COMPUTATIONAL FINANCE & RISK MANAGEMENT

UNIVERSITY *of* WASHINGTON

Department of Applied Mathematics

Strategy Optimization and Rules: Supplemental

CFRM 522

Introduction to Trading Systems

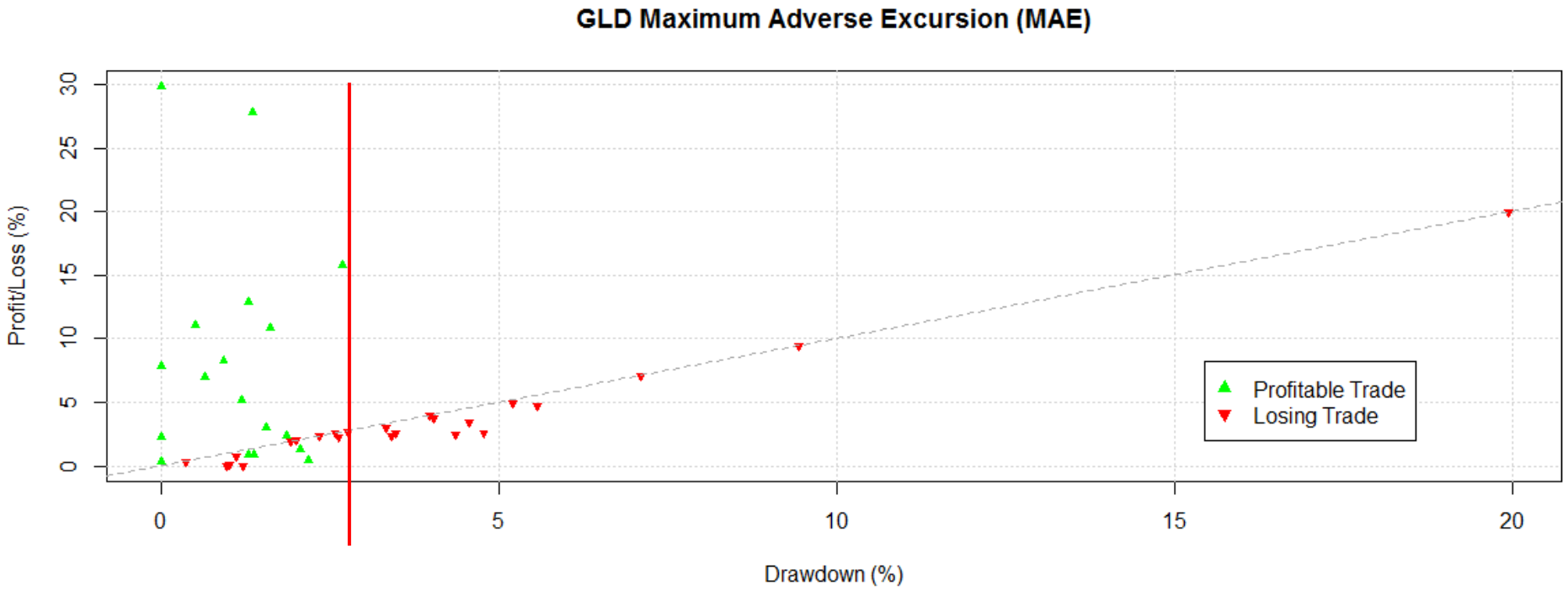
Lecture References

- Ilya Kipnis, *Nuts and Bolts of Quantstrat, Part IV*

https://quantstrattrader.wordpress.com/2014/09/24/nuts-and-bolts-of-quantstrat-part-iv/

- Note: Parts I – III, and Part V, are also good references
- However, be aware there are occasional places where the author supplements the code with functions in his own package that enhance quantstrat (eg `sigAND(.)` in Part III)

Manual Optimization: Setting Stop Loss Order Percentage (GLD)



Manual Optimization: Setting Stop Loss Order Percentage (GLD)

```
# No stop-loss:
#     Net.Trading.PL Profit.To.Max.Draw Max.Drawdown Ann.Sharpe
# GLD          14160           1.622737    -8725.998   3.027645

# With 2% stop-loss (PL results are actually worse):
#     Net.Trading.PL Profit.To.Max.Draw Max.Drawdown Ann.Sharpe
# GLD       12086.04            1.68341    -7179.501   2.974293

# With 3% stop-loss (PL results are actually worse):
#     Net.Trading.PL Profit.To.Max.Draw Max.Drawdown Ann.Sharpe
# GLD       10927.92           1.413619    -7730.459   2.794993

# With 4% stop-loss (PL results and other metrics substantially improved)
# Net.Trading.PL Profit.To.Max.Draw Max.Drawdown Ann.Sharpe
# GLD       16907.76           2.092172    -8081.439   3.846536
```

`type` parameter

- We have seen the type parameter in both `add.rule(.)`, and in `enable.rule(.)`

- In particular, we set `type = 'chain'`

- This setting is used for setting up stop-loss and trailing stop orders, as we saw

`type` parameter

- The other commonly used settings for `type` are
  - `'enter'`
  - `'exit'`

- These are self-explanatory (as we have seen already)

- We will use these in `enable.rule(.)` soon for optimizing moving average indices in strategies such as MACD, Bollinger Bands, dual MA crossovers ("Luxor Jr"), and Luxor itself.

Further Details on `add.rule(.)` and `enable.rule(.)`

- In Part IV of the lecture reference, there is a very good overview of the settings used in `add.rule(.)`, and by extension `enable.rule(.)`

- Some of the main points follow on the next slides

Further Details on `add.rule(.)` and `enable.rule(.)`

- The `name` of the strategy function is almost always **`"ruleSignal"`**
- The arguments to `ruleSignal`:
  - a) The signal column (`sigCol`)
  - b) the value that signals a trigger (`sigVal`)
  - c) the order type (`ordertype`)
  - d) the order side (`orderside`)
  - e) to replace any other open signal (`replace`)
  - f) The order quantity (`orderqty`) is no order-sizing function is used.
  - g) the preferred price (defaults to `Close`, but we need to use a different setting for the Luxor strategy. Note: Kipnis prefers using the open price)
  - h) the order sizing function (`osFUN`)
  - i) the arguments to the order-sizing function.
  - j) Focus is on market orders for Kipnis' post; we have seen an example, however, of implementing stop loss and trailing stop orders

Further Details on `add.rule(.)` and `enable.rule(.)`

- Remaining arguments for `add.rule(.)`
  - The rule type (**`type`**)
    - Will comprise either "`enter`" or "`exit`" for most strategies
    - "`chain`" for OCO (stop loss orders in particular)
  - The **`path.dep`** argument is ***always*** TRUE.
  - The **`label`** for the rule. Not necessary if your entry and exit rules are your absolute final points of logic in your backtest; however, if you wish to use stop-loss orders, then the rules need labels, as we have already seen.

Further Details on `add.rule(.)` and `enable.rule(.)`

- While most of the logic to adding your basic rule is almost always boilerplate outside the arguments to **`ruleSignal`**, *it's the arguments to* **`ruleSignal`** *that allow users to customize rules*.

  - The **`sigCol`** argument is a string that has the exact name of the signal column (in `mktdata`) that you wish to use to generate your entries (or exits) from. This is the same string that went into the label argument of your `add.signal` function calls. *In quantstrat, labels effectively act as logical links between indicators, signals, rules, and more.*

  - The **`sigVal`** argument is what value to use to trigger rule logic. Since signal output is (typically, for our class) comprised of ones (TRUE) and zeroes (FALSE), set sigVal to TRUE. The author describes more complicated situations that we won't cover, at least for now.

Further Details on `add.rule(.)` and `enable.rule(.)`

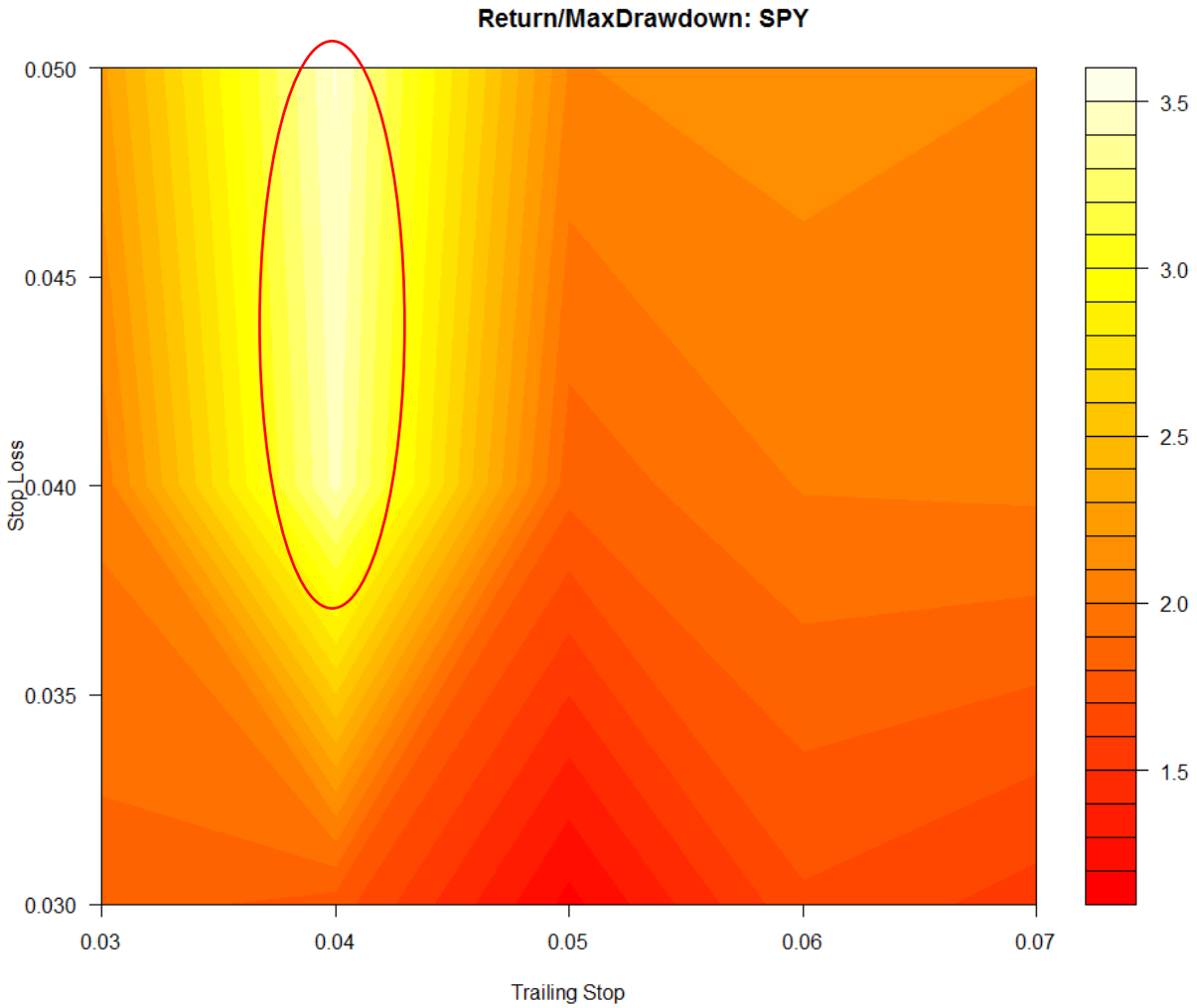
- Arguments to `ruleSignal`, cont'd:
  - The **`ordertype`** argument is the order type. For most of our strategies, we've mostly used "`market`" type orders, which are the simplest. *Market orders execute at the next bar **after** receiving the signal*. We have also now covered stop-loss and trailing stop cases.
  - The **`orderside`** argument takes one of two values–"`long`" or "`short`". This separates rule executions into two bins, such that *long sells won't work on short positions and vice versa*. It also serves to add clarity and readability to strategy specifications.
  - The **`replace`** argument defaults to TRUE
    - Usually set to FALSE; otherwise, it can wreak havoc on entry and exit trades.
    - Only set to TRUE when at the top of a stop-loss chain, in which we *do* want this order to be replaced (see sample code).
  - The **`orderqty`** argument applies only when there's no `osFUN` specified.

Further Details on `add.rule(.)` and `enable.rule(.)`

- Arguments to `ruleSignal`, cont'd:
  - The **`prefer`** argument exists for specifying what aspect of a bar a trade will get in on. quantstrat *by default executes at the close of the next bar*. We would need to modify this for strategies such as Luxor, where we want to wait for a recent high to be hit before entering the market, for example.
  - **`osFUN`** specifies the order-sizing function to use.

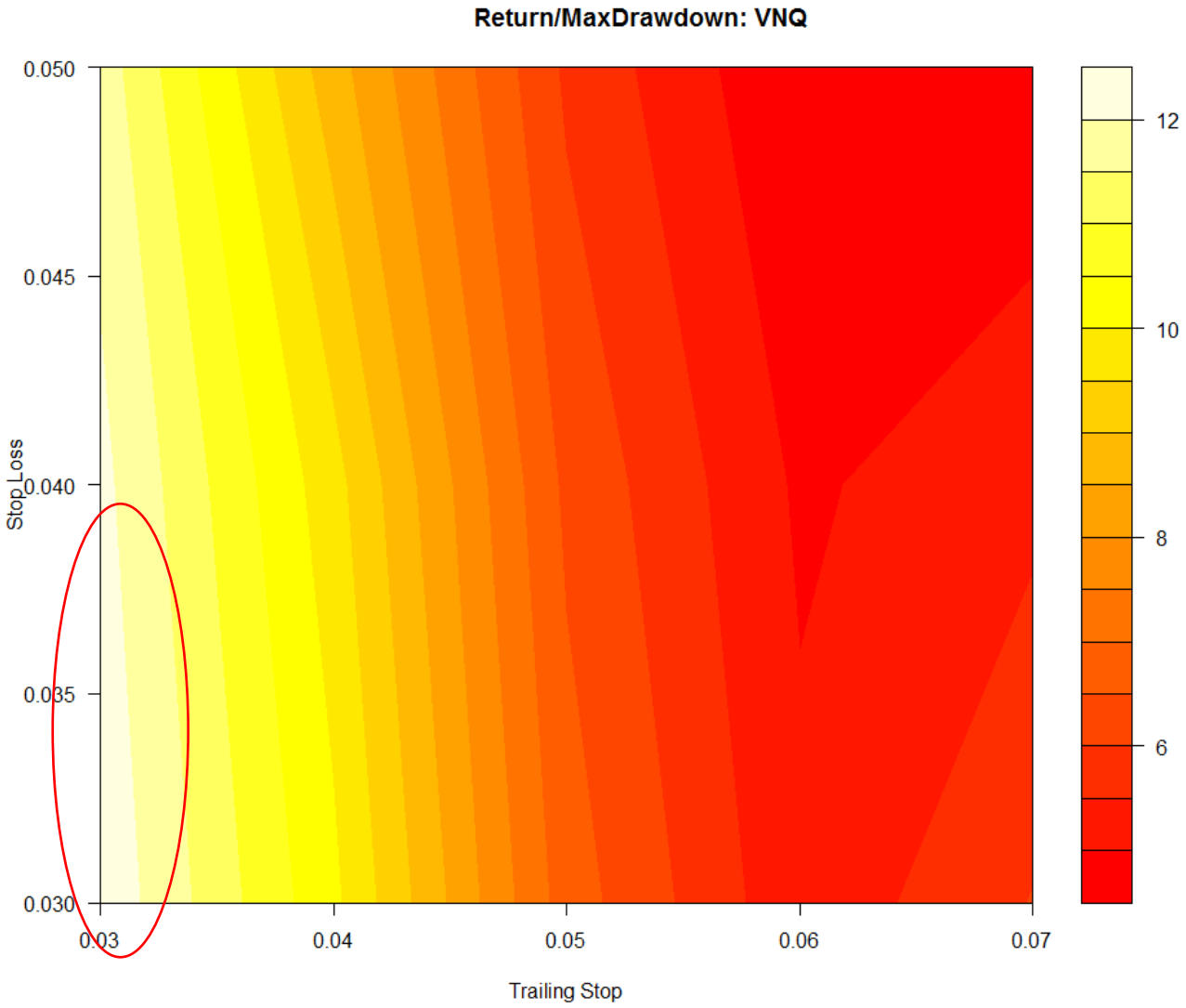
Back to Optimization

- For SPY, we looked at a range of values for the stop-loss and trailing stop percentages. We can see from the heat map that we get a stable region to work with:



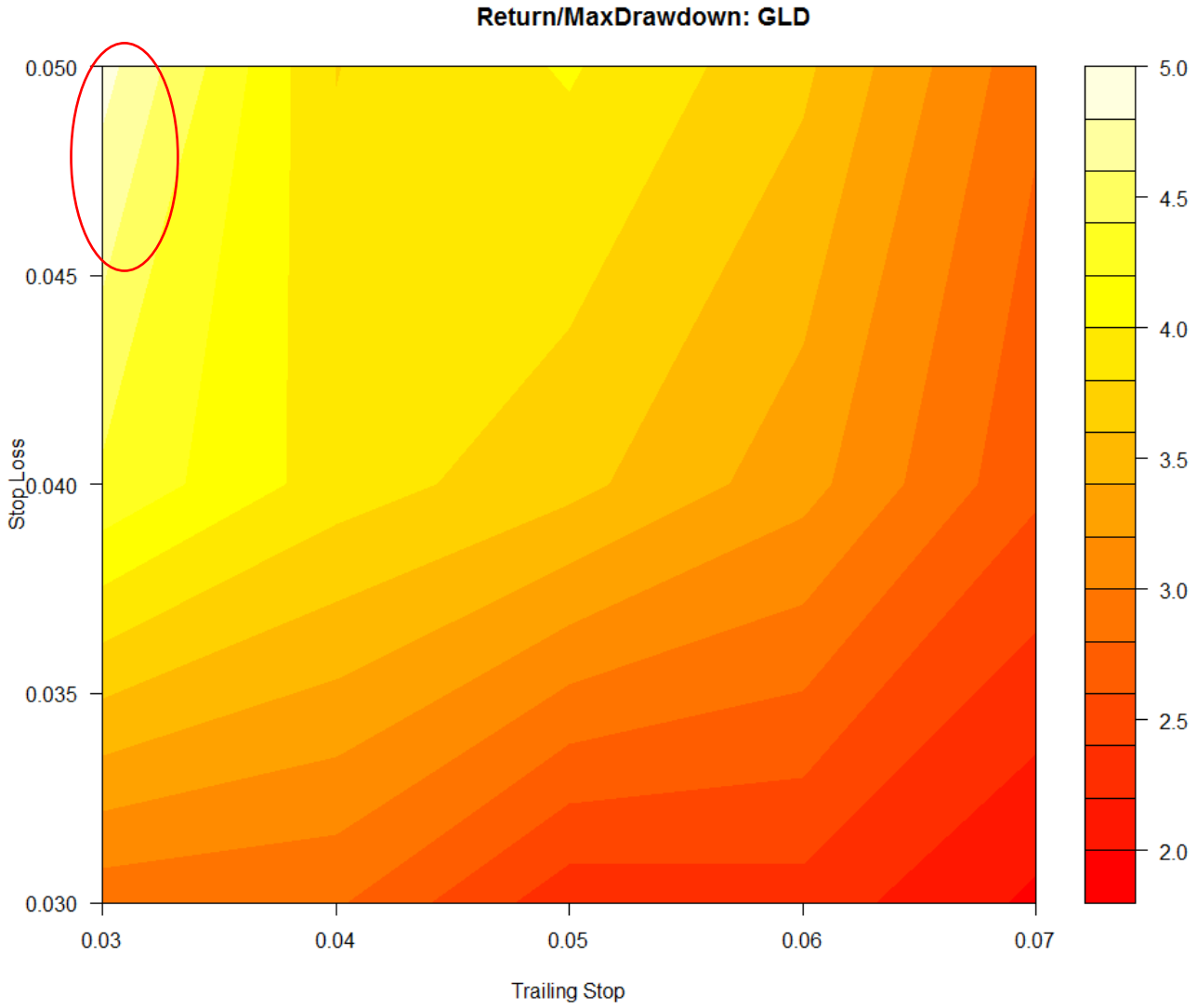
Back to Optimization

- Not so great for VNQ, but at least we have something to work with:



Back to Optimization

- For GLD, however, the results aren't quite so clear.



3-D Visualization

- 3-D visualization is also possible in quantstrat
- Use the tradeGraphs(.) function
- We use the results object for both the heat maps and tradeGraphs(.):

```
# Generate heat map (from Guy Yollin's quantstratIII notes)
# FUN can be median or mean.
z <- tapply(X=results$tradeStats$Profit.To.Max.Draw,
            INDEX=list(results$tradeStats$TrailingLONG,
                       results$tradeStats$StopLossLONG),
            FUN=mean)
```

```
## quantstrat:::tradeGraphs(.)
## Draw 3D graphs from tradeStats results:
tradeGraphs(stats = results$tradeStats,
            free.params = c("TrailingLONG","StopLossLONG"),
            statistics = c("Profit.To.Max.Draw", "Net.Trading.PL",
                           "Max.Drawdown"))
```

- See sample code provided on Canvas

Back to Optimization

- Moral of the story: One size does not fit all.

- **[END]**