Recursive depth to compute the geothmetic meandian (from xkcd) does not depend on the variance, skewness, or kurtosis of the input

William John Holden

13 March 2021

Geothmetic Meandian

xkcd 2435 presents a novel and interesting "Geothmetic Meandian," a statistic for a vector derived recursively from arithmetic mean, geometric mean, and median.

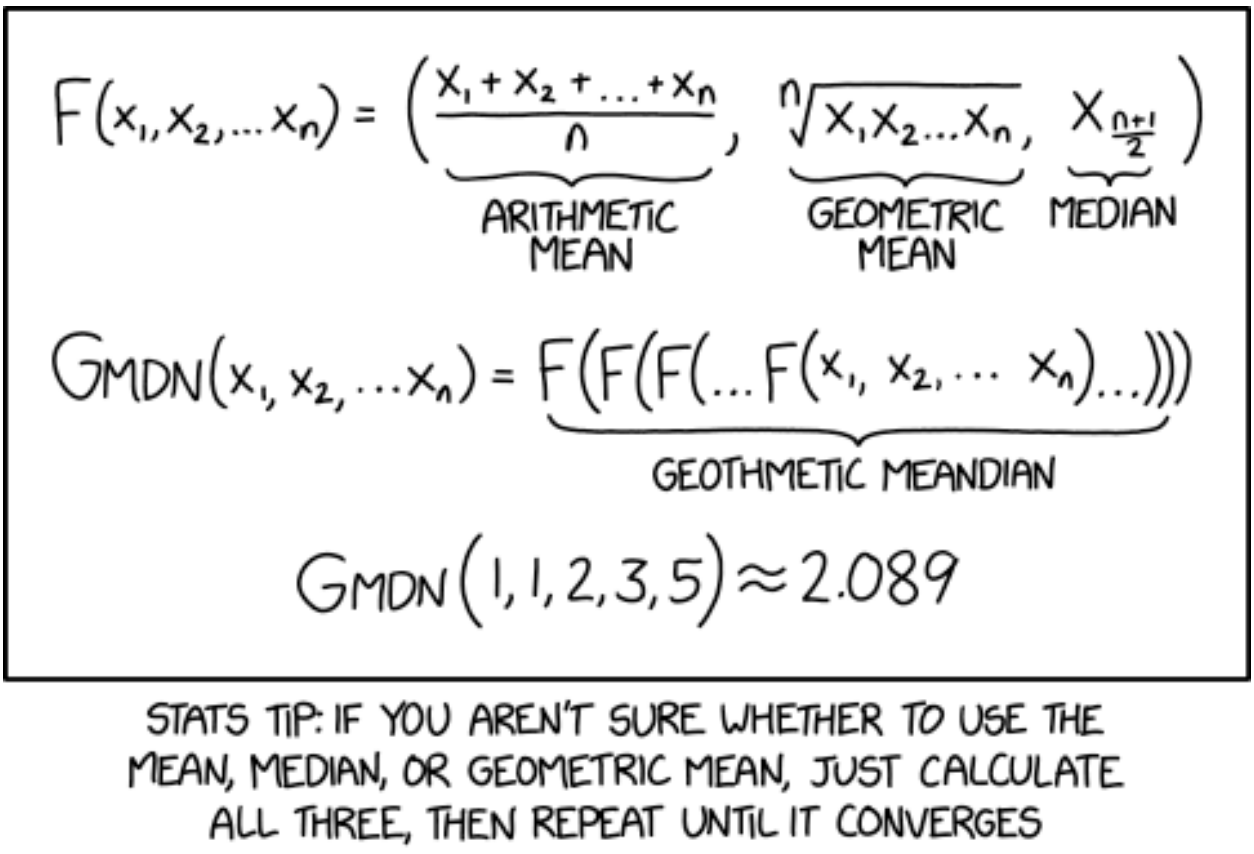


Figure 1: xkcd 2435

This function is really interesting! Here is an implementation in Julia.

```
using Statistics, StatsBase;
gmdn(x) = all(y->isapprox(y, x[1]), x) ? x[1] : gmdn([mean(x), geomean(x), median(x)]);
```

The `isapprox` comparison operator (≈) is used instead of `isequal` (==) to correct for precision errors.

Otherwise, the `gmdn` function frequently fails to converge due to a fixed point where the arithmetic mean, geometric mean, and median are all equal.

Here is the same geothmetic meandian computed in the comic:

```
gmdn([1,1,2,3,5])
```

```
## 2.0890579551529527
```

Reddit user u/firefly431 has proven that the `gmdn` function always converges.

How many recursive calls to `gmdn` are needed before the arithmetic mean, geometric mean, and median converge? My intuition is that the number of recursive calls will increase with at least one of the second, third, and forth moments (variance, skewness, and kurtosis).

Recursive Depth of the `gmdn` function

Let us modify the `gmdn` function to return the recursive depth instead of the geothmetic meandian.

```
gmdn_depth(x) = all(y->isapprox(y, x[1]), x) ? 1 :
  1 + gmdn_depth([mean(x), geomean(x), median(x)]);
```

The base case is 1:

```
gmdn_depth(ones(100))
```

```
## 1
```

The number of recursive calls to compute `gmdn` for the example input is:

```
gmdn_depth([1,1,2,3,5])
```

```
## 17
```

Let us analyze some larger random samples.

```
# X_uniform is a 30 row by 10 column matrix of uniformly distributed random variables.
X_uniform = 100 .* rand(30, 10);
mean(X_uniform)
```

```
## 49.77483515189965
```

```
std(X_uniform)
```

```
## 30.589419130818538
```

```
[gmdn_depth(column) for column in eachcol(X_uniform)]
```

```
## 10-element Vector{Int64}:
##  17
##  16
##  16
##  17
##  17
##  17
##  17
##  17
##  18
##  18
```

The values of `X_uniform` are uniformly distributed between 0 and 100. How about a normal distribution?

```
# X_normal is a 30 row by 10 column matrix of normally distributed random variables.
X_normal = 50 .+ 10 .* randn(30, 10);
mean(X_normal)
```

```
## 50.67685769204344
```

```
std(X_normal)
```

```
## 10.047660701783252
```

```
[gmdn_depth(column) for column in eachcol(X_normal)]
```

```
## 10-element Vector{Int64}:
##  15
##  15
##  14
##  15
##  15
##  13
##  15
##  15
##  14
##  15
```

It looks like the value of `gmdn_depth` could correlate to the standard deviation, σ (`std` in Julia).

Randomly distributed inputs

Let us use the Sinh-archsinh distribution to generate random distributions. The parameter ϵ influences the skew and the parameter δ influences the kurtosis of the output distribution.

$$H\left(X, \epsilon, \delta\right) = \sinh\left[\delta \sinh^{-1}\left(x\right) - \epsilon\right]$$

```
H(X, epsilon, delta) = sinh.(delta .* asinh.(X) .- epsilon);
```

We need some input vectors with variable moments. This is trickier than one might expect. Random number generators are generally constructed to generate uniform or normal distributions. The idea here is to:

1. Generate a matrix X of normally distributed random variables with $\mu = 100$ and $\sigma = 10$.
2. Generate parameter pairs (ϵ, δ) from the Cartesian product $\{-2, -1, 0, 1, 2\} \times \{.5, .75, 1, 1.25, 1.5\}$.
3. Compute the Sinh-arcsinh transformation $X' = H\left(X, \epsilon, \delta\right)$ over each column in X against parameter pairs (ϵ, δ), cycling parameter pairs as needed.
4. Compute the standard deviation, skewness, kurtosis, and `gmdn_depth` for each transformed column.
5. Combine all four statistics into a data frame.

In Julia:

```
X = 100 .+ 10 .* randn(30, 100);
parameters = Iterators.product([-2, -1, 0, 1, 2], [.5, .75, 1, 1.25, 1.5]);
X_ = [H(x, e, d) for (x, (e, d)) in zip(eachcol(X), Iterators.cycle(parameters))];
using DataFrames;
df = DataFrame(StdDev = std.(X_),
               Skewness = skewness.(X_),
               Kurtosis = kurtosis.(X_),
               GMDN_Depth = gmdn_depth.(X_));
first(df, 10)
```

```
## 10×4 DataFrame
##  Row   StdDev      Skewness    Kurtosis    GMDN_Depth
##        Float64     Float64     Float64     Int64
##
##    1    2.50559    -0.110843   -1.24655           12
##    2    0.78402    -0.16827     0.724668          10
##    3    0.377406   -0.244188   -0.904867          15
##    4    0.135034   -0.327379    0.354314          13
##    5    0.0578967  -0.396633    0.208562          13
##    6   11.4835      0.232954   -0.621725          15
##    7    6.0399      0.35033     0.0705089         12
##    8    2.40153     1.02078     1.75015           14
##    9    0.654665   -0.0412755  -0.551803          12
##   10    0.313837    0.199577   -1.05918           13
```

Before we dive into plots, observe that the standard deviation for columns in X' can be *huge*.

```
describe(df)
```

```
## 4×7 DataFrame
##  Row   variable    mean         min        median     max      nmissing  eltyp
##        Symbol      Float64      Real       Float64    Real     Int64     DataT
##
##    1   StdDev      122.097         0.0526   8.8562    1662.77         0  Float
##    2   Skewness     -0.0100062  -0.977729  -0.021468  1.02078         0  Float
##    3   Kurtosis     -0.164102    -1.31233  -0.287925  2.21828         0  Float
##    4   GMDN_Depth   13.62              10  14.0             15         0  Int64
##                                                                 1 column omitted
```

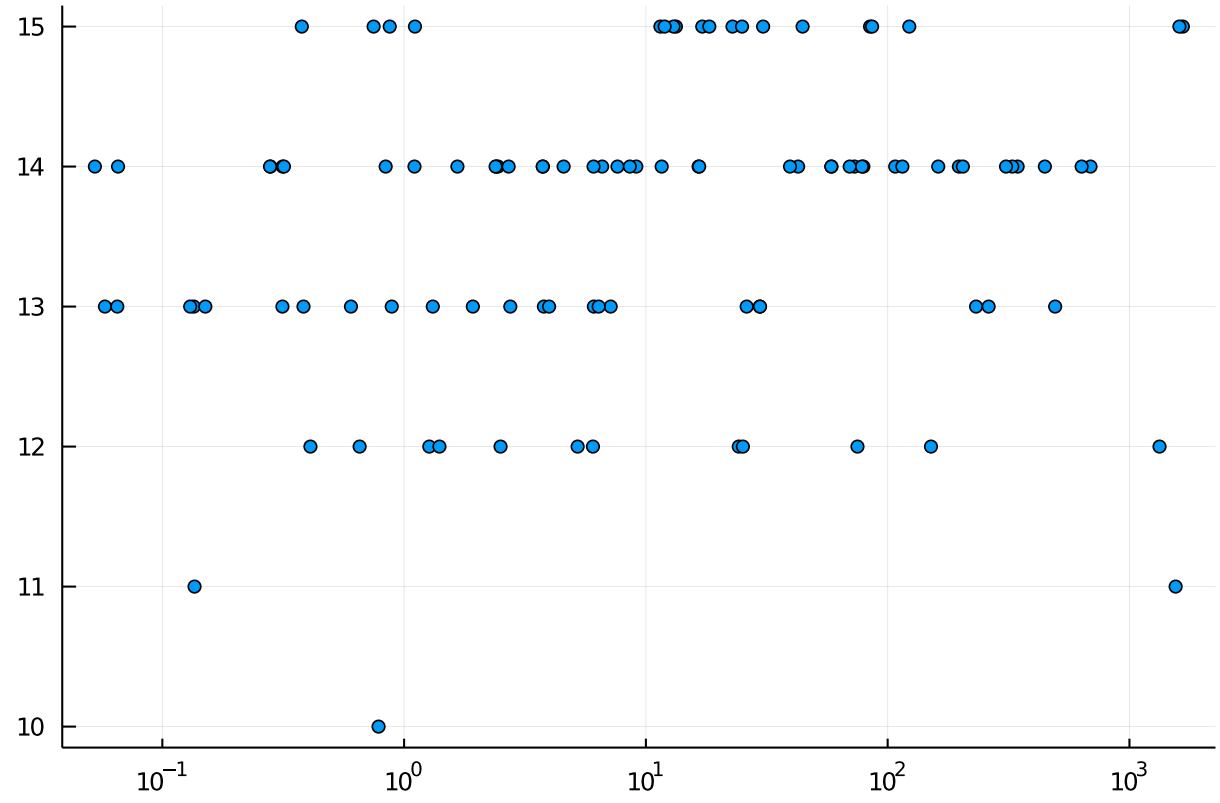
Plots of `gmdn` recusive depth to input moments

Now for the plots!

```
using Plots;
gr();
```

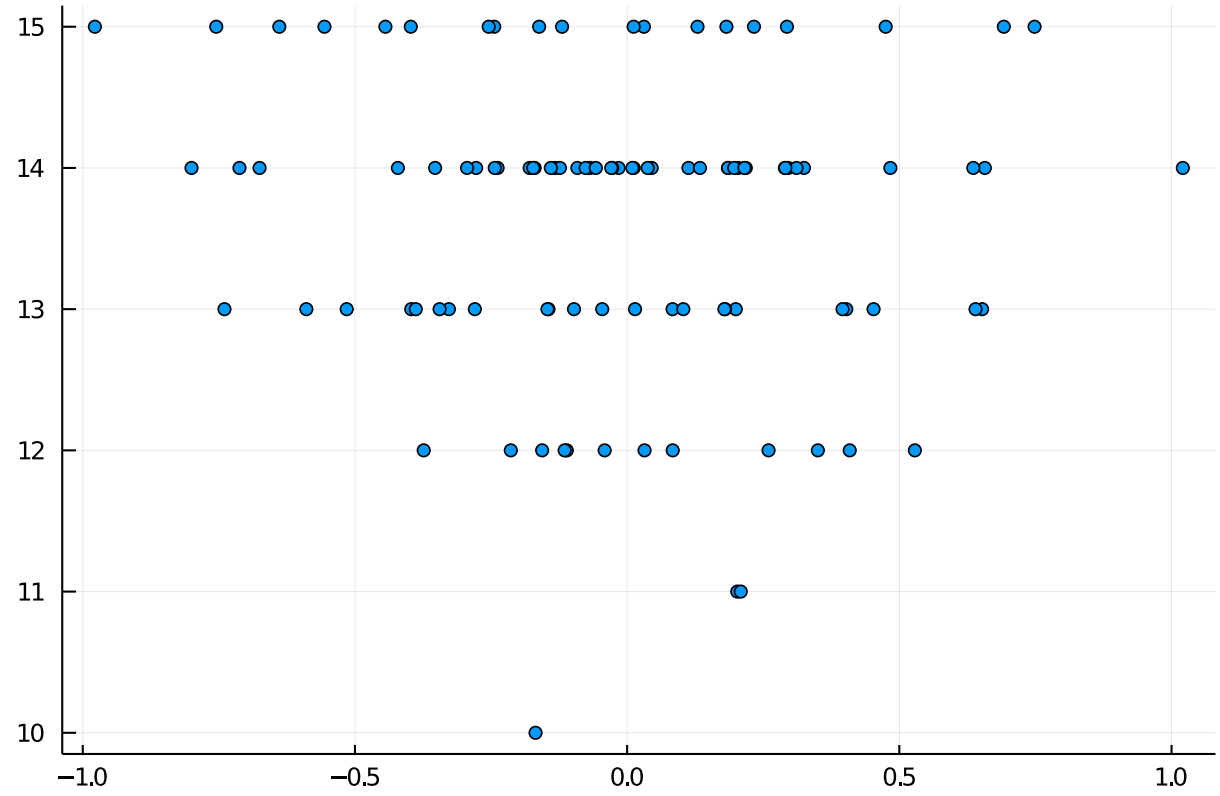
We begin with standard deviation.

```
# show independent variable on a logarithmic scale
scatter(df.StdDev, df.GMDN_Depth, xaxis=:log, legend = false)
```



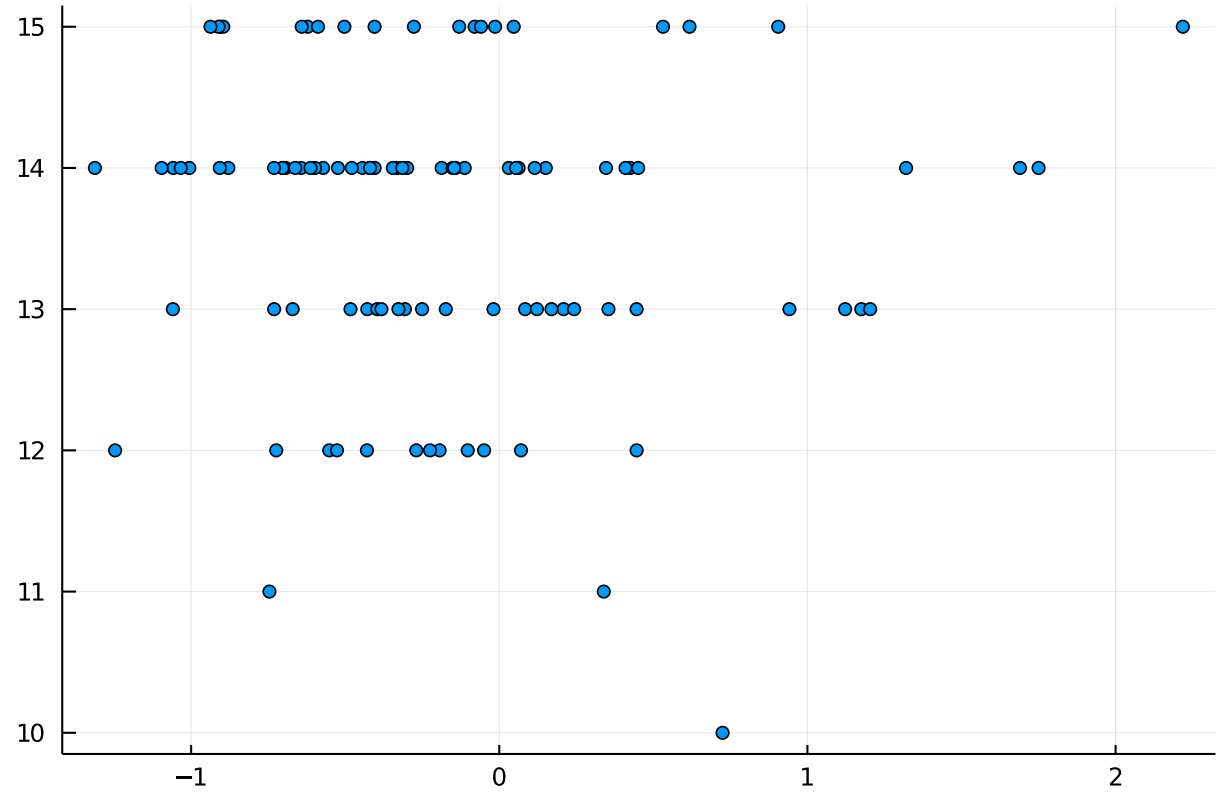
It does not appear that `gmdn_depth` corresponds to the standard deviation of the input vector. Next, skewness.

```
scatter(df.Skewness, df.GMDN_Depth, legend = false)
```



It does not appear that `gmdn_depth` corresponds to the skewness of the input vector.

```
scatter(df.Kurtosis, df.GMDN_Depth, legend = false)
```



It does not appear that `gmdn_depth` corresponds to the kurtosis of the input vector.

Linear regressions and ANOVA tests

The plots certainly do not suggest any linear relationship. Using the `lm` function we can find R^2 to assess the strength of the line of best fit.

```
using GLM;
```

Standard deviation:

```
r2(lm(@formula(GMDN_Depth ~ StdDev), df))
```

```
## 0.0003282765218863837
```

A R^2 value of near 1 would indicate that the line of best fit is a very accurate model. This R^2 value is near 0, which shows that the line of best fit is a very *poor* model. The same is true of skewness:

```
r2(lm(@formula(GMDN_Depth ~ Skewness), df))
```

```
## 0.007189800569794236
```

And kurtosis:

```
r2(lm(@formula(GMDN_Depth ~ Kurtosis), df))
```

```
## 0.003037900163180174
```

Finally, we can attempt to fit all three at once, but this model also fits the data very poorly:

```
r2(lm(@formula(GMDN_Depth ~ StdDev + Skewness + Kurtosis), df))
```

```
## 0.010306584950319664
```

A more general test for the responsiveness of a dependent variable to an independent variable is the Analysis of Variance (ANOVA). Julia does not have an `aov` function that is as easy to use as R. We can copy the datframe from Julia into R for analysis.

```
library(JuliaCall)
df <- julia_eval("df")
model <- aov(df$GMDN_Depth ~ df$StdDev + df$Skewness + df$Kurtosis)
model
```

```
## Call:
##    aov(formula = df$GMDN_Depth ~ df$StdDev + df$Skewness + df$Kurtosis)
##
## Terms:
##                 df$StdDev df$Skewness df$Kurtosis Residuals
## Sum of Squares    0.03597     0.76210     0.33112 108.43081
## Deg. of Freedom         1           1           1        96
##
## Residual standard error: 1.062774
## Estimated effects may be unbalanced
```

```
summary(model)
```

```
##             Df Sum Sq Mean Sq F value Pr(>F)
## df$StdDev    1   0.04  0.0360   0.032  0.859
## df$Skewness  1   0.76  0.7621   0.675  0.413
## df$Kurtosis  1   0.33  0.3311   0.293  0.589
## Residuals   96 108.43  1.1295
```

Each of the probabilities shown in the rightmost column are larger than 0.10, and in traditional hypothesis testing we would fail to reject a null hypothesis with $\alpha = 0.05$.

Conclusion

My hunch was wrong. The recursive depth to compute the `gmdn` does not depend on the variance, skewness, or kurtosis of the input.