5 Way Rank Probabilities

Fred Viole

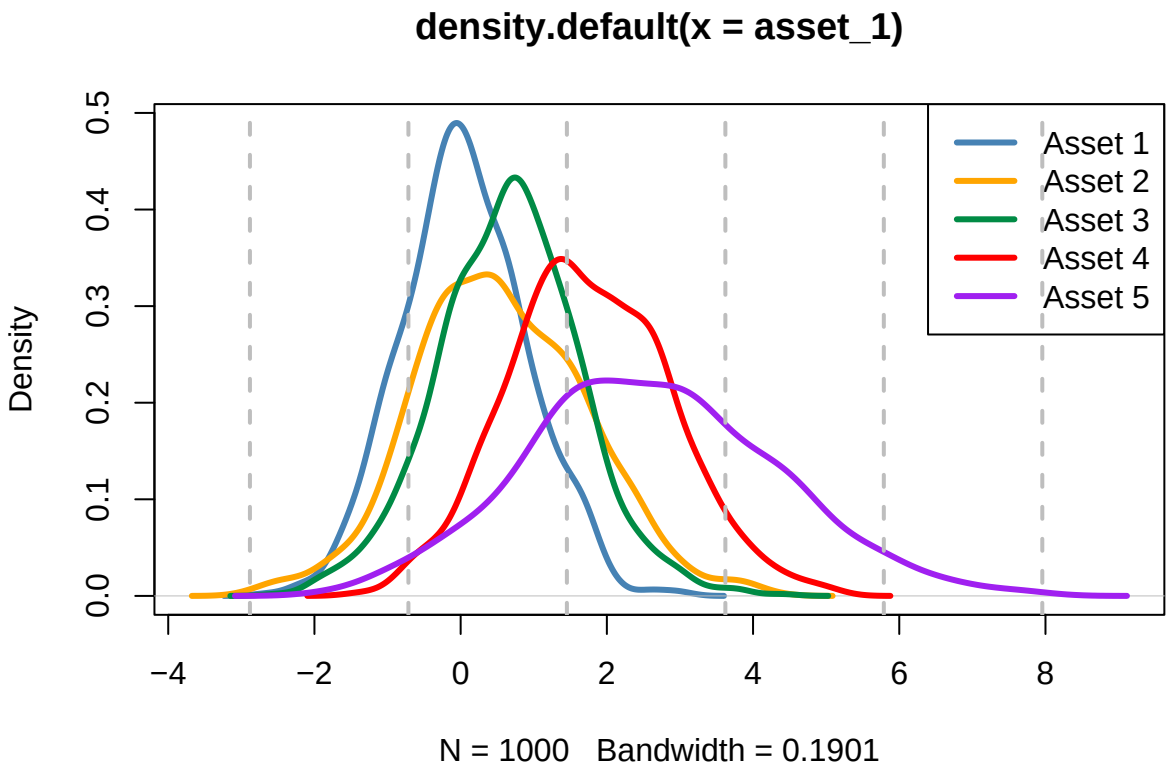
12/9/2021

Generate Data

```
library(data.table)
library(sn)
library(NNS)

set.seed(12345)
n = 1000
asset_1 <- rsn(n, xi = -.5, omega = 1, alpha = 1)
asset_2 <- rsn(n, xi = -.25, omega = 1.5, alpha = 1)
asset_3 <- rsn(n, xi = 0, omega = 1.2, alpha = 1)
asset_4 <- rsn(n, xi = 1, omega = 1.3, alpha = 1)
asset_5 <- rsn(n, xi = 1.5, omega = 2, alpha = 1)

assets <- setDT(list(asset_1, asset_2, asset_3, asset_4, asset_5))

# Grid values
cutpoints = seq(min(assets), max(assets), length.out = 6)
```

Plot

Grey dashed lines are the target value all `UPM` calculations are based off of for each density and then averaged. This is analogous to the shifted distribution $f^{\rightarrow a}(.)$.



Generate Probabilities

`rank_5` based on relative shifted densities measured from several lattice points

```
# UPM from min value
upms = apply(assets, 2, function(x) UPM(1, cutpoints, x))
upms = cbind(upms, rowSums(upms))
upms = upms/upms[,ncol(upms)]
upms = upms[complete.cases(upms),-ncol(upms)]

rank_5_probabilities = colMeans(upms)
rank_5_probabilities
```

```
##         V1         V2         V3         V4         V5
## 0.04909595 0.07822471 0.07570879 0.16709926 0.62987130
```

MC Verification

```
replicates = 1000
races = replicate(replicates, sample(1:5, 5, replace = F, prob = rank_5_probabilities))


Results = matrix(NA, 5,5)

for(i in 1:5){
  for(j in 1:5){
    Results[i,(6-j)] = sum(races[j,]==i)/replicates
  }
}

colnames(Results) = paste0("Rank_", 1:5)
rownames(Results) = paste0("Asset_", 1:5)
Results
```

```
##         Rank_1 Rank_2 Rank_3 Rank_4 Rank_5
## Asset_1  0.449  0.259  0.156  0.094  0.042
## Asset_2  0.232  0.290  0.241  0.160  0.077
## Asset_3  0.243  0.276  0.270  0.135  0.076
## Asset_4  0.076  0.161  0.265  0.328  0.170
## Asset_5  0.000  0.014  0.068  0.283  0.635
```

```
# sanity check
rowSums(Results)
```

```
## Asset_1 Asset_2 Asset_3 Asset_4 Asset_5
##       1       1       1       1       1
```

```
colSums(Results)
```

```
## Rank_1 Rank_2 Rank_3 Rank_4 Rank_5
##      1      1      1      1      1
```

Ranks from your notebook with $\rho = 0$.

```
[10] a2l.to_ltx(RP)
```

```
\begin{tabular}{lrrrrr}
\toprule
     & Rank 1 & Rank 2 & Rank 3 & Rank 4 & Rank 5 \\\n\midrule
 Asset 1 &  0.37 &  0.33 &  0.20 &  0.08 &  0.02\\
 Asset 2 &  0.32 &  0.24 &  0.21 &  0.15 &  0.08\\
 Asset 3 &  0.20 &  0.26 &  0.28 &  0.19 &  0.07\\
 Asset 4 &  0.04 &  0.10 &  0.19 &  0.37 &  0.31\\
 Asset 5 &  0.07 &  0.08 &  0.12 &  0.21 &  0.52\\
\bottomrule
\end{tabular}
```

Sampling with Dependence

Alternatively we can just infer the probabilities from the ranks of the tuple elements themselves...

```
empirical_samples = cbind(assets,
                  t(apply(assets, 1, function(x) order(x, decreasing = F))))

colnames(empirical_samples) = c(paste0("Asset_", 1:5),
                                paste0("Rank_", 1:5))

empirical_samples
```

```
##           Asset_1    Asset_2     Asset_3    Asset_4    Asset_5 Rank_1 Rank_2
##    1:  0.41569963  1.5362199  0.72866722  0.3777486  0.4843671      4      1
##    2: -0.74338181  1.5266611 -0.41525272  2.5461900  0.1822763      1      3
##    3: -1.35706186  1.2815910  0.96831269  0.7305048 -0.3425123      1      5
##    4: -0.24974470  0.4594755 -0.10040605  0.6829659 -0.8650106      5      1
##    5: -0.94912754 -0.5656314  1.52439859  2.0519467  2.2742939      1      2
##   ---
##  996:  0.30423320  1.8321389 -0.04146175  2.2884961  5.1382054      3      1
##  997: -0.01683564  1.6197419  0.83305115 -0.2918981  2.2538016      4      1
##  998:  0.95846702  0.1406389  0.57545219  1.9620473  4.0652914      2      3
##  999: -0.32830364  1.0182948  1.14756094  1.5355784  4.8748741      1      2
## 1000: -0.91661260  0.2310668  0.51263975  1.3483795  3.7713607      1      2
##       Rank_3 Rank_4 Rank_5
##    1:      5      3      2
##    2:      5      2      4
##    3:      4      3      2
##    4:      3      2      4
##    5:      3      4      5
##   ---
##  996:      2      4      5
##  997:      3      2      5
##  998:      1      4      5
##  999:      3      4      5
## 1000:      3      4      5
```

```
Results = matrix(NA, 5,5)

for(i in 1:5){
  for(j in 6:10){
    Results[i,(j-5)] = sum(empirical_samples[, .SD, .SDcols = j]==i) / n
  }
}

colnames(Results) = paste0("Rank_", 1:5)
rownames(Results) = paste0("Asset_", 1:5)
Results
```

```
##         Rank_1 Rank_2 Rank_3 Rank_4 Rank_5
## Asset_1  0.451  0.302  0.174  0.063  0.010
## Asset_2  0.282  0.253  0.236  0.166  0.063
## Asset_3  0.190  0.281  0.305  0.167  0.057
## Asset_4  0.041  0.100  0.186  0.387  0.286
## Asset_5  0.036  0.064  0.099  0.217  0.584
```

Increasing the empirical samples via `meboot`

If we have small samples, we can increase them using the maximum entropy bootstrap which preserves the dependence structure within time series.

We started off with 1,000 observations, let's make it 10,000 using $\rho = 0.95$ for each bootstrap replicate.

```
bootstraps = do.call(cbind, lapply(assets, function(x)
  as.vector(NNS.meboot(unlist(x), reps = 10, rho = .95)$replicates)))


bootstrap_samples = cbind(bootstraps,
                  t(apply(bootstraps, 1, function(x) order(x, decreasing = F))))

bootstrap_samples = data.table(bootstrap_samples)

colnames(bootstrap_samples) = c(paste0("Asset_", 1:5),
                                paste0("Rank_", 1:5))

bootstrap_samples
```

```
##             Asset_1    Asset_2    Asset_3    Asset_4   Asset_5 Rank_1 Rank_2
##     1:  0.04582999  1.4119126  1.2994833  0.8410055 0.6990538      1      5
##     2: -1.04242568  0.1865475 -0.6226921  2.9006947 1.9091502      1      3
##     3: -1.44204830  1.2410014  0.8214673  1.1737755 0.5496580      1      5
##     4: -0.56487754  0.1005746 -0.0426594  1.5614354 0.1737312      1      3
##     5: -1.32079976 -0.8427943  1.8170268  1.7667717 2.6971145      1      2
##    ---
##  9996:  0.28158951  1.7590540  0.7101065  2.2161565 5.6763253      1      3
##  9997: -0.04713496  1.5389258  1.7241096 -0.3286118 2.6452641      4      1
##  9998:  1.34310786  0.5924532  0.9536630  1.6232472 3.6180842      2      3
##  9999: -0.10432566  1.1448038  1.4546092  2.2970129 4.5833049      1      2
## 10000: -0.84067000  0.7205704  0.7035229  1.4163911 3.0643839      1      3
```

```
##         Rank_3 Rank_4 Rank_5
##     1:      4      3      2
##     2:      2      5      4
##     3:      3      4      2
##     4:      2      5      4
##     5:      4      3      5
##    ---
##  9996:      2      4      5
##  9997:      2      3      5
##  9998:      1      4      5
##  9999:      3      4      5
## 10000:      2      4      5
```

```
Results = matrix(NA, 5,5)

for(i in 1:5){
  for(j in 6:10){
    Results[i,(j-5)] = sum(bootstrap_samples[, .SD, .SDcols = j]==i) / nrow(bootstraps)
  }
}

colnames(Results) = paste0("Rank_", 1:5)
rownames(Results) = paste0("Asset_", 1:5)
Results
```

```
##          Rank_1 Rank_2 Rank_3 Rank_4 Rank_5
## Asset_1 0.4384 0.2888 0.1883 0.0699 0.0146
## Asset_2 0.2692 0.2579 0.2295 0.1708 0.0726
## Asset_3 0.1963 0.2736 0.2922 0.1658 0.0721
## Asset_4 0.0447 0.1038 0.1893 0.3876 0.2746
## Asset_5 0.0514 0.0759 0.1007 0.2059 0.5661
```

Asymptotic properties need to be investigated, here's 1e6 replicates:

```
bootstraps2 = do.call(cbind, lapply(assets, function(x)
  as.vector(NNS.meboot(unlist(x), reps = 1000, rho = .95)$replicates)))


bootstrap_samples2 = cbind(bootstraps2,
                  t(apply(bootstraps2, 1, function(x) order(x, decreasing = F))))

bootstrap_samples2 = data.table(bootstrap_samples2)


Results = matrix(NA, 5,5)

for(i in 1:5){
  for(j in 6:10){
    Results[i,(j-5)] = sum(bootstrap_samples2[, .SD, .SDcols = j]==i) / nrow(bootstraps2)
  }
}

colnames(Results) = paste0("Rank_", 1:5)
rownames(Results) = paste0("Asset_", 1:5)
Results
```

```
##           Rank_1   Rank_2   Rank_3   Rank_4   Rank_5
## Asset_1 0.447808 0.296088 0.180374 0.063969 0.011761
## Asset_2 0.276346 0.248454 0.242835 0.170901 0.061464
## Asset_3 0.195724 0.284314 0.281299 0.174721 0.063942
## Asset_4 0.037284 0.099070 0.203099 0.372655 0.287892
## Asset_5 0.042838 0.072074 0.092393 0.217754 0.574941
```

References:

- Vinod, Hrishikesh D. and Viole, Fred, *Arbitrary Spearman's Rank Correlations in Maximum Entropy Bootstrap and Improved Monte Carlo Simulations* (June 7, 2020). Available at SSRN: <http://dx.doi.org/10.2139/ssrn.3621614>
- Viole, F. and Nawrocki, D. (2016) LPM Density Functions for the Computation of the SD Efficient Set. *Journal of Mathematical Finance*, **6**, 105-126. doi: 10.4236/jmf.2016.61012.