Optimization in Finance

This is sourced from STAD70 course practice questions and sample R code taught by professor Sotos. If you have any questions/concerns/comments feel free to email me: cristal.wang111@gmail.com (mailto:cristal.wang111@gmail.com).

1.Dynamic Programming (I/n)

Use Dynamic Programming to find optimal trading strategy under transaction costs given price evolution

```
library(tseries)
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
St = get.hist.quote( "SHOP.TO", start = "2023-01-01", end = "2023-03-31", quote = "Close"
)
```

```
## time series starts 2023-01-03
## time series ends   2023-03-30
```

```
plot(St, lwd=2, type = "o", pch=16, main = "Shopify Close Price")
```


Shopify Close Price
St
50
55
60
65
70
Jan
Feb
Mar
Apr
Index

```
tc = .1 # cost per transaction

n = length(St)

# optimal value function
n.val = rep(0,n)  # neutral (n)
l.val = rep( as.numeric(St[n])-tc, n ) # long (l)

# optimal strategy
n.str = rep(0,n)
l.str = rep(1,n)

# backward induction
for( i in (n-1):1 ){

  n2n = n.val[i+1]
  n2l = l.val[i+1] - as.numeric(St[i]) - tc
  if( n2n >= n2l ){  # action: n -> n
    n.val[i] = n2n
    n.str[i] = 0
  }else{             # action: n -> l
    n.val[i] = n2l
    n.str[i] = 1
    n.str[(i+1):n] = l.str[(i+1):n]
  }

  l2n = n.val[i+1] + as.numeric(St[i]) - tc
  l2l = l.val[i+1]
  if( l2n >= l2l ){  # action: l -> n
    l.val[i] = l2n
    l.str[i] = 0
    l.str[(i+1):n] = n.str[(i+1):n]
  }else{             # action: l -> l
    l.val[i] = l2l
  }
}

# calculate cumulative optimal value
run.val = rep(0, n)
run.val[1] = n.str[1] * ( - as.numeric(St[1]) - tc )
for(i in 2:n){
  if( n.str[i] == n.str[i-1] ){
    run.val[i] = run.val[i-1]
  }else if( n.str[i] > n.str[i-1] ){
    run.val[i] = run.val[i-1] - as.numeric(St[i]) - tc
  }else{
    run.val[i] = run.val[i-1] + as.numeric(St[i]) - tc
  }
}

run.val[n]
```

```
## [1] 56.84
```

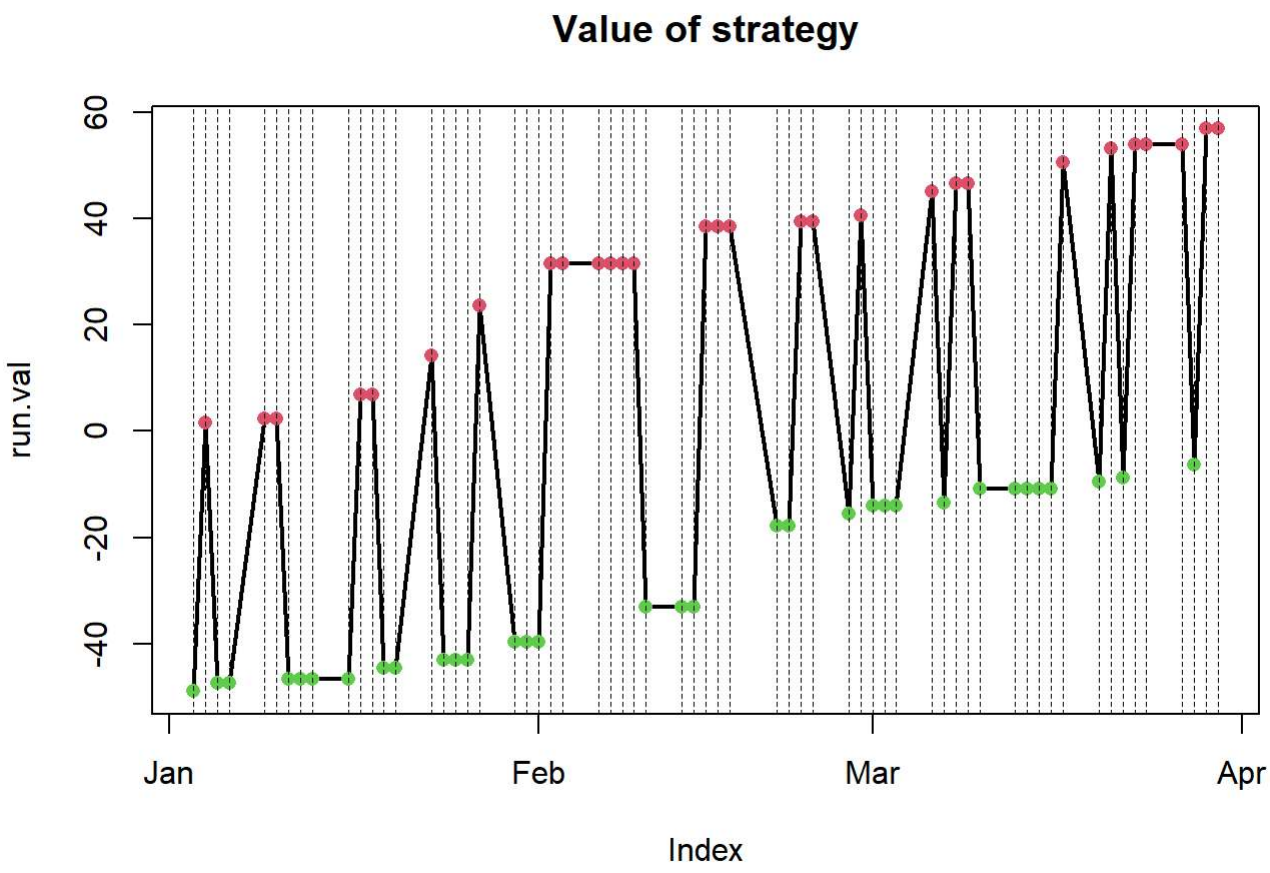
```
n.val[1]
```

```
## [1] 56.84
```

```
# create time series
n.val = zoo::zoo( n.val, order.by = time(St))
n.str = zoo::zoo( n.str, order.by = time(St))
run.val = zoo::zoo( run.val, order.by = time(St))

# plots
plot( St, lwd=2, main = "Strategy")
points( St, col = n.str+2, pch = 16)
abline( v = time(St), lwd=.1, lty=2)
```


```
plot( run.val, lwd=2 , main = "Value of strategy")
points( run.val, col = n.str+2, pch = 16)
abline( v = time(St), lwd=.1, lty=2)
```



2.Dynamic Programming(l/n/s)

Consider the example of trading an asset with transaction costs, assuming you have perfect knowledge of its price. Adapt the code from lecture 12 to find the optimal strategy and its cumulative P/L, assuming you can also short the asset, i.e. you have to consider three states: long, neutral, and short. Note that when you go from long to short position, or vice-versa, you have to pay twice the transaction cost because you sell/buy two units of asset.

Test your code on Shopify (SHOP.TO) closing daily prices from Jan 1 to Mar 31, 2023 with a transaction cost of $0.1/share, and plot the strategy.

```
library(tseries)

St = get.hist.quote( "SHOP.TO", start = "2023-01-01", end = "2023-03-31", quote = "Close"
)
```

```
## time series starts 2023-01-03
## time series ends   2023-03-30
```

```
plot(St, lwd=2, type = "o", pch=16, main = "Shopify Close Price")
```


Shopify Close Price
St
50
55
60
65
70
Jan
Feb
Mar
Apr
Index

```
tc = 1 # cost per transaction

n = length(St)

# optimal value function
n.val = rep( 0, n ) # neutral (n)
l.val = rep( 0, n ) # long (l)
s.val = rep( 0, n ) # short (s)

l.val[n] =  as.numeric(St[n])-tc
s.val[n] = -as.numeric(St[n])-tc

# optimal strategy
n.str = rep(  0, n )
l.str = rep(  1, n )
s.str = rep( -1, n )

# Backward Induction/Dynamic Programming
# We have 3 options/transitions to consider for each state:

for( i in (n-1):1 ){

  # entering at n, compare: n2n, n2l, n2s
  n2n = n.val[i+1]
  n2l = l.val[i+1] - as.numeric(St[i]) - tc
  n2s = s.val[i+1] + as.numeric(St[i]) - tc
  if( n2n >= max( n2l, n2s) ){          # action: n -> n
    n.val[i] = n2n
    n.str[i] = 0
  }else if( n2l >= max( n2n, n2s) ){    # action: n -> l
    n.val[i] = n2l
    n.str[i] = 1
    n.str[(i+1):n] = l.str[(i+1):n]
  }else{                                # action: n -> s
    n.val[i] = n2s
    n.str[i] = -1
    n.str[(i+1):n] = s.str[(i+1):n]
  }

  # entering at l, compare: l2n, l2l, l2s
  l2n = n.val[i+1] + as.numeric(St[i]) - tc
  l2l = l.val[i+1]
  l2s = s.val[i+1] + 2*( as.numeric(St[i]) - tc )
  if( l2n >= max( l2l, l2s ) ){        # action: l -> n
    l.val[i] = l2n
    l.str[i] = 0
    l.str[(i+1):n] = n.str[(i+1):n]
  }else if( l2l >= max( l2n, l2s ) ){  # action: l -> l
    l.val[i] = l2l
  }else{
    l.val[i] = l2s                     # action: l -> s
    l.str[i] = -1
```

```
    l.str[(i+1):n] = s.str[(i+1):n]
  }

  # entering at s, compare: s2n, s2l, s2s
  s2n = n.val[i+1] - as.numeric(St[i]) - tc
  s2l = l.val[i+1] + 2*(-as.numeric(St[i]) - tc)
  s2s = s.val[i+1]
  if( s2n >= max( s2l, s2s ) ){        # action: s -> n
    s.val[i] = s2n
    s.str[i] = 0
    s.str[(i+1):n] = n.str[(i+1):n]
  }else if( s2l >= max( s2n, s2s ) ){  # action: s -> l
    s.val[i] = s2l
    s.str[i] = 1
    s.str[(i+1):n] = n.str[(i+1):n]
  }else{
    s.val[i] = s2s                     # action: s -> s
    s.str[i] = -1
  }

}

# calculate cumulative optimal value
run.val = rep(0, n)
run.val[1] = - n.str[1] * as.numeric(St[1]) - abs(n.str[1]) * tc
for(i in 2:n){
  if( n.str[i] == n.str[i-1] ){
    run.val[i] = run.val[i-1]
  }else if( n.str[i] > n.str[i-1] ){
    run.val[i] = run.val[i-1] + (n.str[i] - n.str[i-1]) * (-as.numeric(St[i]) - tc)
  }else{
    run.val[i] = run.val[i-1] + (n.str[i-1] - n.str[i]) * (as.numeric(St[i]) - tc)
  }
}

run.val[n]
```

```
## [1] 58.52
```

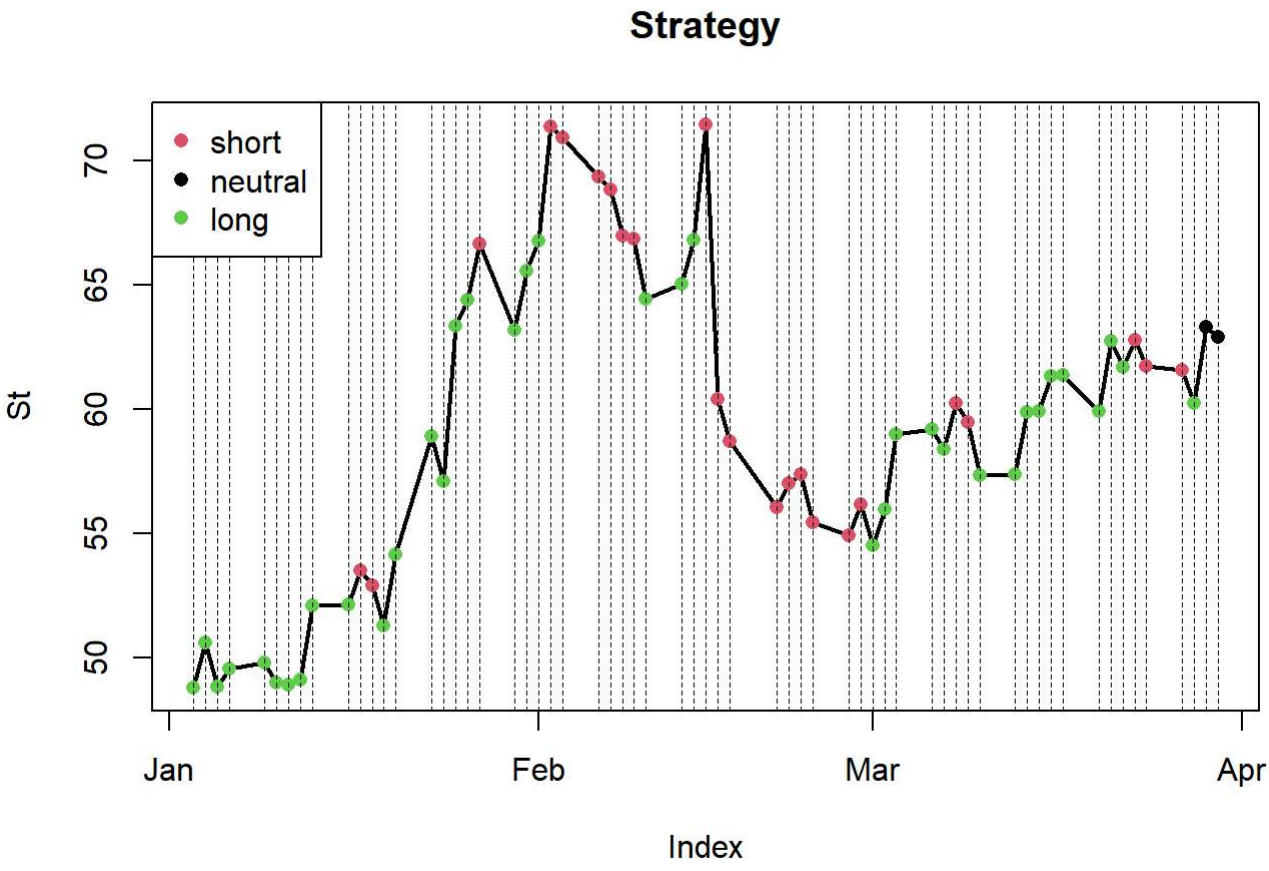
```
n.val[1]
```

```
## [1] 58.52
```

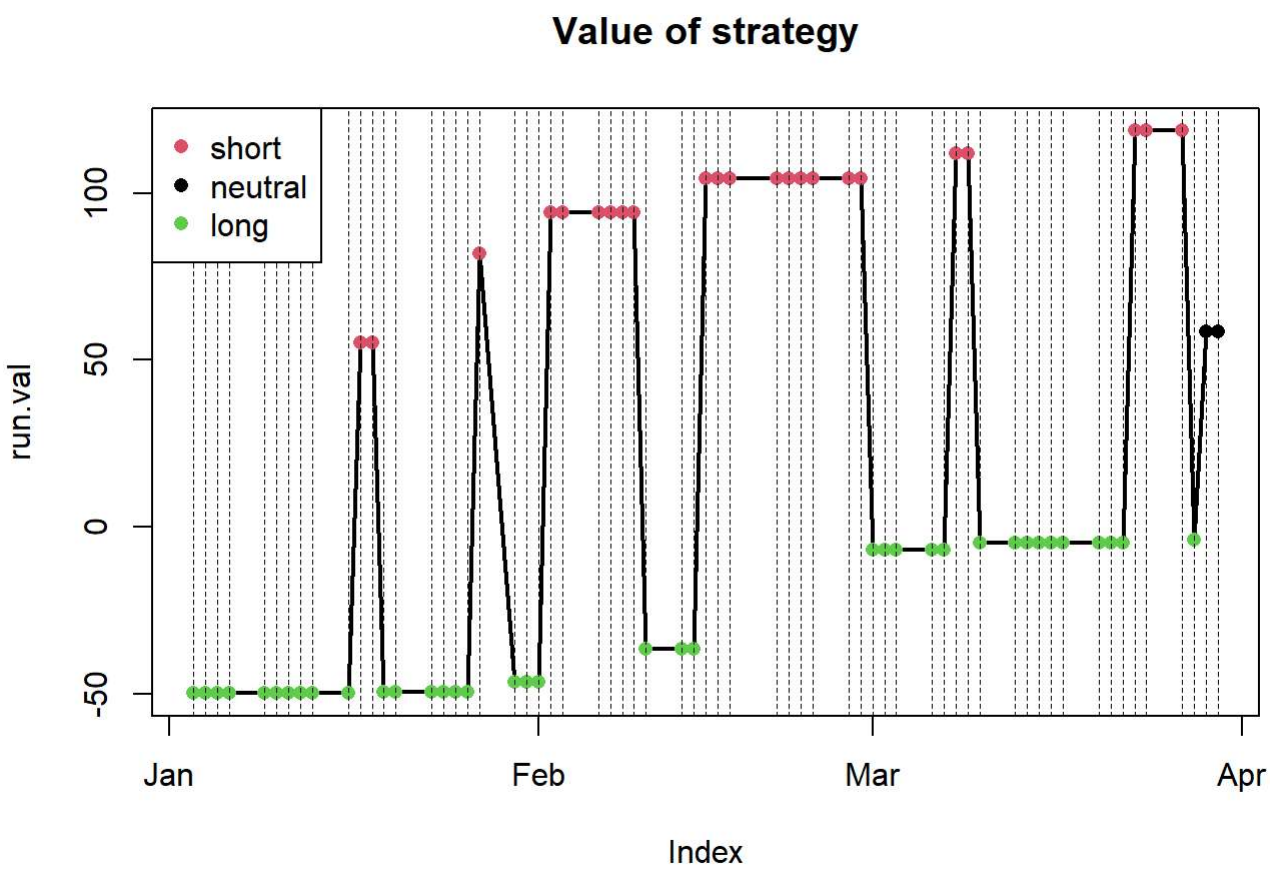
```
# create time series
n.val = zoo::zoo( n.val, order.by = time(St))
n.str = zoo::zoo( n.str, order.by = time(St))
run.val = zoo::zoo( run.val, order.by = time(St))

# plots
tmp = 1 + n.str * 2
tmp = replace( tmp, n.str == -1, 2 )

plot( St, lwd=2, main = "Strategy")
points( St, col = tmp, pch = 16)
abline( v = time(St), lwd=.1, lty=2)
legend( "topleft", pch=16, col = c(2,1,3), c("short","neutral","long"))
```




```
plot( run.val, lwd=2 , main = "Value of strategy")
points( run.val, col = tmp, pch = 16)
abline( v = time(St), lwd=.1, lty=2)
legend( "topleft", pch=16, col = c(2,1,3), c("short","neutral","long"))
```


Value of strategy
short
neutral
long
run.val
100
50
0
-50
Jan
Feb
Mar
Apr
Index