** to save the changes.

13. From the **Build** menu, select **Batch Build**. In the window that appears, click **Select All**, then **Build**
14. The compiled file, **minwageDP_DLL.dll** appears in the **release** directory. Note that the minwageDP_#.gau files reference this file, so make sure to move the **minwageDP_DLL.dll** file to the C:/research/minwage/C, or update the pathname in the dll call (search "dlibrary") in minwageDP_#.gau

Appendix 5: Understanding and Running the Minimum Wage Simulations

The minimum wage simulation programs are a collection of tightly coupled programs spanning several languages. It is important to understand all of it because if you change one part of it, you're incredibly likely to break the rest of it. The program can be thought of in terms of the decision rules (mostly in C, setup in gauss), the simulations (all in gauss), and the output (partly in gauss, partly in Stata). This program is supporting the paper "The Consumption Response to Minimum Wage Hikes" by Aaronson, Agarwal, French and perhaps other papers in the future.

Setup and Directory Structure

All files should come in a zip file named "research.zip". You can place them in whatever directory you'd like. I highly recommend the following (a) if you chose "C:\Research\minwage\" then you won't have to change a million little details in the code [because it's mostly coupled to that directory structure] and (b) if you don't choose to place it in C:, and decide to change the code, don't place it in a directory with spaces in the name (ex: c:\documents and settings\etc) since gauss frequently spazzes out about this; yes you can get around it but it's just not worth fighting.

Decision Rules

The decision rules are set up and called from gauss but actually run in C. In order to compile and run this, you'll need a windows C compiler (I recommend MS Visual Studio which at the time of this writing, the fed would provide) and Gauss.

Setting up the decision rules

The decision rules are set up in **initializations.gau** , found in the base install directory. Here's the basic idea, each run set up is given a number in the setting switch, then you call that number later and you can be assured to run or rerun the exact same settings (plus it's a lot easier later to trace what you ran). There's a huge amount of dubiously useful in line documentation, but here's what you need to do if you're setting up a new run. Start by taking the base case for the paper and copy+pasting that section into a new part of the program (the section starts with elseif (**settingSwitch** == 67.1); and ends the line before the next elseif()) and giving it a new number (ex: 77.1, something that IS NOT YET IN USE). I picked 67.1 because at the time of this writing it was considered the base case. Why copy+paste the old one? Because (a) the vast majority of it won't change between runs and (b) it makes sure all of the variables you need are initialized.

Now that we have a working spec, let's take a look inside. The names try to follow the naming conventions in the paper however some of the names were reserved words in gauss and also the paper has changed over time. So the run down is **rho** is what the paper is calling γ or Coefficient of relative risk aversion. **beta** is the discount factor and also called β in the paper. **theta** is the utility weight on durables and also called θ in the paper. **PIgrowthR** is the income growth per period and called α_1 in the paper. **downpaymentR** is called π in the paper and is the downpayment rate. **delta** is the durables depreciation rate also called δ in the paper. **R** is almost what the paper calls r or the quarterly interest rate, in initializations R is the r+1. **fixedDurableCost** is the multiplier on today's stock to determine fixed cost of buying a durable, the paper calls this "adjustment cost". numPIStates is the number of states we're going to create for the PIstates matrix down below. **transitionM**, **PIstates**, and **c** are all set below via tauch, they're just initialized here, for a full explanation of exactly how these work, Eric has a paper on the tauch function. **sse** is variance of AR(1) innovations called σ^2_E in the paper. **ssu** is the variance of transitory innovations called σ^2_u in the paper. **rho_paper** is what the paper calls ρ or the autocorrelation of income. Skipping down a bit we see **hikeStart** which is the time period at which the minimum wage workers in the simulation will get a minimum wage hike. PIgrowthChangeTime is the time period at which the workers cease getting large pay increases each period and level off. **hikeLength** is how long the minimum wage workers getting the wage hike see that extra hike. Past that, the rest of these variables don't really manifest in the paper directly but are here to set up the grids which are getting passed to the decision rules. There are a few functions in this file but most of them aren't used or aren't worth mentioning here, you can read through the in file documentation for them, but they come up once a year.

When it's time to run the initialization you've set up, you must run the file **minwageDP_1.gau** in gauss. You'll notice a few things, first there are many versions of this file, this is so that you can run multiple decision rules at the same time, the process can take 8 or more hours, so it may be to your advantage to leave 3 or more copies running over the weekend. Each copy that you run will need to be modified in several ways. First, you'll notice many file paths such as "C:\\Research\\minwage\\...." etc. These will all need to be changed to your path locations. There seems to have been an attempt to swap between file paths based on the user, but you'll notice just how incomplete that attempt was (look at the #include section for functions, for example). Second, **settingSwitch** is the variable that you must set to the number of the initialization you wish to run. Third, **hikeSwitch** is a true/false (1 = true, 0 = false) flag for if the decision rules will be calculated for a hike condition or not. That's about it for this file, the rest of it is dedicated to initializing even more variables and printing a bunch of the conditions to the screen for you to visually verify. The line that starts the call to the C code is dllcall getDecisionRulesGAUSS().

Once the call to the C code is complete, there's some minor housekeeping to be done. If you have all of your results saved to a default directory (ex:\\results\\newResults_tmp_1) you should now go to that directory and copy the results to a permanent directory so that the results are not overwritten. If you're changing that path each and every single time you run **minwageDP_1.gau**, you can skip this step.

The decision rule calculations

This section is a walking tour of the C code. If you open the project in MS Visual Studio, you'll notice a bunch of things. First off, most of this code is dead code, it never gets used. Second, this project is set to compile to a .dll or a library file for windows. Basically this means that the function will be compiled and gauss can use it but gauss itself doesn't need to recompile to use the function (don't worry about the specifics, but that's the basic idea). Finally, and this is very important, back in **minwageDP_1.gau** dlibrary was set to point to C:\research\minwage\C**minwageDP_DLL.dll**. This means that if you don't change that path, if you recompile this project, you'll have to copy your new .dll file to that directory in order to make this program run. Inside minwageDP, you can find getDecisionRulesGAUSS() which is the function gauss

is calling [here is where all the variables are getting passed in]. You can see it sets up the data structures and then calls getDecisionRules(). First note that in getDecisionRules() you're writing out to "C:\\Temp\\minwageDP.log" keep that in mind in case you need to change the path or delete the file or look at the log, etc (If you don't have a Temp directory in C:\\, create it). Second, the function basically goes period by period calling getIncome() and then calling getBestDecision() [which is over in minwageDP.h]. Both of these functions are fairly mechanical and just grind through their equations. All of this C code can be rather intimidating, but for the most part it's just a bit jumpy and disorganized but not too tough to work through. That's about it, these matrices will be returned to gauss which will then save them to your path.

Simulations

Simulations are run by running **minwageSimmerGrapher.gau** in gauss. If you open the file, you can see almost nothing is in it other than setting up the **basePath** to match the directory where your decision rules were saved with no hike, the **innovationPath** to match the directory where your decision rules were saved with a hike (or point to the same directory as basePath if you're running a no hike scenario). A few of the other paths may have to be updated as well the first time you run the program, but you should be used to this by now. Basically all this program does is call the function sim_graph() in the file **minwage_Sim_Graph_f.gau**.

The sim_graph() function does all the real heavy lifting in the simulations, let's take a quick walking tour through it. After the program reads in the saved matrices from the decision rules, the program then reads in the scf data that we generated in Stata (see the income2pi section of sim_graph). I've added a ton of in document comments to this section, I encourage you to read them if you're looking for exact details. Next the program mashes the matrices from the hike and no hike together at the specified **hikeStart**. The next section is rather mechanically iterating through observations and time periods computing income, checking where the simulated person falls on the decision rule grid, and setting his/her consumption & investment for the period. There's a ton of little checks to make sure someone doesn't slide off the edge of a grid, etc, but that's the basic gist of this section. Finally, the entire 2nd half of this program is just printing out graphs and numbers to the screen.

Output

The final section of the program is a disjointed set of programs that generate output in various formats for the paper. These are all fairly simple program, but there are enough of them that it's easy to miss one.

One key question is "what are the median/quantile values of buffer, assets, and cash on hand just before people learn of the hike?" This is answered in **printMedians.gau**. Basically the program just reads in the matrices, sorts them, and prints the median value to the screen. You'll have to adjust **version** to run the file. Adjust **loadPath** and **loadPathHike** according to where you're storing permanent versions of the Decisions/Simulations matrices output. I copy+paste these quantile values from the screen into a spreadsheet called "Version Table".

Finally, there are a number of regressions in Stata tracking differences in consumption and investment based on our hike. If you want to run these the first challenge is reading the data into Stata. Since at the time of writing this, our handy conversion tool for changing gauss files into Stata files was removed, there's a fast way to just print the matrices to a flat file and read that into Stata. To print out the matrices run **printMatrix.gau** after changing **version**, which changes **loadPath**. Next you'll need to open **SS_all_regs.do**. The first portion of the program is devoted to reading in the data. Stata sometimes treats the data as strings and sometimes as reals. Once the data is read in, the

regression can be run by specifying the two input files (the time period of the rate hike is inferred from the directory name). I copy+paste the regression output from the screen into a spreadsheet called “Version Table”.