CHANGES BY FILE

1. **main.r** 
   1. Controlled by function called “main” - calls initializer, create\_initial\_data, and simulation-loop
      1. initializer: Set up initial market, prices, and storage variables
      2. create\_initial\_data: Build linit(see input) sized matrix of initial market data
      3. simulation-loop: Begin modelling process with calls to EstParams
2. **functions-lm-v1.R**
   1. **Risk:**
      1. Added *MSFE* function to calculate mean square forecast error for a given agent
      2. Added function *hetero\_MSFE* to compute a risk aversion value by calculating the mean squared forecast error for [memory] forecasts back for all agents and averaging them
      3. New Input Variables:
         1. *risk\_constant*: The risk constant used to calculate market price after using MSFE calculations
         2. *risk\_type*:
            1. 0 = no endogenous risk (If risk type is 0, MSFE will not be calculated so simulation should run faster)
            2. 1 = homogenous risk (MSFE for RepAgent)
            3. 2 = heterogenous risk (MSFE averages across all agents)
      4. Switches are controlled in *Market\_Price* function and input parameters are in the file *input-lm-v1.2.txt*
   2. **Estimation models:**

The vairous methods for selecting a model to predict the market

* + 1. lm: set run\_type to 0
       1. Run simple linear regression model
    2. glmnet with lambda=0: set run\_type to 1
       1. Also a similar linear regression model
       2. In theory equivalent to lm, though they are not exactly the same
       3. included for comparision purposes
    3. glmnet with lambda=.1: set run\_type to 2
       1. glmnet a specifically defiened lambda value
       2. higher lambda will make the model favor including fewer terms
       3. Included for comparision. Not very useful itself, as giving the function a specific lambda is akin to telling it what the correct answer is
    4. glmnet with cross validation: set run\_type to 3
       1. glmnet where the function decides on a lamba
    5. Hybrid forward/backward selection: set run\_type to 4
       1. Uses a hyrid of forward/backward model selection
       2. selects the model with lowest BIC value
  1. **Model Updates:**
     1. Lags and powers are now inputs
        1. UpdateMatrix -> matrix size is now dependent on lag and power inputs from the user. This allows for more complicated conditions to be used.
        2. Allows us to generalize code throughout functions and main file, to accommodate any sized matrix.

1. **plots-lm-v1.2.R**
   1. Run it line by line
2. **input-lm-v1.2.txt**

Input file for the various parameters. Values are retrieved by GetMacros() in functions.

Input values are as follows:

* 1. popsize: the number of agents in the populations
  2. rounds: Number of rounds in the simulation
  3. linit: The number of rounds used for initializing data
  4. memory: number of rounds back of data the agents remember. Must be smaller than linit
  5. pupdate: percentage of the population who update their model each round
  6. lags: how many lags to include in simulation function
  7. powers: highest power to include in simulation function
  8. numBubbles: keep this at 0. Used for MonteCarlo to indicate a bubble after each run.
  9. startPrice: the initial price used in the simulation. Should equal

(dividend / interest) + dividend

* 1. bubbleThresholdHigh: High boundary set for bubble indicator
  2. bubbleThresholdLow: Low boundary set for bubble indicator
  3. interest: used to initialize price in arbitrage condition
  4. dividend: average dividend for the stock
  5. shockRangeDiv: How much higher or lower the dividend is than the average dividend value
  6. risk\_constant: value in range -1 to 1, for risk factor
  7. pshock: change there will be a shock to dividend

1. **test\_program.r**

This tests all of the current models on randomly generated data. These models are

* 1. lm
  2. glmnet with lambda = 0
  3. glmnet with lambda = 0.01
  4. glmnet with lambda = 1
  5. glmnet with cross validation

The top of the file includes several variables that can change how the test is performed

* average - the average value for the random data
* div - the maximum deviation from the average
* lags - the maximum number of lags in the model
* powers - the maximum number of powers in the model
* randSeed - the random seed, for reproducible results
* memory - the number of rounds used for the model
* start\_time - the number of rounds until the agent starts
* end\_time - the total number of rounds
* model\_names - the names of the models

After these models are calculated, test\_plots.r is called and saves various plots in .pdf format.

1. **monte\_carlo\_test.r**
   1. Before Running MonteCarlo:
      1. Comment out set.seed at the top of main file
      2. Comment out call to main at the bottom of main file
      3. Comment out “runType <<- subset(input, Varnames == "runType")[[2]]” in GetMacros in functions file
      4. Comment out “randSeed <<- subset(input, Varnames == "randSeed")[[2]]” in GetMacros in function file
      5. Comment out print statements in main
   2. Run the simulation with the MonteCarlo package
      1. Currently must be sequential (1 cpu) due to error
      2. Change randSeed list and run\_type list for desired output
      3. Change nreps in the call to MonteCarlo for desired repitions
      4. Each run is set to 0 (no bubble) or 1 (bubble), and total bubbles is printed.
2. **montecarlo.r**
   1. This file is relatively useless, but we decided to include it anyway
   2. Was the start of writing our own montecarlo package, but we decided it would be a better use of our time to use the MonteCarlo package
   3. Having issues with parallelization packages on HPC
3. **montetest.r**
   1. Copied from the MonteCarlo package documentation
   2. Uses a test function to run the MonteCarlo package
   3. Good for trying to expand runs to multiple cores because you will not have to wait for simulation runs to finish
   4. Can run on its own
4. **Dependencies:**
   1. glmnet, MonteCarlo, snow, snowfall, rlecuyer, dplyr, zoo, rlist, xts, ggplot2
   2. To install a package, in the R repl use the command install.packages(“package\_name”). For example, to install glmnet you would use the command install.packages(“glmnet”)
   3. once a package is installed, put the line library(package\_name)