Nonlinear Scaling Normalization with NNS

Install NNS

We need the latest version of NNS available on GitHub.

```
require(devtools); install_github('OVVO-Financial/NNS',ref = "NNS-Beta-Version")
require(NNS)
```

Nonlinear Scaling Normalization

NNS nonlinear scaling normalization involves using the degree of nonlinearity between variables to then transform the values to a shared range.

NNS's technique is different than linear scaling where all variables are assumed to have a perfect linear relationship (and by extension wind up with the same mean); and different than standardization ($\frac{(x-\mu)}{\sigma}$) which does not consider variable relationships.

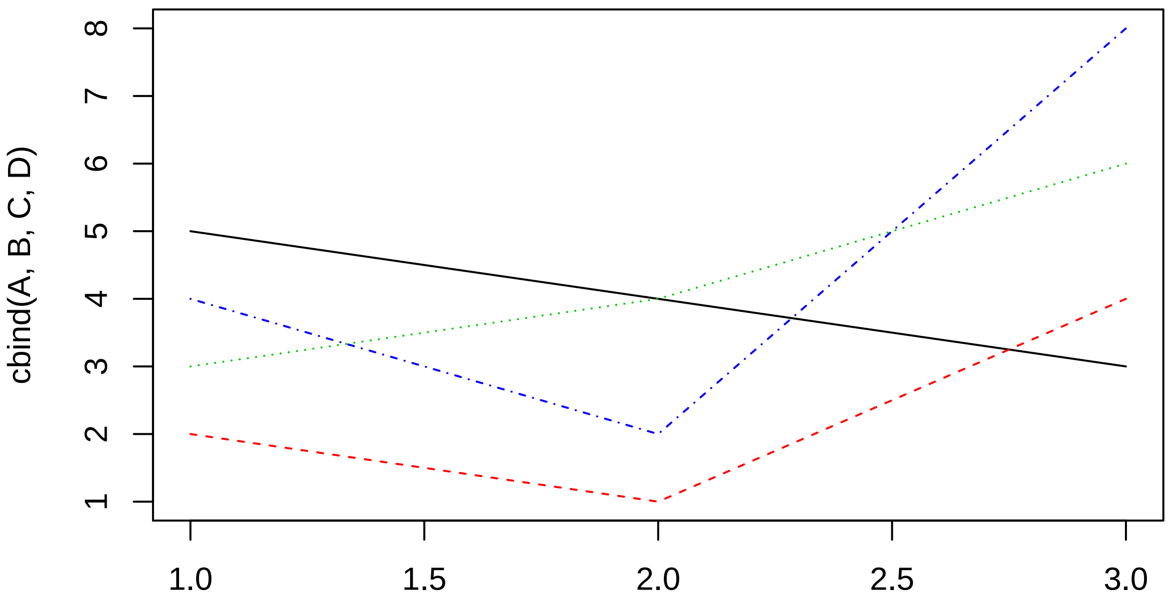
NNS is also different than quantile normalization, which as we'll show is not robust to different original scale values in multivariate problems.

Examples

A multivariate example of each technique will highlight the differences.

We will use the quintessential quantile normalization example on Wikipedia as well as a more intricate financial variable example.

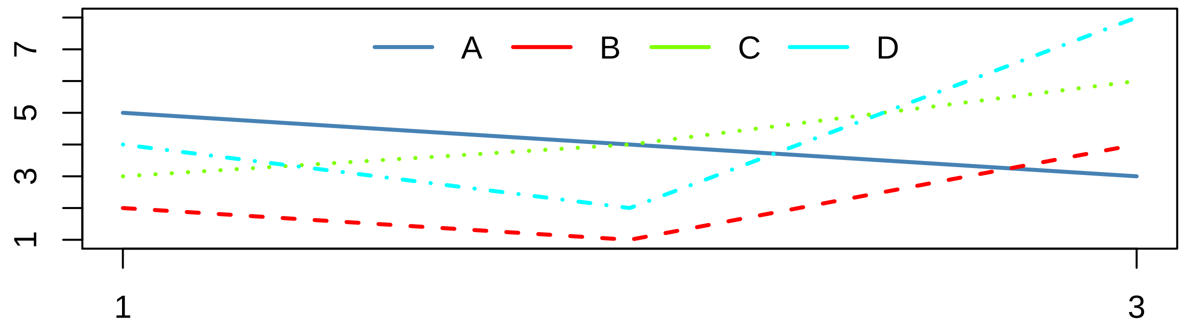
```
A=c(5,4,3);B=c(2,1,4);C=c(3,4,6);D=c(4,2,8)
matplot(cbind(A,B,C,D),type = 'l')
```

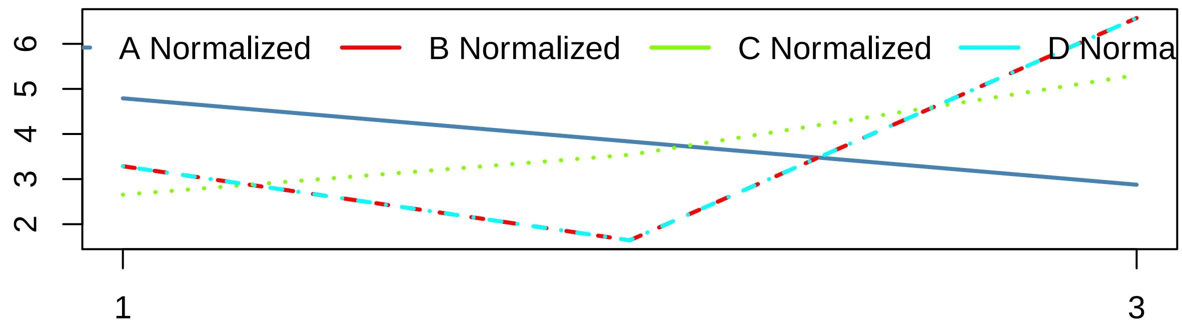

cbind(A, B, C, D)
1
2
3
4
5
6
7
8
1.0
1.5
2.0
2.5
3.0

Linear Scaling

NNS offers a linear scaling option which essentially sets the correlation between variables to 1.

```
lin.norm.variables=NNS.norm(cbind(A,B,C,D),chart.type='l',Linear=T)
```





A quick check verifies all of the means are equal for the linear scaling technique:

```
t(lin.norm.variables)
```

```
##       [,1]     [,2]     [,3]
## A 4.791667 3.833333 2.875000
## B 3.285714 1.642857 6.571429
## C 2.653846 3.538462 5.307692
## D 3.285714 1.642857 6.571429
```

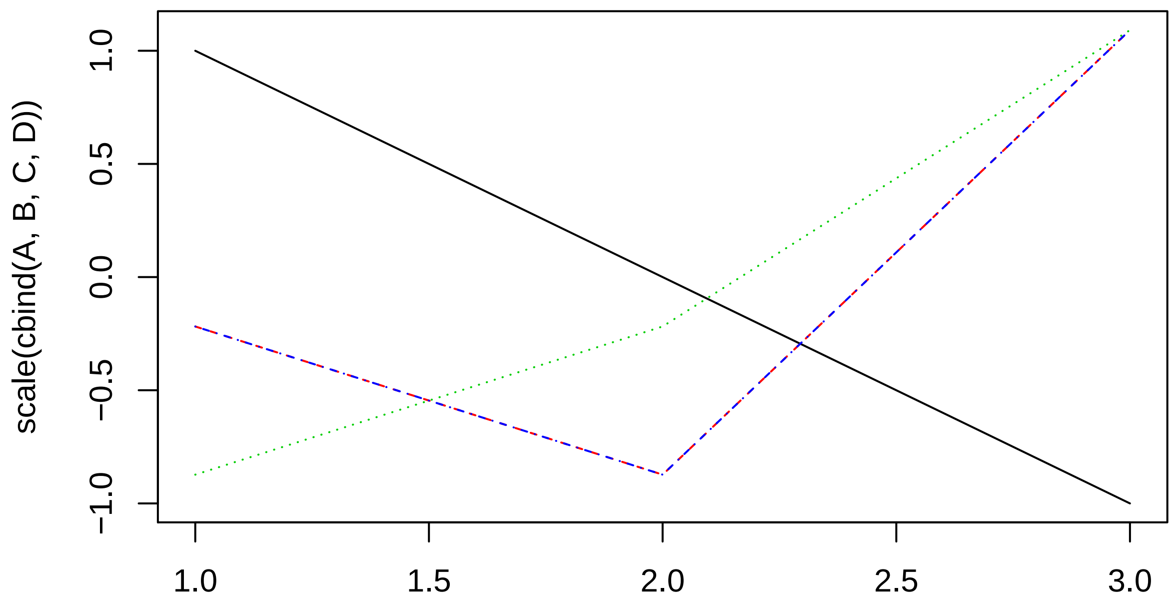
```
apply(lin.norm.variables,2,function(x) c(mean=mean(x),sd=sd(x)))
```

```
##              A        B        C        D
## mean 3.8333333 3.833333 3.833333 3.833333
## sd   0.9583333 2.509506 1.351272 2.509506
```

Stadard Standarization

Variables after going through a standard standardization technique will have a zero mean and unit variance. The following demonstrates the results.

```
matplot(scale(cbind(A,B,C,D)),type = 'l')
```



Means=0; variances=1…

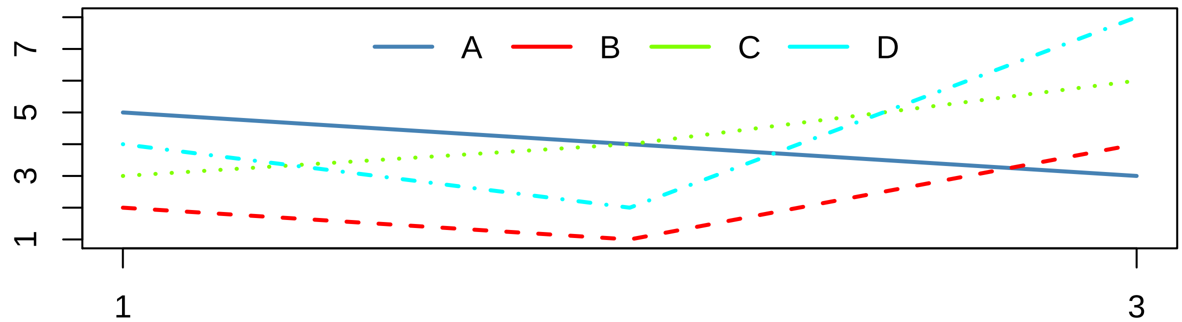
```
apply(scale(cbind(A,B,C,D)),2,function(x) c(mean=mean(x),sd=sd(x)))
```

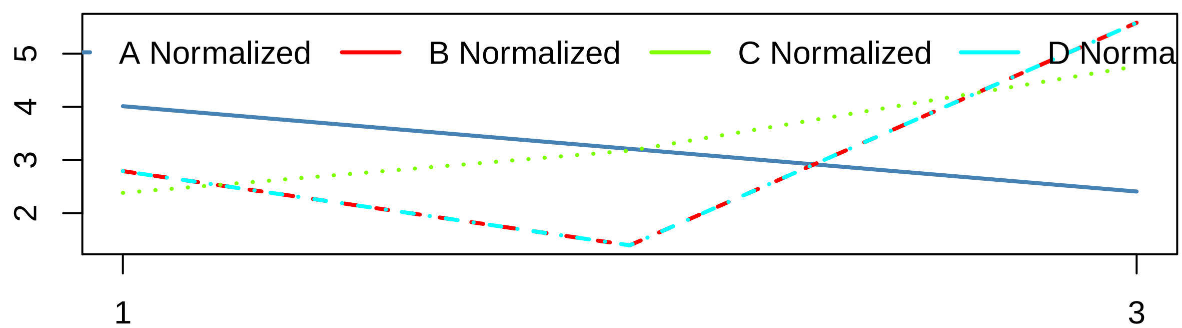
```
##      A             B            C             D
## mean 0 -7.401487e-17 2.035409e-16 -7.401487e-17
## sd   1  1.000000e+00 1.000000e+00  1.000000e+00
```

NNS Nonlinear Scaling

We can now compare the noninear scaling normaliztion technique.

```
nonlin.norm.variables=NNS.norm(cbind(A,B,C,D),chart.type='l',Linear=F)
```





And we note the difference in means along with the smaller standard deviation for each transformed variable versus the other methods.

```
t(nonlin.norm.variables)
```

```
##       [,1]     [,2]     [,3]
## A 4.011820 3.209456 2.407092
## B 2.790724 1.395362 5.581447
## C 2.381756 3.175674 4.763511
## D 2.790724 1.395362 5.581447
```

```
apply(nonlin.norm.variables,2,function(x) c(mean=mean(x),sd=sd(x)))
```

```
##             A        B        C        D
## mean 3.209456 3.255844 3.440314 3.255844
## sd   0.802364 2.131450 1.212731 2.131450
```

Quantile Normalization

Quantile normalization is a technique for making two or more distributions identical in statistical properties. This is the antithesis of `NNS.norm` as we seek to retain the uniqueness of each distribution while creating a shared range of observations.

```
#install if necessary
#install.packages("BiocInstaller",
#  repos="http://bioconductor.org/packages/3.4/bioc")

require(BiocInstaller)
```

```
## Loading required package: BiocInstaller
```

```
## Bioconductor version 3.4 (BiocInstaller 1.24.0), ?biocLite for help
```

```
#if necessary...
#biocLite('preprocessCore')

require(preprocessCore)
```

```
## Loading required package: preprocessCore
```

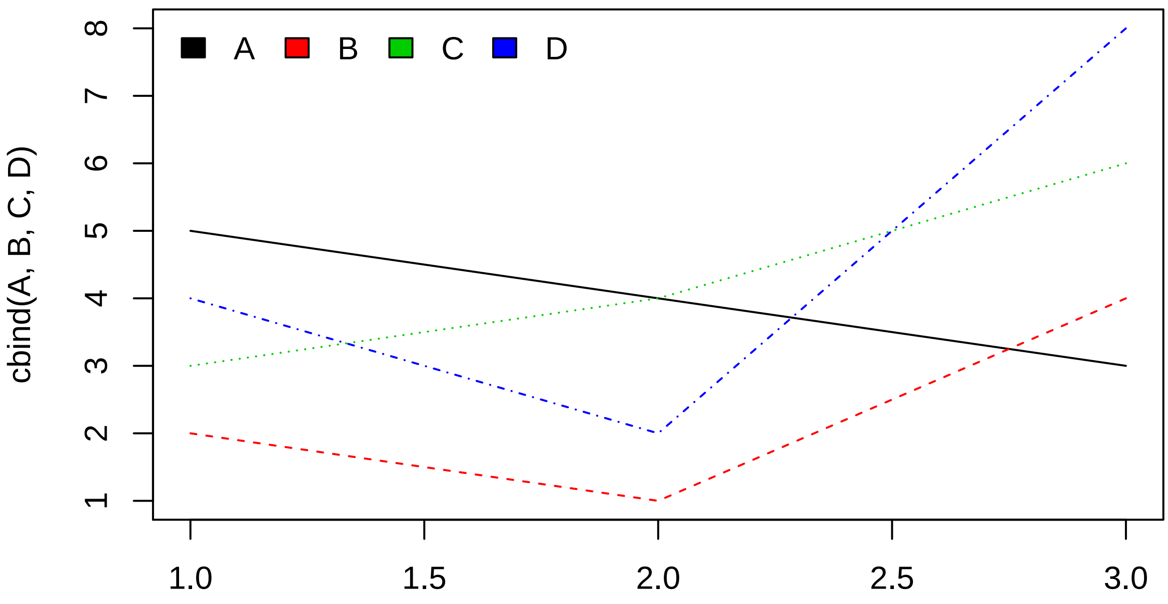
```
#the function expects a matrix
#create a matrix using the same example
mat <- matrix(c(5,2,3,4,4,1,4,2,3,4,6,8),
#mat <- matrix(c(20,1,5,13,19,2,6,12,17,3,8,10),
             ncol=3)
row.names(mat)=c("A","B","C","D")
mat
```

```
##   [,1] [,2] [,3]
## A    5    4    3
## B    2    1    4
## C    3    4    6
## D    4    2    8
```

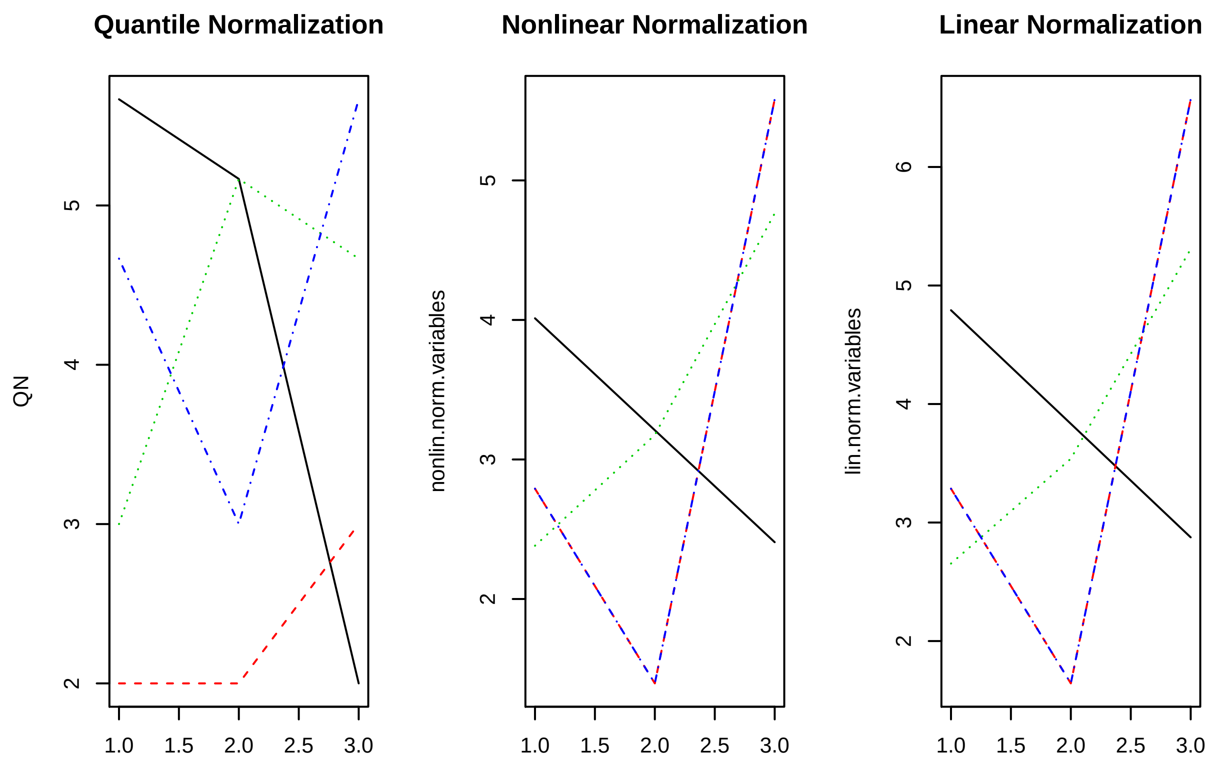
```
qn.mat=apply(normalize.quantiles(mat),1,function(x) c(mean=mean(x),sd=sd(x)))
colnames(qn.mat)=c("A","B","C","D")
QN=normalize.quantiles(mat)
```

Comparing all methods with the original dataset yields:

```
### Rotate matrix for plotting as QN is row based variables
QN=t(QN[1:nrow(QN),])

#Plot
matplot(cbind(A,B,C,D),type = 'l')
legend('topleft',legend=c("A","B","C","D"),col = seq_len(4),fill=seq_len(4),horiz = T,bty='n')
```




```
par(mfrow=c(1,3))
matplot(QN,type = 'l')
title(main = "Quantile Normalization")
matplot(nonlin.norm.variables,type = 'l')
title(main = "Nonlinear Normalization")
matplot(lin.norm.variables,type = 'l')
title(main = "Linear Normalization")
```


Quantile Normalization
Nonlinear Normalization
Linear Normalization
QN
nonlin.norm.variables
lin.norm.variables
1.0
1.5
2.0
2.5
3.0
2
3
4
5
6

Financial Series Example

Let's load some financial variables that have overlapping prices...

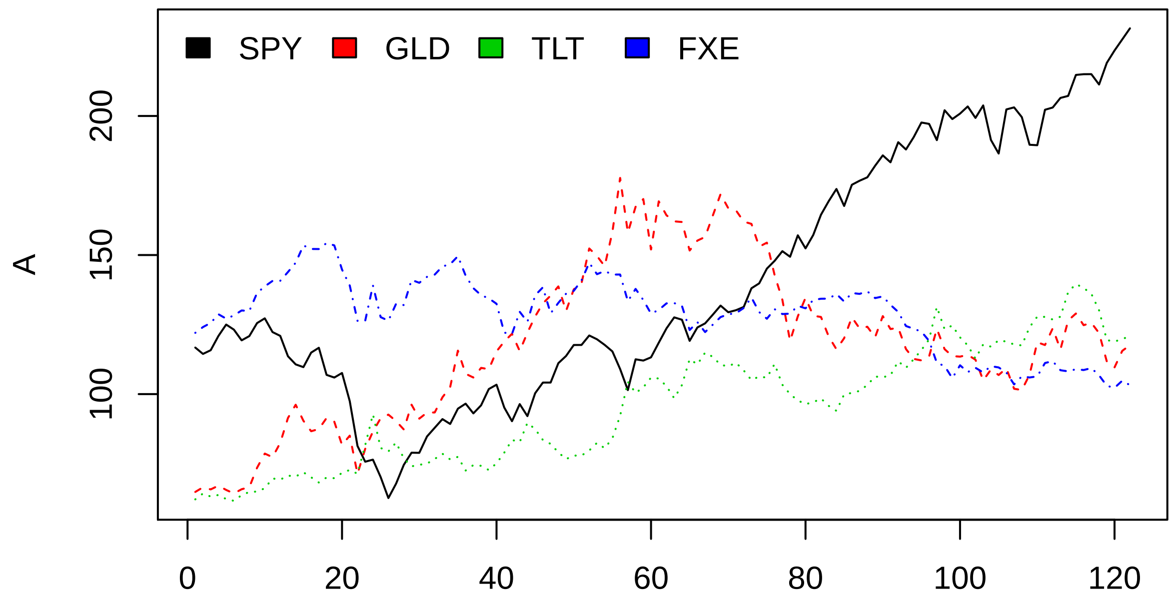
```
require(quantmod)
getSymbols(c("SPY","GLD","TLT","FXE"),src='yahoo')
```

```
## [1] "SPY" "GLD" "TLT" "FXE"
```

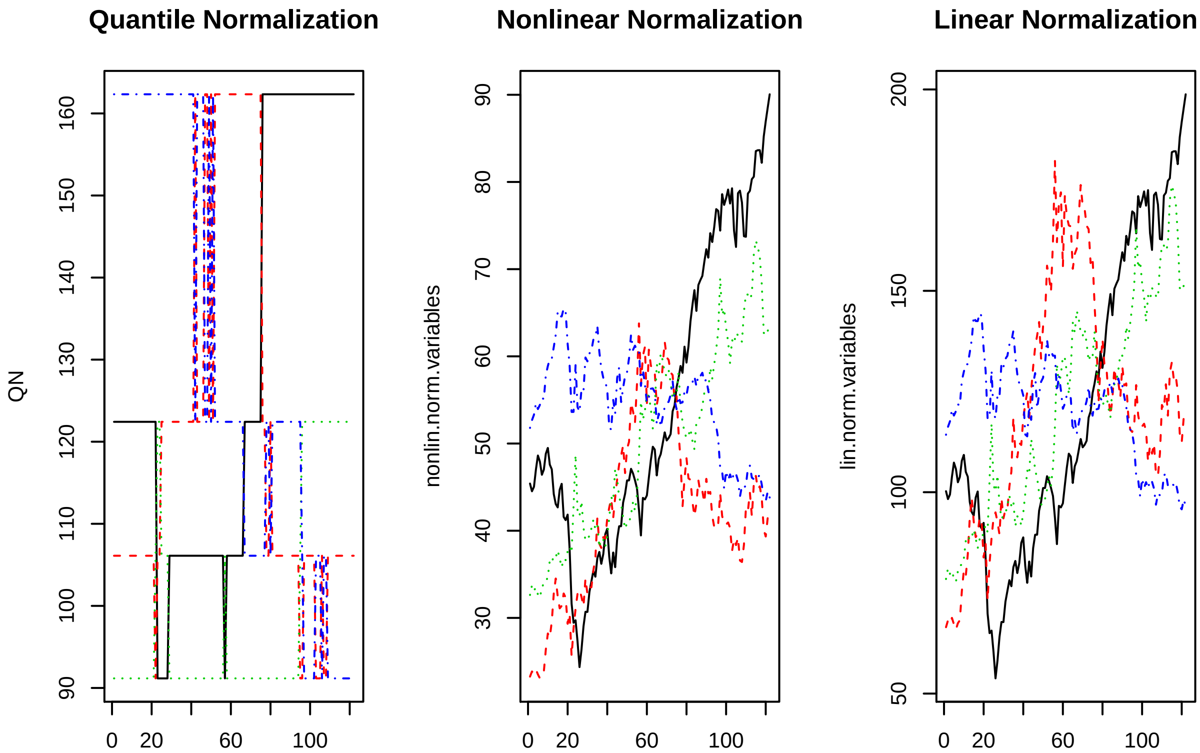
```
SPY<- Ad(to.monthly(SPY))
GLD<- Ad(to.monthly(GLD))
TLT<- Ad(to.monthly(TLT))
FXE<- Ad(to.monthly(FXE))

A<- as.matrix(cbind(SPY,GLD,TLT,FXE))
A<- A[complete.cases(A),]

# Rotate matrix to variables by row for QN
qn.mat=t(A[1:nrow(A),])
QN=normalize.quantiles(qn.mat)
nonlin.norm.variables=NNS.norm(A,Linear = F)
lin.norm.variables=NNS.norm(A,Linear = T)
```

Comparing all methods with the original dataset yields:

```
#Rotate QN results so matplot can use columns
QN=t(QN[1:nrow(QN),])

#Plot
matplot(A,type = 'l')
legend('topleft',legend=c("SPY","GLD","TLT","FXE"),col = seq_len(ncol(A)),fill=seq_len(ncol(A)),horiz =
```




```r
par(mfrow=c(1,3))
matplot(QN,type = 'l')
title(main = "Quantile Normalization")
matplot(nonlin.norm.variables,type = 'l')
title(main = "Nonlinear Normalization")
matplot(lin.norm.variables,type = 'l')
title(main = "Linear Normalization")
```



Note the tighter overall range for `NNS.norm`.

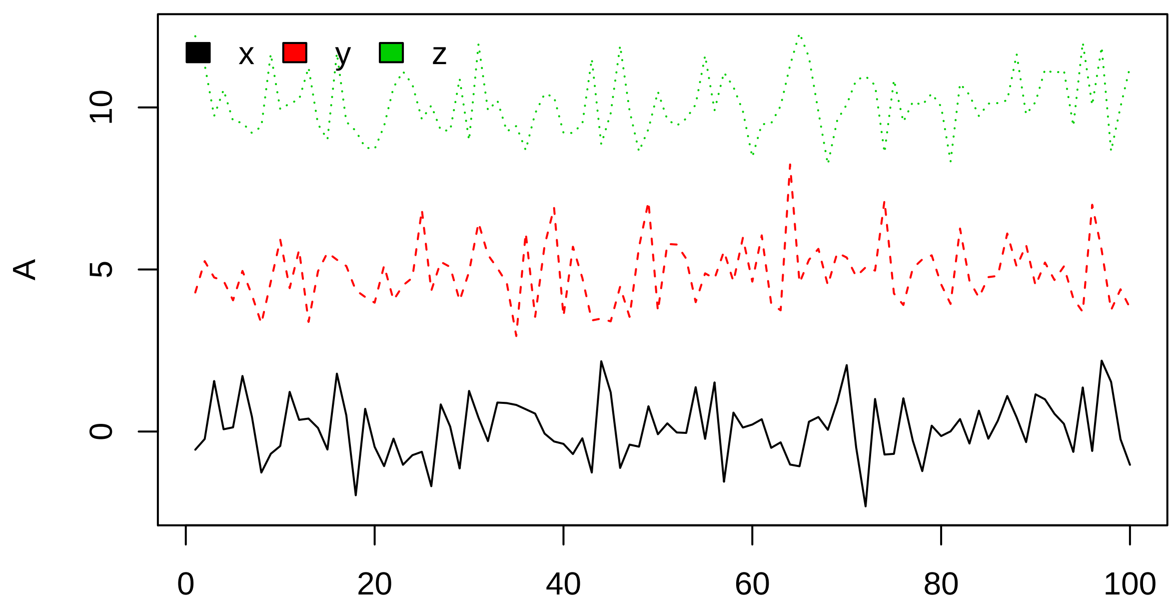
Non-overlapping variables

When the variables do not share any ranges of observations, quantile normalization merely reports the mean value for that variable.

```
set.seed(123)
x=rnorm(100);y=rnorm(100)+5;z=rnorm(100)+10
A=as.matrix(cbind(x,y,z))
nonlin.norm.variables=NNS.norm(A,Linear = F)
lin.norm.variables=NNS.norm(A,Linear = T)

# Rotate matrix to variables by row for QN
qn.mat=t(A[1:nrow(A),])
QN=normalize.quantiles(qn.mat)
```

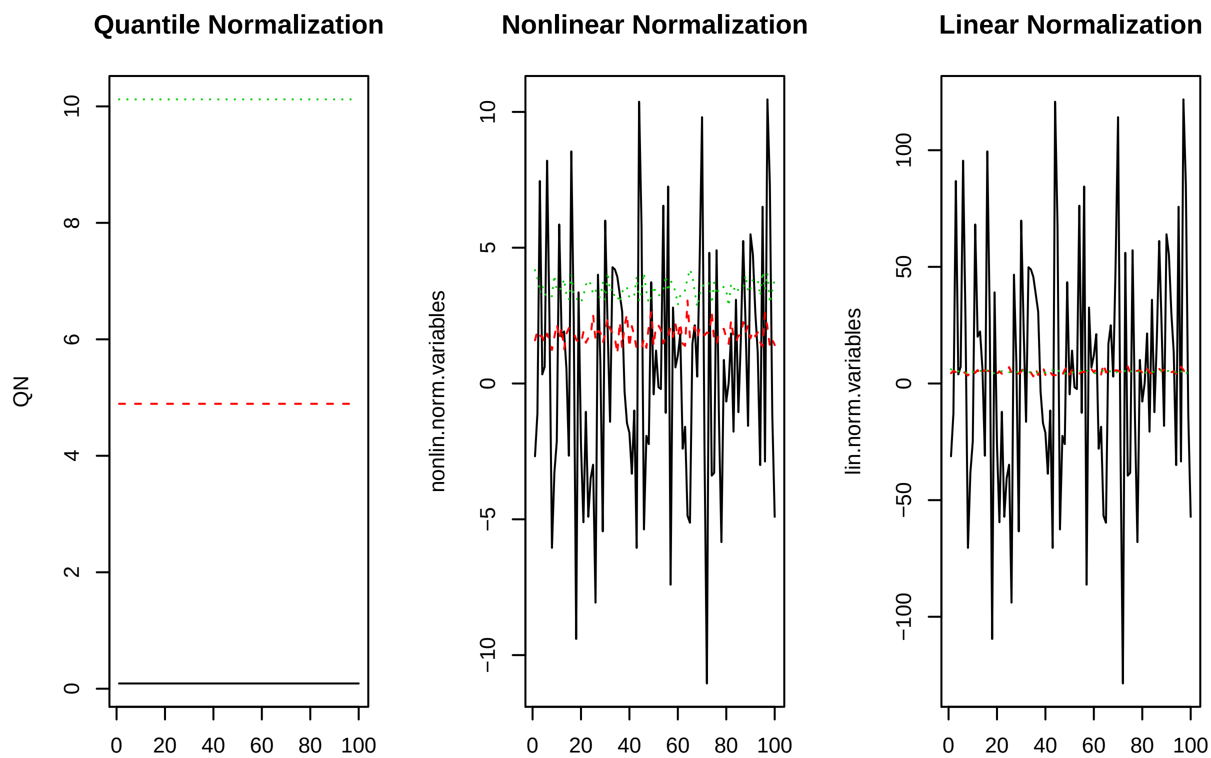
Quantiles have a problem…

```
#Rotate QN results so matplot can use columns
QN=t(QN[1:nrow(QN),])

#Plot
matplot(A,type = 'l')
legend('topleft',legend=colnames(A),col = seq_len(ncol(A)),fill=seq_len(ncol(A)),horiz = T,bty='n')
```




```
par(mfrow=c(1,3))
matplot(QN,type = 'l')
title(main = "Quantile Normalization")
matplot(nonlin.norm.variables,type = 'l')
```

```
title(main = "Nonlinear Normalization")
matplot(lin.norm.variables,type = 'l')
title(main = "Linear Normalization")
```



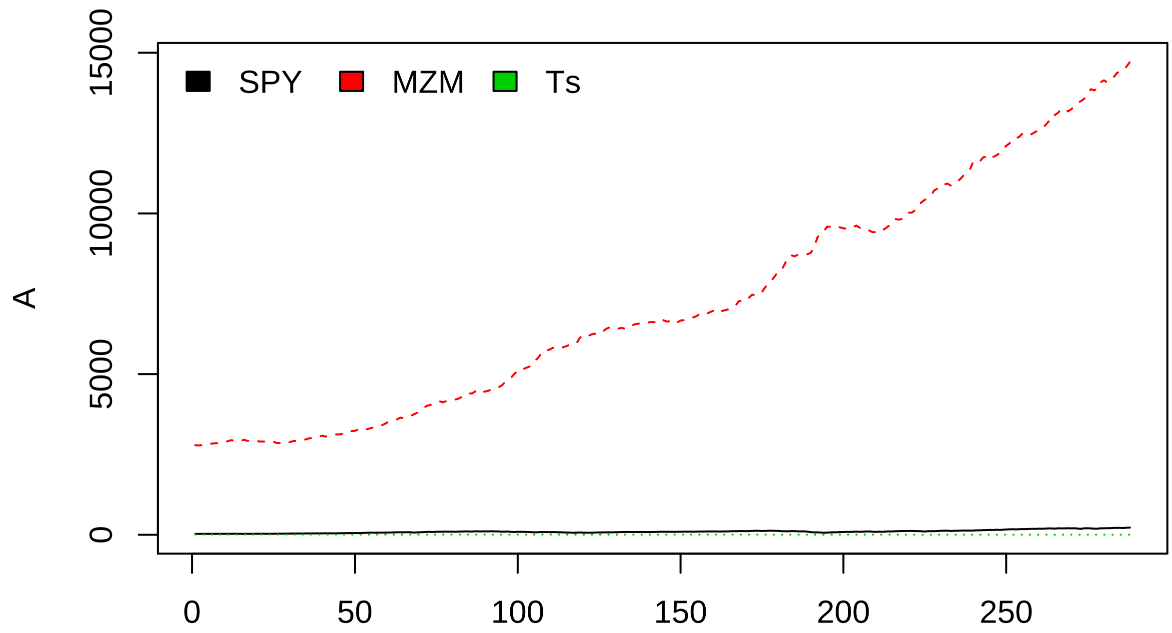
Orders of Magnitude Reduced

We can successfully remove orders of magnitude differences between variables. Here is a financial example using money supply (MZM) measured in billions of dollars, the S&P 500 (SPY) closing prices measured in points and US 10-year Treasury yields (Ts) measured in percentage points.

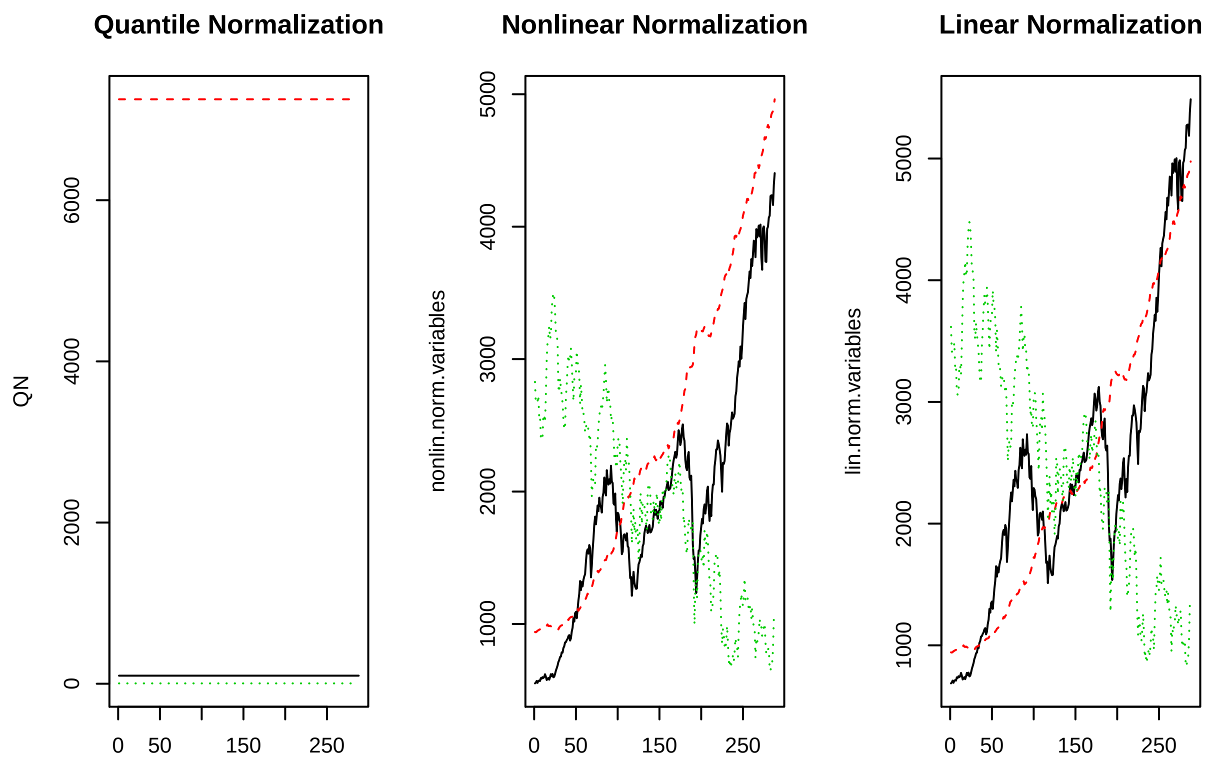
```r
getSymbols("SPY",src='yahoo',from="1965-01-01")
getSymbols(c("MZMNS","DGS10"),src='FRED',from="1965-01-01")

SPY<- Ad(to.monthly(SPY['1962::']))
MZM<- MZMNS['1962::']
Ts<- Cl(to.monthly(DGS10['1962::']))
```

```
## Warning in to.period(x, "months", indexAt = indexAt, name = name, ...):
## missing values removed from data
```

```r
A=as.matrix(cbind(SPY,MZM,Ts))
A<- A[complete.cases(A),]
# Rotate matrix to variables by row for QN
qn.mat=t(A[1:nrow(A),])
nonlin.norm.variables=NNS.norm(A,Linear = F)
lin.norm.variables=NNS.norm(A,Linear = T)

QN=normalize.quantiles(qn.mat)

#Rotate QN results so matplot can use columns
QN=t(QN[1:nrow(QN),])

#Plot
matplot(A,type = 'l')
legend('topleft',legend=c("SPY","MZM","Ts"),col = seq_len(ncol(A)),fill=seq_len(ncol(A)),horiz = T,bty=
```




```r
par(mfrow=c(1,3))
matplot(QN,type = 'l')
title(main = "Quantile Normalization")
matplot(nonlin.norm.variables,type = 'l')
title(main = "Nonlinear Normalization")
matplot(lin.norm.variables,type = 'l')
title(main = "Linear Normalization")
```



Practical Applications

Normalization eliminates the need for multiple y-axis charts and prohibits the misuse thereof. Furthermore, placing variables on the same axes with shared ranges permits a more relevant conditional probability analysis, which along with time normalization is used in the `NNS.caus` routine for identifying causal relationships between variables.

To learn more about NNS statistics and their theoretical foundations, see "*Nonlinear Nonparametric Statistics: Using Partial Moments*" available on Amazon: http://a.co/5bpHvUg

Check back to see more NNS examples posted on GitHub:

https://github.com/OVVO-Financial/NNS/tree/NNS-Beta-Version/examples