STEVENS INSTITUTE of TECHNOLOGY
School of Business

FE 582

Lecture 7: Trading Strategy and Simulation

Dragos Bozdog

Case Study: Trading Strategy and Simulation

Topics:

- Computational Topics
- The Data Format & Reading the Financial Data
- Visualizing the Time Series
- Finding Opening and Closing Positions

    Identifying & Displaying Positions
    Calculating Profit and Loss for a Position
    Optimize

- Simulation Study

Computational Topics

- Designing, writing and testing small, reusable functions.
- Developing efficient code.
- Profiling code and finding the bottlenecks.
- Numerical optimization by grid search.
- Dividing data into training and test subsets to determine optimal values.
- Simulation of a stochastic process.

Combine DataSets

Subset the two stocks and create the data frame of common dates and pairs of stock values. We do this with the function definition:

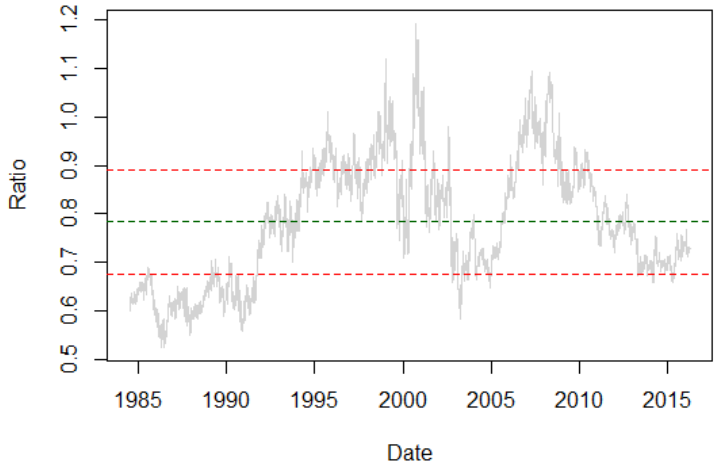
```
combine2Stocks=
function(a, b, stockNames = c(deparse(substitute(a)), deparse(substitute(b))))

rr = range(intersect(a$Date, b$Date))
a.sub = a[ a$Date >= rr[1] & a$Date <= rr[2],]
b.sub = b[ b$Date >= rr[1] & b$Date <= rr[2],]
```

Visualizing Time Series

Plot ratio of two stocks

```
plotRatio =
function(r, k = 1, date = seq(along = r), ...)
{
  plot(date, r, type = "l", ...)
  abline(h = c(mean(r),
               mean(r) + k * sd(r),
               mean(r) - k * sd(r)),
         col = c("darkgreen", rep("red", 2*length(k))),
         lty = "dashed")
}
```



Finding Opening and Closing Positions

- The function takes the entire ratio vector and an optional starting day that is an index into the vector at which we should start the search.

- Specify first position position/index as *1*. After we find the first position, we would specify the day of the end of that position as the starting point from which to search for the next position. In this way, we will move across the time series in blocks, starting from where we ended the previous position.

- The function needs the value for *k* to define the extremes, and it can optionally accept the mean and standard deviation (*m* and *s*) of the ratio. This allows us to compute these statistics just once and avoid recalculating them for each call.

Finding Opening and Closing Positions

- The function starts by determining the upper and lower bound above and below which we would open a position. These are the variables up and down, respectively.

- It then discards all the ratio values up to (but not including) the startDay.

- Compute a logical vector as long as the remaining values in the ratio vector, which tells us if that ratio value is outside the bounds.

Finding Opening and Closing Positions

```
findNextPosition =
  # e.g.,  findNextPosition(r)
  #        findNextPosition(r, 1174)
  # Check they are increasing and correctly offset
function(ratio, startDay = 1, k = 1,
         m = mean(ratio), s = sd(ratio))
{
  up = m + k *s
  down = m - k *s

  if(startDay > 1)
     ratio = ratio[ - (1:(startDay-1)) ]

  isExtreme = ratio >= up | ratio <= down

  if(!any(isExtreme))
      return(integer())

  start = which(isExtreme)[1]
```

Finding Opening and Closing Positions

```
backToNormal = if(ratio[start] > up)
                   ratio[ - (1:start) ] <= m
               else
                   ratio[ - (1:start) ] >= m

  # return either the end of the position or the index
  # of the end of the vector.
  # Could return NA for not ended, i.e. which(backToNormal)[1]
  # for both cases. But then the caller has to interpret that.

  end = if(any(backToNormal))
          which(backToNormal)[1] + start
        else
          length(ratio)

  c(start, end) + startDay - 1
}
```

Finding Opening and Closing Positions

- Now we have our findNextPosition() function and we can and need to test it.
- We specify *k* and then call it:

```
k = .85
a = findNextPosition(r, k = k)
```

- This gives us the first position we open and close and we'll assign that to *a* .

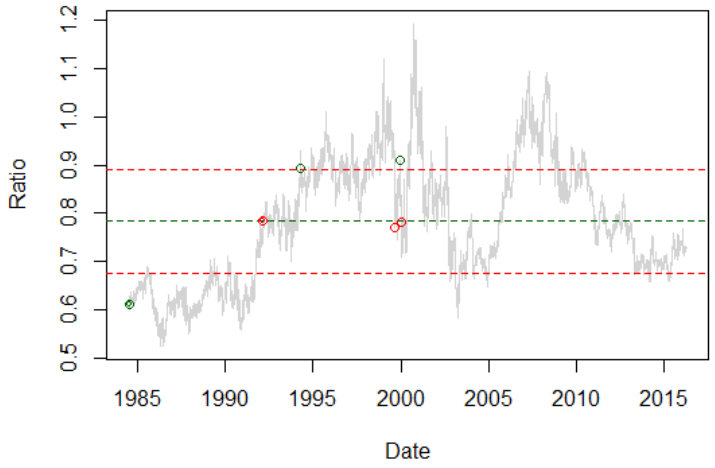
- For the next position, we call findNextPosition() again, but this time specify the starting point as the end of the previous position,

```
a = findNextPosition(r, k = k)
b = findNextPosition(r, a[2], k = k)
c = …
```

Displaying Positions

- We can pass a vector of *x* and *y* coordinates in a single call to symbols() to draw two circles for the start and end dates. We can create a function for this as

```
showPosition =
function(days, ratios, radius = 100)
{
  symbols(days, ratios, circles = rep(radius, 2),
          fg = c("darkgreen", "red"), add = TRUE, inches = FALSE)
}
```



Finding All Positions

```
getPositions =
function(ratio, k = 1, m = mean(ratio), s = sd(ratio))
{
   when = list()
   cur = 1

   while(cur < length(ratio)) {
      tmp = findNextPosition(ratio, cur, k, m, s)
      if(length(tmp) == 0)  # done
         break
      when[[length(when) + 1]] = tmp
      if(is.na(tmp[2]) || tmp[2] == length(ratio))
         break
      cur = tmp[2]
    }

   when
}
```

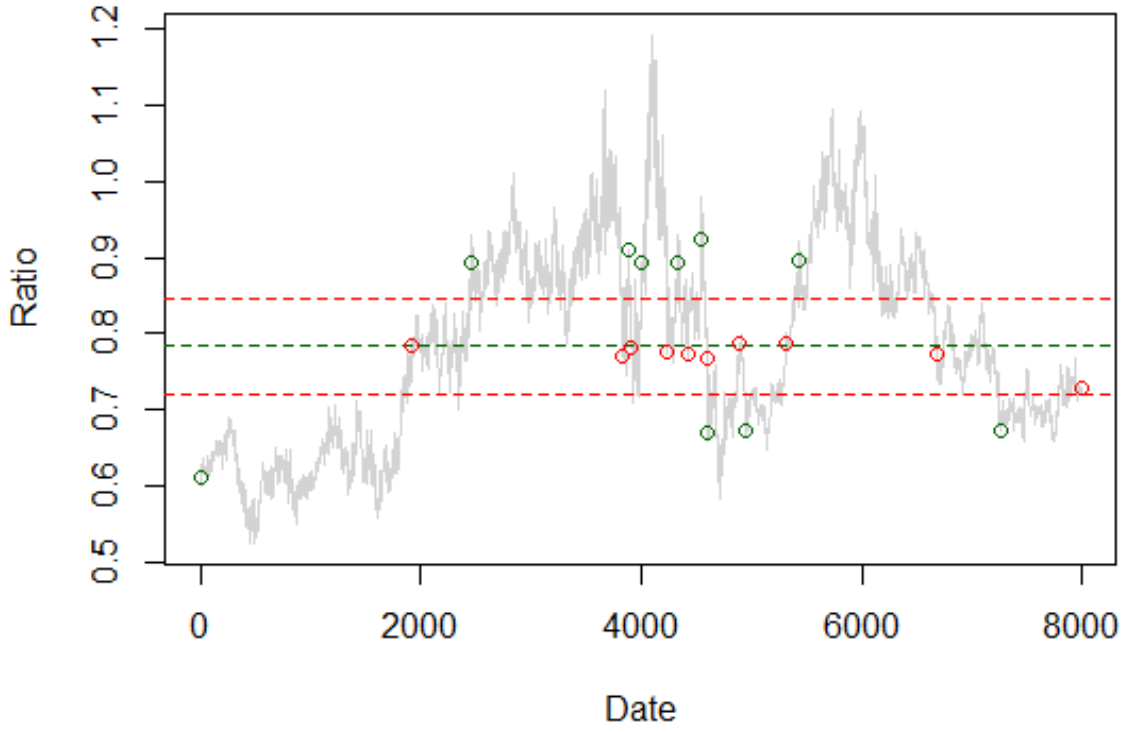
Visualizing All Positions

```
getPositions =
function(ratio, k = 1, m = mean(ratio), s = sd(ratio))
{
   when = list()
   cur = 1

   while(cur < length(ratio)) {
      tmp = findNextPosition(ratio, cur, k, m, s)
      if(length(tmp) == 0)  # done
         break
      when[[length(when) + 1]] = tmp
      if(is.na(tmp[2]) || tmp[2] == length(ratio))
         break
      cur = tmp[2]
    }

   when
}
```

Visualizing All Positions



Compute Profit for a Position

- We determine whether, at the opening of the position, we are selling A or selling B. This depends on whether the ratio is high or low, respectively.

```
positionProfit =
  #  r = overlap$att/overlap$verizon
  #  k = 1.7
  #  pos = getPositions(r, k)
  #  positionProfit(pos[[1]], overlap$att, overlap$verizon)
function(pos, stockPriceA, stockPriceB,
         ratioMean = mean(stockPriceA/stockPriceB),
         p = .001, byStock = FALSE)
{
  if(is.list(pos)) {
    ans = sapply(pos, positionProfit,
                 stockPriceA, stockPriceB, ratioMean, p, byStock)
    if(byStock)
      rownames(ans) = c("A", "B", "commission")
    return(ans)
  }
```

Compute Profit for a Position

```
# prices at the start and end of the positions
  priceA = stockPriceA[pos]
  priceB = stockPriceB[pos]
    # how many units can we by of A and B with $1
  unitsOfA = 1/priceA[1]
  unitsOfB = 1/priceB[1]
    # The dollar amount of how many units we would buy of A and B
    # at the cost at the end of the position of each.
  amt = c(unitsOfA * priceA[2], unitsOfB * priceB[2])
    # Which stock are we selling
  sellWhat = if(priceA[1]/priceB[1] > ratioMean) "A" else "B"
  profit = if(sellWhat == "A")
             c((1 - amt[1]),  (amt[2] - 1), - p * sum(amt))
           else
             c( (1 - amt[2]),  (amt[1] - 1),  - p * sum(amt))
  if(byStock)
     profit
  else
     sum(profit)
}
```

Finding the Optimal Value for k

```
# prices at the start and end of the positions
  priceA = stockPriceA[pos]
  priceB = stockPriceB[pos]
    # how many units can we by of A and B with $1
  unitsOfA = 1/priceA[1]
  unitsOfB = 1/priceB[1]
    # The dollar amount of how many units we would buy of A and B
    # at the cost at the end of the position of each.
  amt = c(unitsOfA * priceA[2], unitsOfB * priceB[2])
    # Which stock are we selling
  sellWhat = if(priceA[1]/priceB[1] > ratioMean) "A" else "B"
  profit = if(sellWhat == "A")
             c((1 - amt[1]),  (amt[2] - 1), - p * sum(amt))
           else
             c( (1 - amt[2]),  (amt[1] - 1),  - p * sum(amt))
  if(byStock)
     profit
  else
     sum(profit)
}
```

Finding the Optimal Value for k

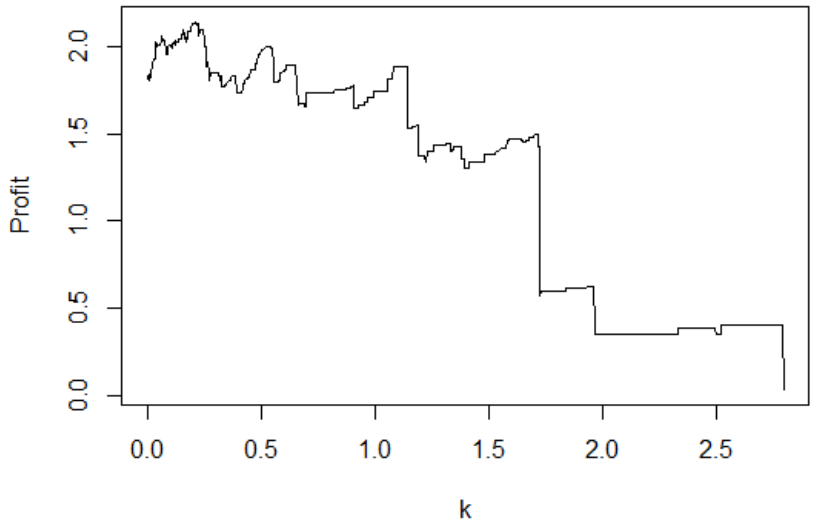
- Find the best value of *k* from training data and then apply it to test data.
- Create the training and test data sets.
- Look at a particular period for the training data, or just split the data in two subsets.

```
i = 1:floor(nrow(overlap)/2)
train = overlap[i, ]
test = overlap[ - i, ]
```

Finding the Optimal Value for k

- Now let's loop over the different values of k and compute the profit for that strategy defined by k:

```
profits =
 sapply(ks,
        function(k) {
          pos = getPositions(r.train, k)
          sum(positionProfit(pos, train$att, train$verizon,
                             mean(r.train)))
        })
plot(ks, profits, type = "l", xlab = "k", ylab = "Profit")
```



Simulation Study

- A very reasonable question an investor should ask when considering whether to use a pairs trading approach is what are the characteristics of the two stocks that yield a high profit margin?

- Consider the following model:

$$X_t^{(1)} = \rho X_{t-1}^{(1)} + \psi(1-\rho)\rho X_{t-1}^{(2)} + \epsilon_t^{(1)}$$
$$X_t^{(2)} = \rho X_{t-1}^{(2)} + \psi(1-\rho)\rho X_{t-1}^{(1)} + \epsilon_t^{(2)}$$

where the error terms $\epsilon_t^{(1)} \sim N\left(0, \sigma_i^2\right)$

- Define the actual stock price as

$$Y_t^{(1)} = \beta_0^{(1)} + \beta_1^{(1)} t + X_t^{(1)}$$
$$Y_t^{(2)} = \beta_0^{(2)} + \beta_1^{(2)} t + X_t^{(2)}$$

Thank You

Dragos Bozdog

For academic use only.