Prepared for: Professor Fontana

Collegio Carlo Alberto
UNIVERSITÀ DEGLI STUDI DI TORINO

Turtles vs. the Euro

Using price data and a genetic algorithm to find profitable trading strategies

Dan Herweg

February 2019

MADAS
MASTER in DATA SCIENCE for COMPLEX ECONOMIC SYSTEMS

This is not the first time that turtles have traded the markets

In the 80s, a famous futures trader set out to prove he could teach anyone to trade

He referred to his students as turtles, the same word used for the agents in netlogo

<u>This project has nothing to do with that</u> (and was written in python), but it is the source of the pictures I found for this presentation, **for which <u>I apologize in advance</u>**

Pictures: https://canadafutures.com/, https://learnpriceaction.com/turtle-trading-strategy-richard-dennis-william-eckhardt/

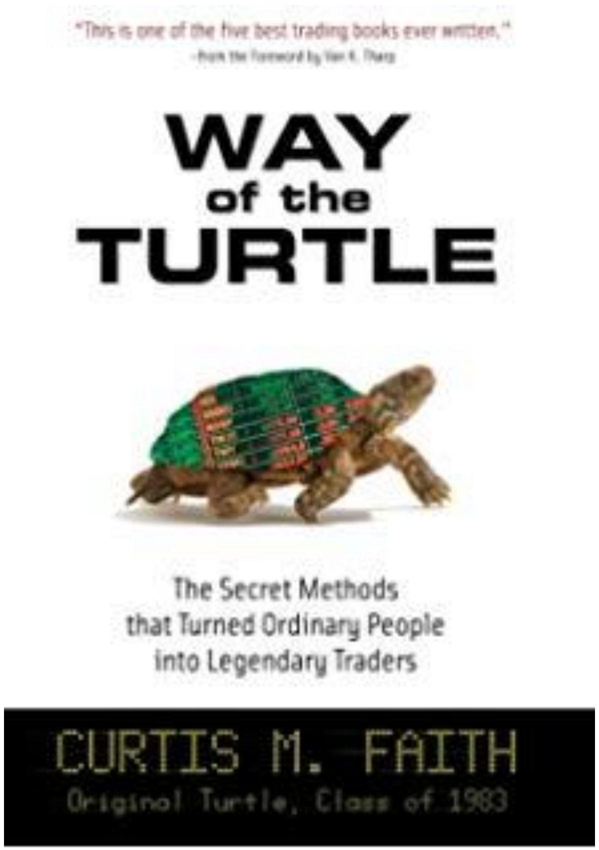
Contents

I. Background

Project Summary

Genetic Algorithms

Data



Picture: https://www.goodreads.com/book/show/503534.Way_of_the_Turtle

What do turtles want?

Project Summary

This project uses a population of agents
in a genetic algorithm

who use price information
from macroeconomically significant assets

to develop a strategy
to predict the best time to buy the Euro
and sell the next day

in order to make money
through speculation
in a simulation

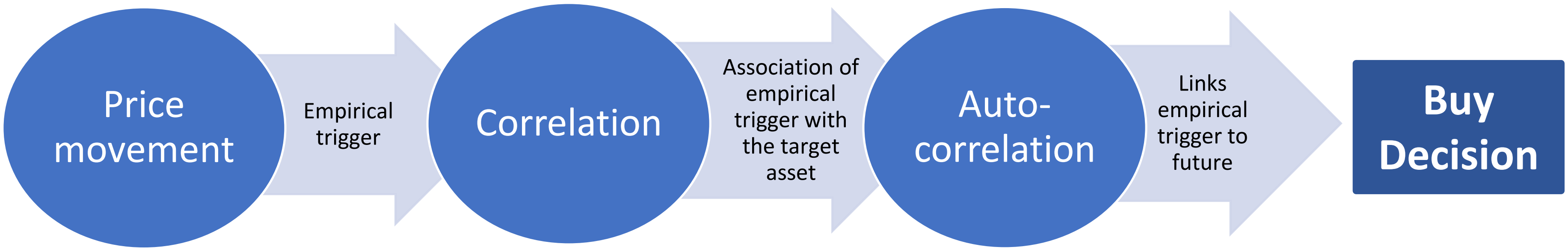
and maybe one day
for real

Picture: https://www.123rf.com/photo_20237672_turtle-on-money.html

What do I expect?

Project Summary

My best guess is that we will see strategies emerge that resemble causal chains that look like this:



For example:

Buy when:
The Euro is up for the past day
The Euro is positively autocorrelated

Buy when:
The Pound is up for the past day
The Pound is positively correlated with the Euro
The Pound is positively autocorrelated

Genetic algorithms can be used to solve big, difficult problems using agents

Genetic Algorithms

- GAs are especially useful for finding local optima in problems characterized as:
  - Not analytically tractable
  - Having too large a variable space for brute force search

Is finding a trading strategy that big of a problem?

Our problem is too big to solve via brute force, so genetic algorithms may be an appropriate approach

Genetic Algorithms

- This model's trading strategies have 60 variables with 3 possible states each, so theoretically this model contemplates 3^{60} trading strategies, or about

42,319,158,275,216,200,000,000,000,000

- Clearly it is infeasible to exhaustively explore an amount of strategies that is
  - 50x the diameter of the visible universe in meters
  - 7,000x the mass of the Earth in kilograms

The structure of a genetic algorithm mimics genes in a sexually reproducing population over generations

Genetic Algorithms

1. Create a population with diverse genotypes

2. Evaluate the population according to a fitness function

3. Repeat the following until a stopping criteria is reached:
   1. Crossover: recombine genes, favoring more successful agents
   2. Mutation: simulate copying errors by flipping genes randomly
   3. Fitness function: Evaluate the new population

Exchange traded funds that track macroeconomically significant were used to characterize market conditions

Data

Data Accessed

Close prices from January 2010 – February 2019 for:

Ticker	Asset tracked
FXE	Euro
SPY	S&P 500 stock index
GLD	Gold
BND	Bonds
USO	Oil
FXB	British Pound
FXY	Japanese Yen
CYB	Chinese Yuan

Features Extracted

- Binary arrays (positive = 1, negative[1] = 0):
  - **Asset price change** for the past 1, 5 and 20 trading days[2]
  - **Correlation of asset returns** for previous 20 trading days
  - **Autocorrelation of asset returns** at lag of 1 day
- One trading day forward returns for the Euro

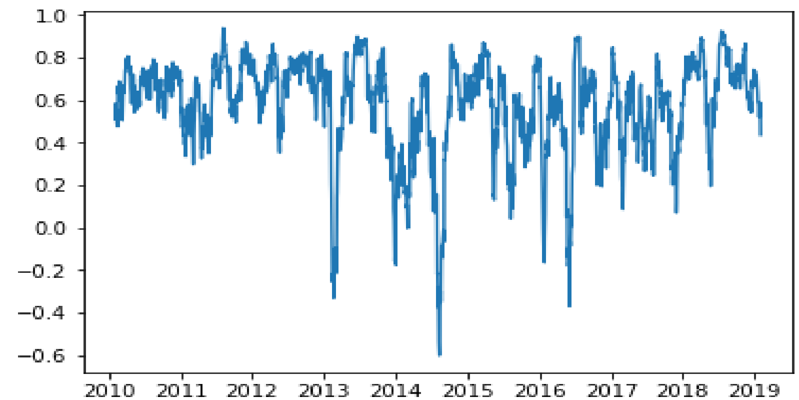
1. Down includes unchanged for all binary variables
2. 1 trading day is daily , 5 trading days is weekly and 20 trading days is monthly for this analysis

Some examples of data

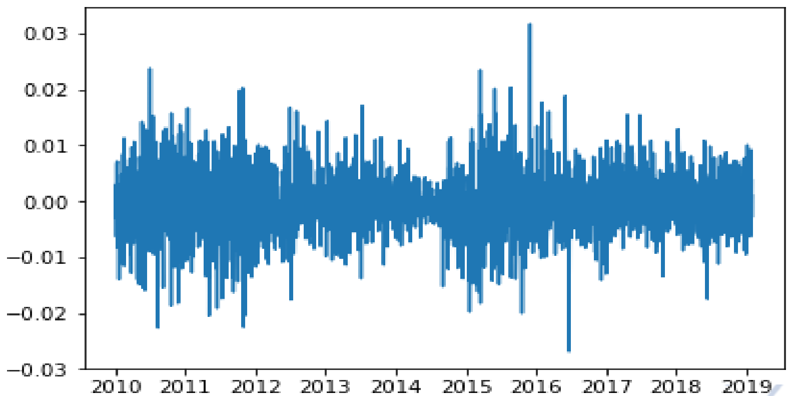
Data

Autocorrelation of FXE

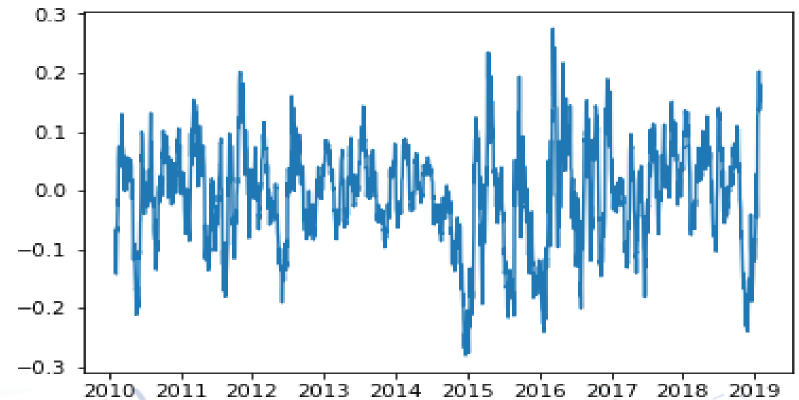
Correlation of FXE and FXB



Daily Returns of FXE



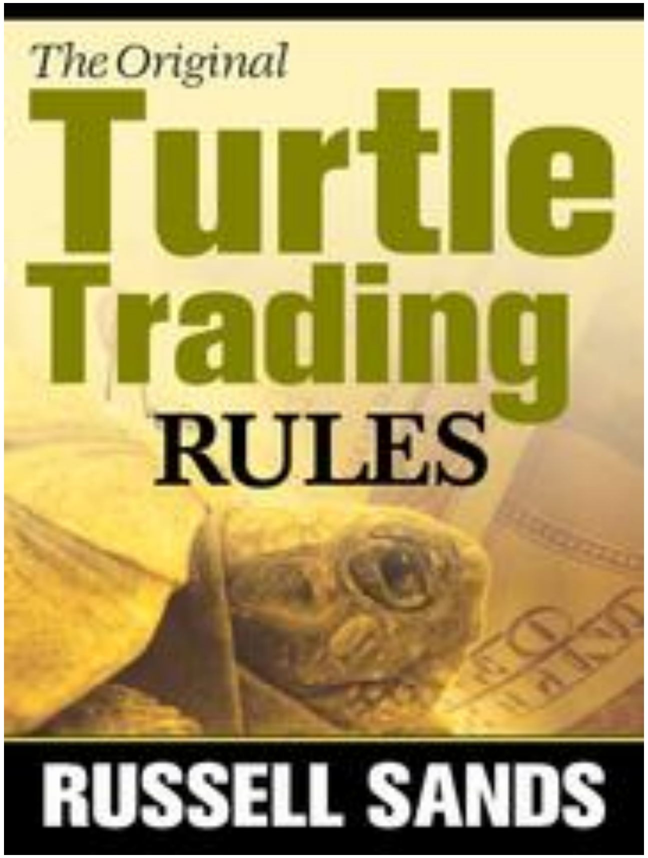
Monthly Returns of USO



II. Model Walkthrough

How Agents Decide to Trade

How the Algorithm Runs



Markets are represented by a binary array

How Agents Decide to Trade

Representation of the market conditions on a theoretical day:

FXE was up today

GLD was down[1] this week

SPY was down this month

FXE and USO are positively correlated over the past month

USO and FXB are negatively correlated over the past month

BND price movements have a negative autocorrelation at lag 1

10100100 01010100 10001010 1011110 101010 01010 0101 010 01 0 10101010

Daily Up/Down

Weekly[2] Up/Down

Monthly[3] Up/Down

Asset Correlations

Autocorrelations Lag 1

1. Down includes unchanged for this and any other binary variable
2. 5 trading days is considered weekly for this analysis
3. 20 trading days is considered monthly for this analysis

Trading strategies are represented as a ternary array of conditions under which the agent will buy

How Agents Decide to Trade

A trader's strategy is represented as requirements for buying:

FXE must be up today

GLD must be down this week

SPY must be down this month

FXE and USO must be positively correlated over the past month

USO and FXB must be negatively correlated over the past month

BND price movements must have a negative autocorrelation at lag 1

10000000 00-100000 0-1000000 0000100 000000 00000 0000 -100 00 0 000-10000

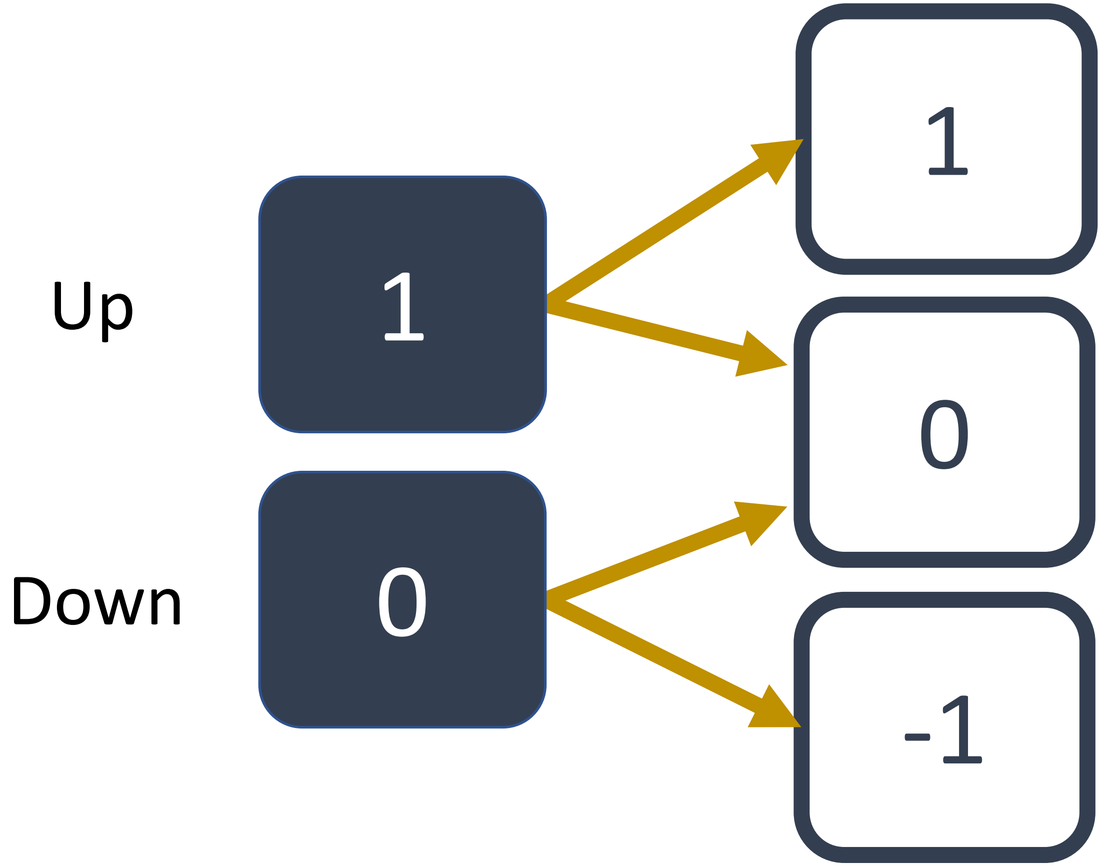
Zeros mean the agent does not consider the variable in their buying decision

If all conditions are satisfied, the trader will buy the Euro

Mapping market variable states to agent strategies

How Agents Decide to Trade

Market Variable | **Agent Strategies**



Buy requires variable state 1

Not a decision variable

Buy requires variable state 0

Mapping market states to agent actions

How Agents Decide to Trade

	Var1	Var2	Var3	
Market State	1	0	1	Variables 1 and 3 are up, variable 2 is down
Agent 1	1	-1	1	Agent Buys
Agent 2	-1	0	1	Agent does nothing
Agent 3	0	-1	0	Agent Buys

The original examples would have resulted in a buy

How Agents Decide to Trade

The market on a theoretical day:

FXE was up today

GLD was down this week

SPY was down this month

FXE and USO are positively correlated over the past month

USO and FXB are negatively correlated over the past month

BND price movements have a negative autocorrelation at lag 1

10100100 01010100 10001010 1011110 101010 01010 0101 010 01 0 10101010

A trader's strategy:

10000000 00-100000 0-1000000 0000100 000000 00000 0000 -100 00 0 000-10000

1,1 OK

0,-1 OK

0,-1 OK

1,1 OK

Zeros do not influence decision

0,-1 OK

0,-1 OK

This trader will buy the Euro today at the close and sell tomorrow at close

The model is set up to train a genetic algorithm on trading days and then test the resulting strategies on later data

How the Program Runs

- **train()** – this function initializes the program by creating agents and variables that will be stored as the program runs, then subjects the population to a genetic algorithm loop until a stopping criteria is met

- **test()** – this function extracts the trading strategies from the training data that will be tested on a period following the training period

The train() function

How the Program Runs

- **train()**
  - **initialize()**
    - create_class()
    - init_strat()
    - create_inst()
    - reset_stats()
  - **loop()**
    - calc_fitness()
    - get_stats()
    - get_wins()
    - crossover()
    - mutation()
    - new_pop()

The training function initializes the program and iterates on a loop function to run the genetic algorithm

How the Program Runs

- **train()**
  - **initialize()** – this function initializes the program by creating agents and variables that will be stored as the program runs
  - **loop()** – this function subjects the population to a genetic algorithm

The initialize function uses four functions to prepare to run the genetic algorithm

How the Program Runs

- **train()**
  - initialize()
    - **create_class()** – creates the class Trader which can save each trader's strategy, trade results and trade dates
    - **init_strat()** – creates a 60-digit string of -1s, 0s and 1s for each trader to represent the conditions that will induce a buy

      The probability of 0s (indifference) is very high because there are ~2000 examples of a possible 2^{60} market conditions, and very demanding traders may never trade
    - create_inst()
    - reset_stats()

The initialize function uses four functions to prepare to run the genetic algorithm

How the Program Runs

- **train()**
    - initialize()
        - create_class()
        - init_strat()

        - **create_inst()** – this creates instances of traders and assigns them strategies

        - **reset_stats()** – this sets (or resets on subsequent initializations) all of the variables that are to be captured during runtime

Train() iterates the loop function, which runs the steps of the genetic algorithm

How the Program Runs

- **train()**
  - initialize()
  - loop()
    - **calc_fitness()** – matches trader strategy with market conditions and assigns trading results when they match
    - **get_stats()** - fills in statistics of interest, beginning with the calculation of the total return of each trader's trades less a penalty for each trade to represent the bid-ask spread
    - **get_wins()** – records additional information about winning strategies for each generation in the algorithm and the top 50 strategies once convergence is reached
    - crossover()
    - mutation()
    - new_pop()

Train() iterates the loop function, which runs the steps of the genetic algorithm

How the Program Runs

- **train()**
  - initialize()
  - loop()
    - calc_fitness()
    - get_stats()
    - get_wins()

    - **crossover()** – weights the trader's strategies by their fitness, using

      $$w = (1 + \text{total return } \%)^{\text{exponent}}$$

      and performs a random weighted selection of two parent strategies

      These strategies are combined in a single point crossover at the midpoint of the strategy to create a child strategy

      This is repeated until strategies are created for the whole population

Train() iterates the loop function, which runs the steps of the genetic algorithm

How the Program Runs

- **train()**
  - initialize()
  - loop()
    - calc_fitness()
    - get_stats()
    - get_wins()
    - crossover()

    - **mutation()** – a proportion of the decision variables is subject to mutation, being replaced by -1, 0 or 1, using the distribution from the strategy creation step

    - **new_pop()** – the newly created strategies are assigned to traders

The test() function

How the Program Runs

- **test()**
  - create_testers()
  - add_BestGAs()
  - test_fitness()

The test function tries out the strategies out of sample

How the Program Runs

- **test()**
  - **create_testers()** – creates the class Tester which can save each tester strategy's name, strategy, trade results, trade dates, training return and testing return
  - **add_BestGAs()** – creates four instances of the tester class
    - The converged upon strategy
    - The best strategy from training
    - Two strategies generalized from the top 50 strategies when the algorithm stopped
      - These were invented to test if GA results would be 'overfit' and looser results more robust
  - **test_fitness()** – tests the returns of each tester over the testing period

The train() and test() functions

How the Program Runs

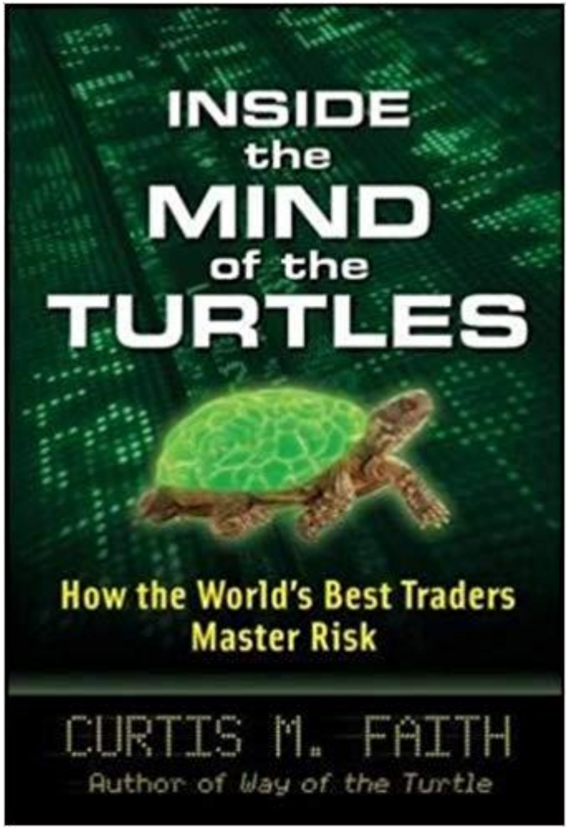
- **train()**
  - **initialize()**
    - create_class()
    - init_strat()
    - create_inst()
    - reset_stats()
  - **loop()**
    - calc_fitness()
    - get_stats()
    - get_wins()
    - crossover()
    - mutation()
    - new_pop()
- **test()**
  - create_testers()
  - add_BestGAs()
  - test_fitness()

III. Analysis

Finding Parameters

Train/Test Example

Walk Forward Tests



Picture: https://www.bookdepository.com/Inside-Mind-Turtles-How-Worlds-Best-Traders-Master-Risk-Curtis-Faith/9780071602433

Test runs were performed to find good parameters with which to run the model

Finding Parameters

- The model has several parameters with no ideal value knowable a priori

- These parameters and the values that were tested are:
  - Number of agents (100, 500)
  - Chance that a market variable is irrelevant to trader strategy (80%, 95%)
  - Mutation rate (3%, 20%)
  - Exponent of crossover weighting function (7, 20)

- Appropriate values could be found empirically based on which make for the best model

The goal was to balance exploration and execution speed (exploitation)

Finding Parameters

- Ideally this could be run over more variables, many times. However, runs grow exponentially with more variables, so only 2^4 models were run

- If these models converged to the same optima, it might make more sense to minimize execution time

- However, they did not, so tradeoffs and judgement calls were necessary

Each run continued until 7 generations converged to the same result or timed out at 50 generations

Finding Parameters

Run Results

Run #	Traders/Irrelevant/ Mutation/Exponent	Converged Result	Best Result (if Better)	Generations to Converge	Time (m:ss)
1	100/0.8/3/7	2.13%		5	0:10
2	100/0.8/3/20	1.46%		2	0:07
3	100/0.8/20/7	2.38%	3.59%	NA	0:42
4	100/0.8/20/20	3.48%	4.97%	NA	0:42
5	100/0.95/3/7	2.49%		14	0:17
6	100/0.95/3/20	1.95%		1	0:07
7	100/0.95/20/7	**4.97%**		34	0:35
8	100/0.95/20/20	4.42%		NA	0:42
9	500/0.8/3/7	3.64%	3.88%	NA	3:47
10	500/0.8/3/20	4.42%	4.97%	38	3:37
11	500/0.8/20/7	3.01%	4.01%	NA	3:45
12	500/0.8/20/20	3.43%	4.97%	NA	3:49
13	500/0.95/3/7	3.16%		8	1:07
14	500/0.95/3/20	3.66%		11	1:40
15	500/0.95/20/7	**4.97%**		35	4:56
16	500/0.95/20/20	**4.97%**		36	4:38

Runs with 100 agents achieved poor results and often converged quickly to them

Some runs evidently started in a sufficiently attractive local optimum and did not explore much

The longest runs were of tolerable duration and seemed to balance exploration and execution best

Three runs with 0.95 irrelevant genes hit the top maximum return

Five of the six runs that timed out after 50 generations with no convergence had high mutation rates

Selected parameters and justification

Finding Parameters

Number of agents: 500

- More agents appeared more likely to converge to a better result

Chance that a market variable is irrelevant: 90%

- The 95% value arrived at better results and seemed more likely to converge

  However, 95% also seemed likely to explore too little, so it seemed better to only nudge toward that value

Mutation rate: 10%

- The combination of high irrelevant genes and high mutation rates to promote exploration. However, the 20% mutation rate seemed too much, often resulting in the model timing out without convergence
  - It should be noted that the mutation rate is likely overstated compared to other models. The high amount of irrelevant genes and using the original distribution of -1/0/1 genes to replace a mutated gene yields an actual rate of randomly changed genes much lower than the model parameter

Exponent: 20

- The higher exponent to boost high fitness strategies tended to find higher return strategies

The first test runs the genetic algorithm for 60 trading days, then tests on the next 60

Train/Test Example

- This will simulate how a model like this might be applied: using last quarter's data and applying the resulting strategy in the next quarter

- Train/test periods are another variable that could be explored and tuned using a measure for test error

- However, here we proceed assuming that it is reasonable that:
  - 60 trading days are enough to detect winning strategies
  - The duration of the conditions that led to the strategies' profitability will persist and be profitable for the next 60 trading days

The first test yielded a strategy with positive return

Train/Test Example

- The strategy that emerged from the training period was to buy the Euro when:
  - Oil was down the previous day
  - The Euro and Yuan were positively correlated
  - The Yen and Bonds were positively correlated

- In the training period, these rules earned 5.0%

- Using these rules in the test period, our tester made 26 trades that yielded 1.6%

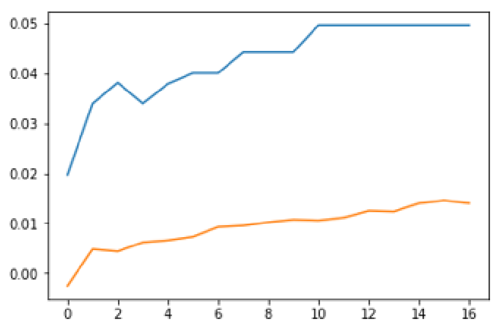
Training and testing model outputs

Train/Test Example

Training

```
train days: 20 - 79
Run:  500/0.9/10/20      Traders/Irrelevant/Mutation/Exponent
gen 1     max: 2.0 %     avg: -0.3 %     time: 4.1s
gen 2     max: 3.4 %     avg: 0.5 %      time: 4.04s
gen 3     max: 3.8 %     avg: 0.4 %      time: 4.04s
gen 4     max: 3.4 %     avg: 0.6 %      time: 4.05s
gen 5     max: 3.8 %     avg: 0.7 %      time: 4.02s
gen 6     max: 4.0 %     avg: 0.7 %      time: 4.07s
gen 7     max: 4.0 %     avg: 0.9 %      time: 4.11s
gen 8     max: 4.4 %     avg: 1.0 %      time: 4.03s
gen 9     max: 4.4 %     avg: 1.0 %      time: 4.04s
gen 10    max: 4.4 %     avg: 1.1 %      time: 4.08s
gen 11    max: 5.0 %     avg: 1.1 %      time: 4.08s
gen 12    max: 5.0 %     avg: 1.1 %      time: 4.06s
gen 13    max: 5.0 %     avg: 1.2 %      time: 4.05s
gen 14    max: 5.0 %     avg: 1.2 %      time: 4.08s
gen 15    max: 5.0 %     avg: 1.4 %      time: 4.09s
gen 16    max: 5.0 %     avg: 1.5 %      time: 4.08s
gen 17    max: 5.0 %     avg: 1.4 %      time: 4.25s
¯\_(ツ)_/¯  - stopping because 7 gens convergance
Total Time:  69.28s
Max achieved at gen:  11
Stopping criteria in gen: 17
```

The genetic algorithm ran for 17 generations to converge on a strategy

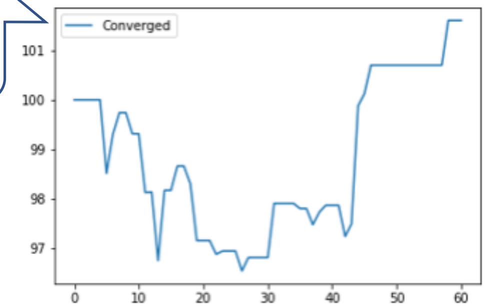


Max and **average** trading strategy over 17 generations

Testing

```
test days: 80 - 139
name          dec vars    train return    test return    trades
Converged            3           5.0 %          1.6 %        26
```

Equity curve for the test period, starting with $100



Training and testing returns

```
Converged          decision vars: 3          return 1.6
- day chg in USO
+ FXECYB corr
+ BNDFXY corr
```

Strategy from training period

After training and testing to positive returns, a validation method is needed to rule out dumb luck

Walk Forward Tests

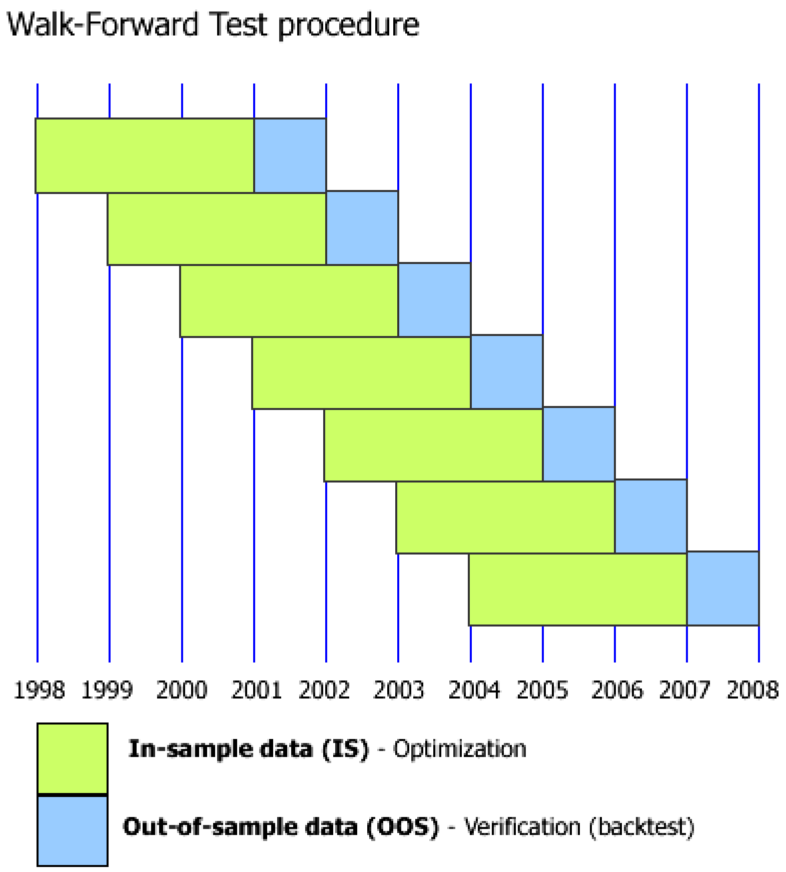
- There is a risk that the agents will discover a strategy that is overfit to the specific period it was trained on

- The test results were weak and inconsistent, and could also be a fluke

If the approach is robust, and changing market regimes exist and can be detected in the model, walk forward testing will yield strong returns and validate the approach

Walk forward testing over all of the data could validate this approach

Walk Forward Tests



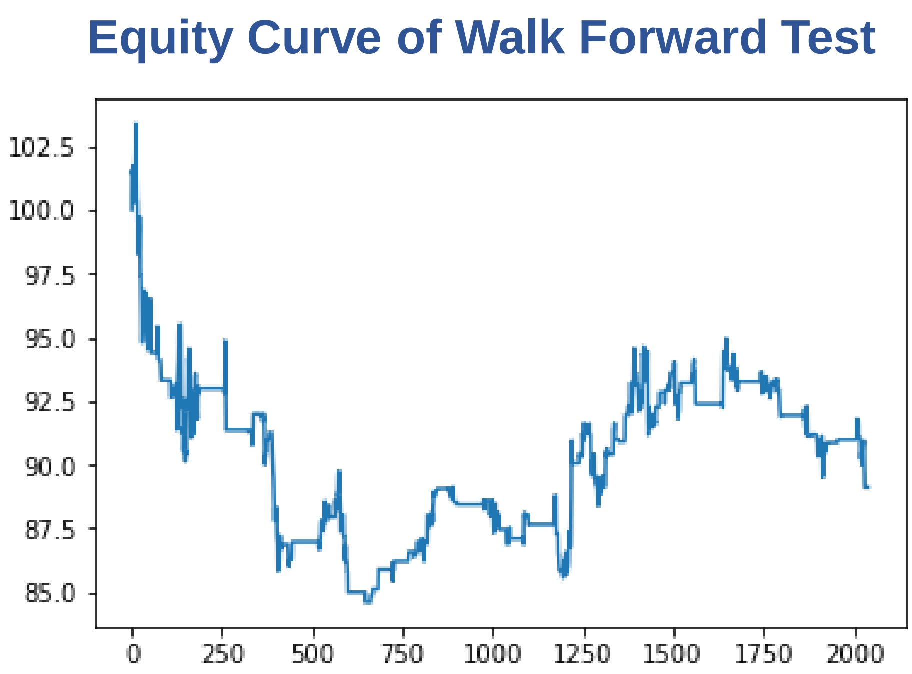
- The walk forward testing procedure mimics real-time training and implementation of a system:
  1. Train the algorithm on a sample period
  2. Test on the period immediately after
  3. Repeat over a large dataset of diverse market conditions

- The training will begin on the day following the initial test period and create a contiguous test set by moving the training and testing start dates forward 60 trading days each iteration

Picture: https://www.amibroker.com/guide/h_walkforward.html

The results of the walk forward test were not profitable

Walk Forward Tests



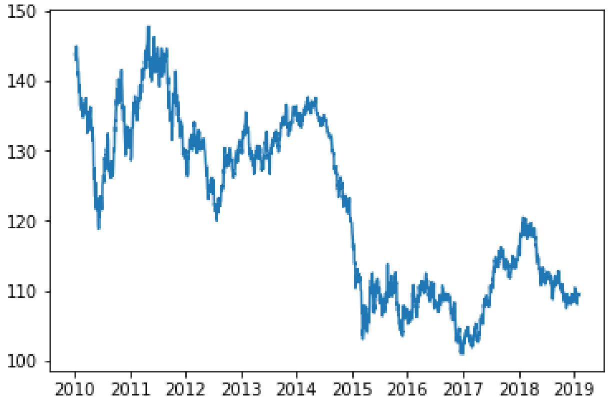
- The equity curve of all of the test periods strung together loses ~10% over 2,000+ trading days
- The strategy steadily dropped 15% over the first 700 days
- While this strategy is long-only and only suffered half of the loss of a buy-and-hold strategy, no ability to identify and exploit market conditions is seen

Now that the model is built, other hypotheses can be tested rapidly

The algorithm found profitable short Euro strategies in the declining market

Walk Forward Tests

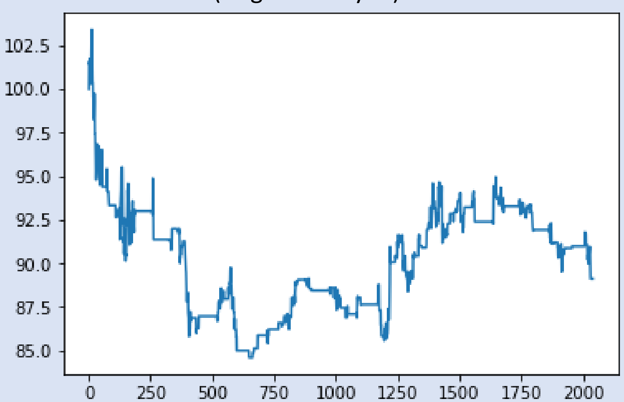
FXE



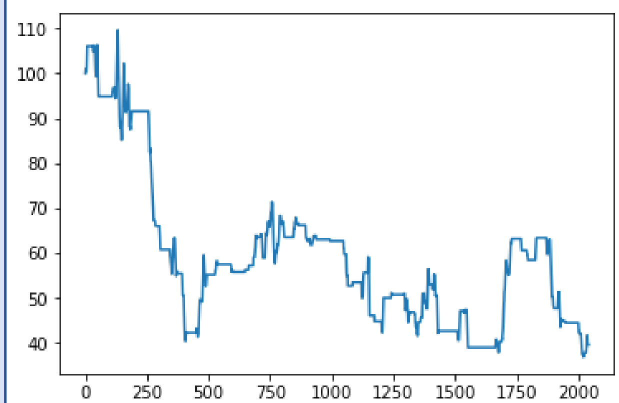
- The GA makes ~10% finding short strategies over the period of 2,000+ trading days
- The 5 day holding period strategy loses ~60%
- **These returns mirror the move in the Euro over the period, as would a random buying or selling strategy. It is not clear we do much better!**

FXE 1 Day Buy

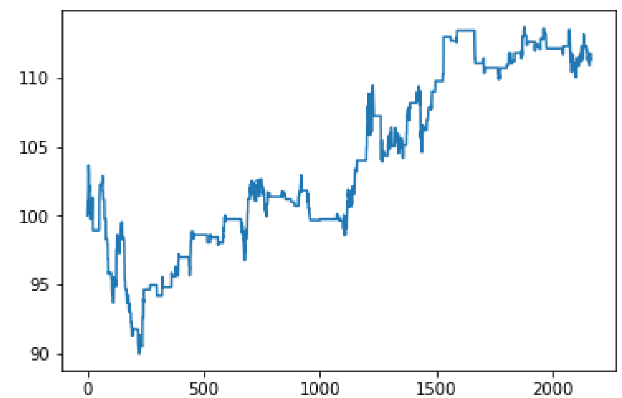
(original analysis)



FXE 1 Week Buy



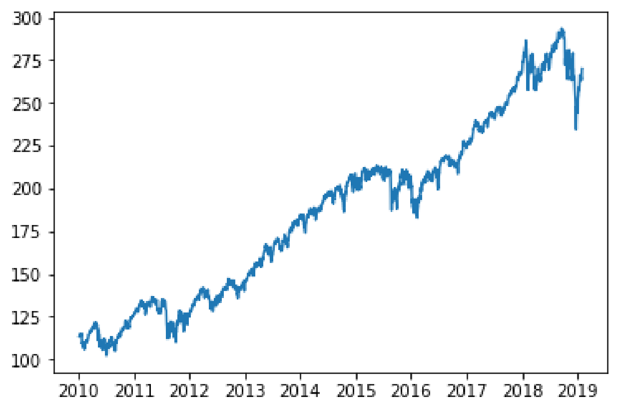
FXE 1 Day Short



Testing the algorithm to buy or sell stocks revealed the same correlation to asset returns generally

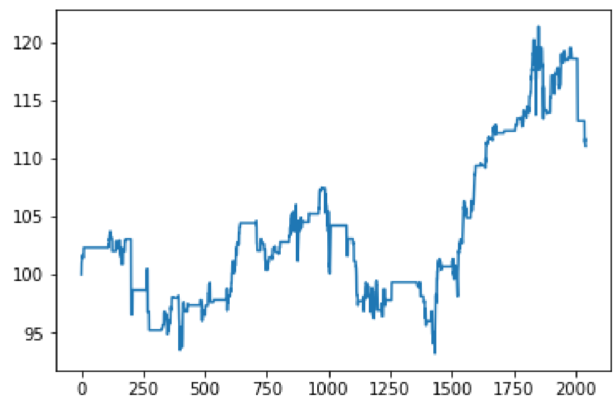
Walk Forward Tests

SPY

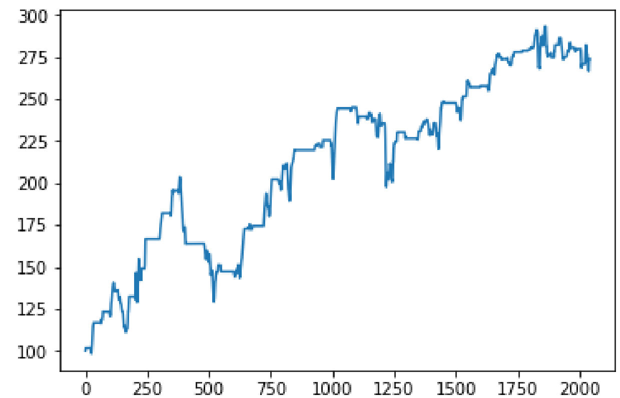


- Again strategies that reflect the overall move in the asset price make money, in this case buying
- The 5 day holding period strategy appears equivalent to a buy and hold strategy
- The shorting strategy manages to only lose 10% in a strong bull market

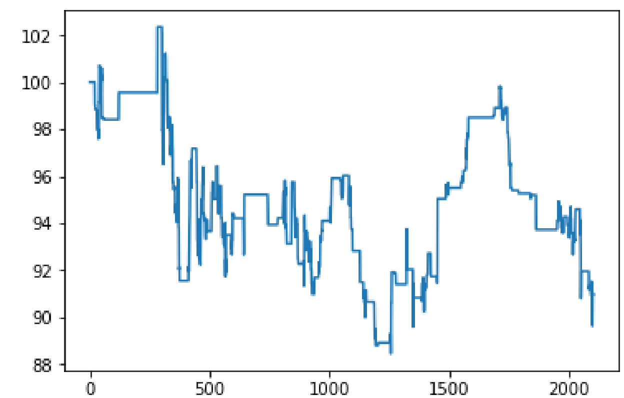
SPY 1 Day Buy



SPY 1 Week Buy



SPY 1 Day Short

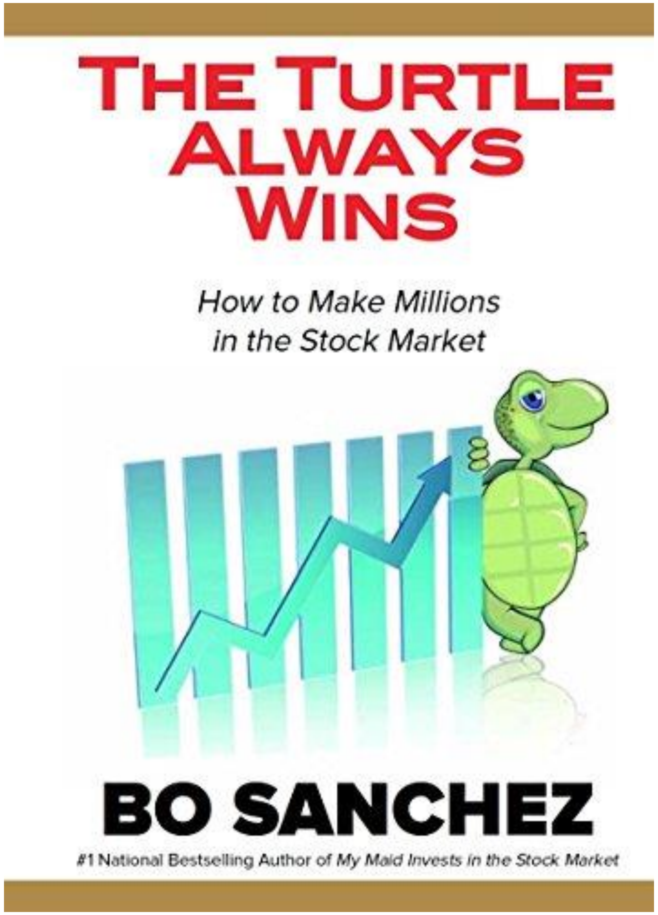


IV. Conclusion

Results

Implications

Next Steps

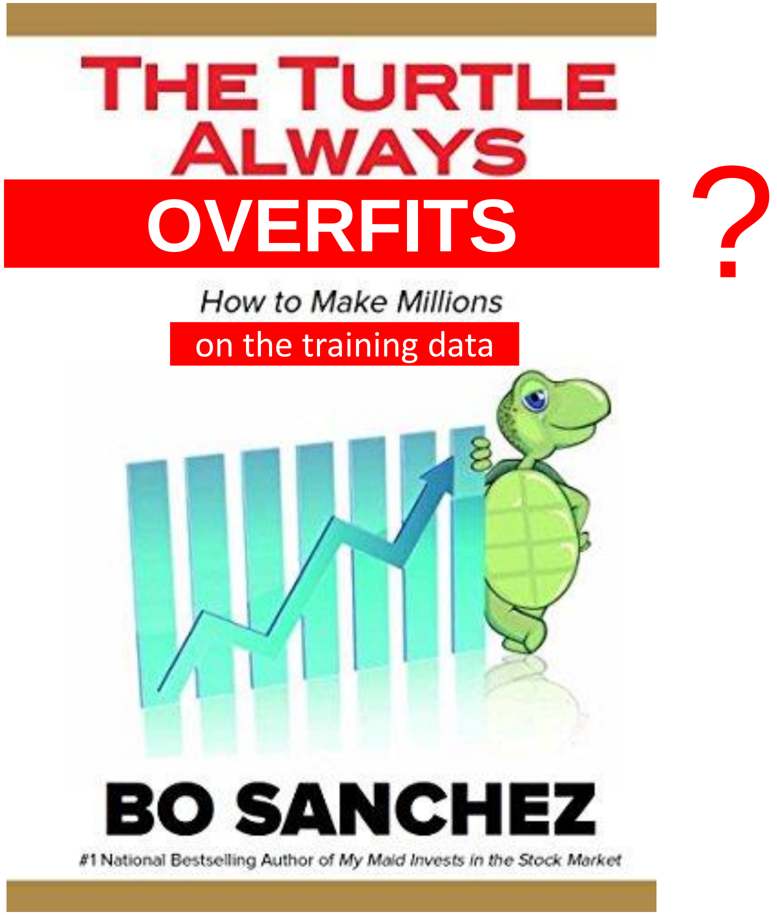


IV. Conclusion

Results

Implications

Next Steps



Picture: https://www.kerygmabooks.com/shop/product/turtle/

The genetic algorithm did not perform as was hoped

Results

- The results from the walk forward tests did not show that the model could consistently find profitable strategies or adapt to changing market conditions

- The strategies that were discovered did not match my expectations and often did not make intuitive sense

- Though we searched a large space, our strategies tended to have few decision variables

This model will not be put into production any time soon, and may have failed for several reasons

Implications

- The data used was all price data – an edge in markets may require data beyond the most public data of all
- Financial markets may be too dynamic to be traded with a model such as the one here
- Overfitting seems to be a major issue in this context

Future iterations of this project might take into account the following suggestions

Next Steps

- Data could be updated
  - Use thresholds of correlation variables to avoid noise, maybe use p values
  - New variables could be more predictive

- A different fitness function could be used
  - A fitness function that took into account the risk profile of the strategy or penalized overfitting in some way might give better results (incorporate Sharpe ratios, max drawdown, comparison to benchmarks, higher spread costs)

- Other train/test timeframes might be more appropriate
  - It is possible that the 60/60 split is not ideal

- Gain access to more computing power
  - Iteration would be much easier at faster speeds!

Resources

- Davies, Kevin J. 2014. Building Winning Algorithmic Trading Systems. Hoboken: John Wiley and Sons.

- Holland, John. 2014. Signals and Boundaries: Building Blocks for Complex Adaptive Systems. Cambridge: The MIT Press.

- Shiffman, Daniel. 2012. The Nature of Code. The Nature of Code.