Day2 - Generating non-uniform random numbers

Michel Bje Randahl Nielsen (s093481)

July 1, 2015

1 About the implementations...

All implementations are done in F# scripts but uses functionality offered by the R programming environment. The scripts utilizes the type provider functionality which F# provides, to invoke functions in R.

F# project page: http://fsharp.org/

F# RProvider project page: https://bluemountaincapital.github.io/FSharpRProvider/

All scripts has been devloped inside visual studio with F# 3.1, but is only dependend on the existance of .Net 4.5 and a F# 3.1 instalation to be run. A full project has been attached to these hand ins and contains all of the scripts and nescesary dll's.

Note that all scripts must run following code in the beginning in order to function, but this piece of code has been omitted in these reports.

```
#r @"..\packages\R.NET.Community.1.5.16\lib\net40\RDotNet.dll"
#r @"..\packages\R.NET.Community.FSharp.0.1.9\lib\net40\RDotNet.FSharp.dll"
#r @"..\packages\RProvider.1.1.8\lib\net40\RProvider.dll"
#r @"..\packages\RProvider.1.1.8\lib\net40\RProvider.Runtime.dll"

open RDotNet
open RProvider
open RProvider.stats
open RProvider.graphics
open RProvider.mvtnorm
open RProvider.MASS
open System
```

2 Exercises I

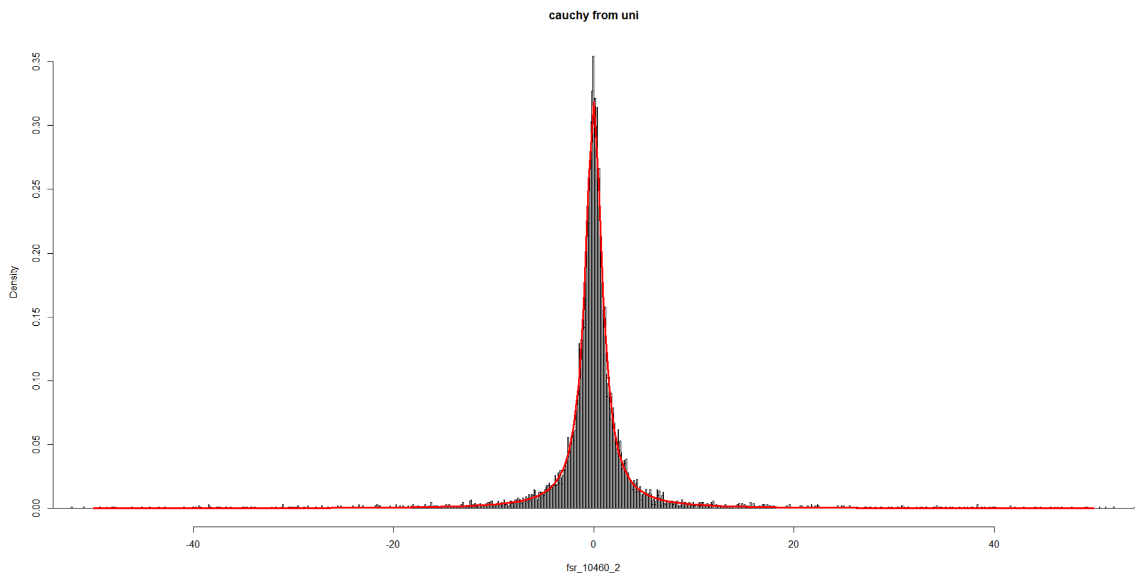
2.1 Simulate a sample from a Cauchy(0,1) distribution with the inversion method

Sampling from uniform dist, calculating inverse cauchy dist function and creating a data set using the resulting function and the uniformly sampled random numbers

```
(*
calculating the inverse function of cauchy(0,1)
u = 0.5 + Math.Atan(x) / Math.PI
u - 0.5 = Math.Atan(x) / Math.PI
(u - 0.5) * Math.PI = Math.Atan(x)
Math.Tan( (u - 0.5) * Math.PI) = x
*)

let Nsim = 10000
let breaks = 25000
let U = R.runif(Nsim).AsNumeric()

//the function for inverse cauchy
let cauchy_inv u = (u - 0.5) * Math.PI
                   |> Math.Tan
let X = Seq.map (fun u -> cauchy_inv u) U
```

Ploting a histogram of the sample and the cauchy dist fun

```
namedParams [
    "freq",box false
    "main",box "cauchy from uni"
    "x",box X
    "breaks", box breaks
    "xlim", box [-50; 50]
]
|> R.hist

let cauchy_fun x = 1.0 / (Math.PI * (1.0 + x ** 2.0))
let xs = [-50.0 .. 0.5 .. 50.0]
namedParams [
    "x", box xs
    "y",box <| Seq.map cauchy_fun xs
    "col", box "red"
    "lwd", box 3
]
|> R.lines
```



The histogram shows that the generated sample follows the cauchy distribution very well.

Performing statistical check on the sample distribution to see if it follows the cauchy dist.

```
namedParams [
    "x", box X
    "y", box "pcauchy"
    "location",box 0
    "scale",box 1
]
|> R.ks_test
|> fun r -> r.Print()
```

Outputs...

```
val it : string =
  "
    One-sample Kolmogorov-Smirnov test

data:  fsr_10460_13
D = 0.011233, p-value = 0.1603
alternative hypothesis: two-sided
"
```

we get a p-value higher than the significant interval and thus reject the null hypothesis that the generated numbers doesn't follow a cauchy dist.

2.2 Simulate a sample from a Beta(2,5) distribution using the rejection method with a uniform distribution as auxiliary distribution

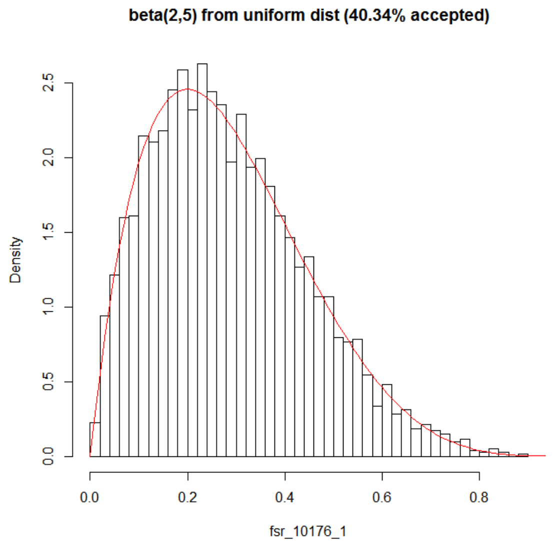
Sampling two sets of uniformly distributed data and defining the function that performs the rejection process

```
let trials = 10000
//list of random numbers to take from
let U1 = R.runif(trials).AsNumeric()
let U2 = R.runif(trials).AsNumeric()
//defining accept function for the rejection process
```

```
let accept_function scaling_factor (u1, u2) =
    let beta_res =
        R.dbeta(u1,shape1=2,shape2=5).AsNumeric()
        |> Seq.head
    scaling_factor*u2 < beta_res

//creating a sample rbeta with the rejection funciton
let rbeta =
    Seq.zip U1 U2
    |> Seq.filter (accept_function 2.5)
    |> Seq.map fst
```

Calculating acceptance ratio and plotting the data in a histogram.

```
let accepted = 100.0 * float(rbeta |> Seq.length) / float trials
namedParams [
    "x", box rbeta
    "main", box (sprintf "beta(2,5) from uniform dist (%A%% accepted)"
       accepted)
    "breaks", box 50
    "probability", box true
]
|> R.hist

let xs = [0.0 .. 0.01 .. 1.0]
let beta_2_5_fun =
    xs
    |> List.map (fun x -> Seq.head(R.dbeta(x, 2, 5).AsNumeric()))
namedParams [
    "x", box xs
    "y",box beta_2_5_fun
    "col", box "red"
]
|> R.lines
```



The histogram reveals that the sampled data from the rejection process seems to follow the beta dist fairly well.

Performing statistical check on the sample distribution to see if it follows the cauchy dist.

```
namedParams [
    "x", box rbeta
    "y", box "pbeta"
    "shape1",box 2
    "shape2",box 5
]
|> R.ks_test
|> fun r -> r.Print()
```

Outputs...

```
val it : string =
  "
    One-sample Kolmogorov-Smirnov test

data:  fsr_10176_6
D = 0.0088649, p-value = 0.9091
alternative hypothesis: two-sided
"
```

we get a fairly high p-value and thus reject the null hypothesis that the generated numbers doesn't follow a beta dist.

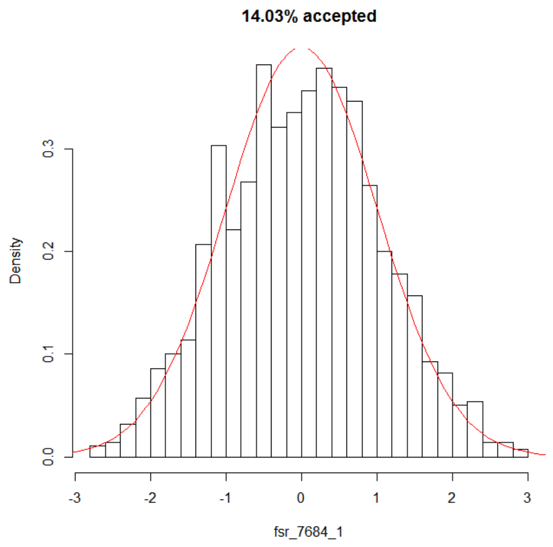
2.3 Simulate a sample from an N(0,1) distribution with the rejection method using the double exponential distribution as auxiliary variable.

Sampling random data uniformly within the range [-3;3] (to represent 3 std) and random data from the double exponential dist, and defining the accept function for the rejection process.

```
let trials = 10000
//sampling random data uniformly for 3 standard deviations
//3 standard deviations account for 99.7%
let U1 = R.runif(trials,min = -3,max = 3).AsNumeric()
//sampling random data from the exponential distribution
let E = R.rexp(trials,1).AsNumeric()
         |> List.ofSeq

let accept_function scaling_factor (u1,u2) =
    let norm_res =
        R.dnorm(u1,mean=0,sd=1).AsNumeric()
        |> Seq.head
    scaling_factor*u2 < norm_res

//generating the N(0,1) distributed data using the rejection method and
   double exponential function as auxiliary funciton
let rnorm =
    Seq.zip U1 E
    |> Seq.filter (accept_function 1.0)
    |> Seq.map fst
```

Calculating acceptance ratio and plotting the data in a histogram.

```
let accepted = 100.0 * float(rnorm |> Seq.length) / float trials
namedParams [
    "x", box rnorm
    "main", box (sprintf "%A%% accepted" accepted)
    "breaks", box 25
    "probability", box true
]
|> R.hist

let xs = [-5.0 .. 0.1 .. 5.0]
let norm_0_1_fun =
    xs
    |> List.map (fun x -> Seq.head(R.dnorm(x, 0, 1).AsNumeric()))
namedParams [
    "x", box xs
    "y",box norm_0_1_fun
    "col", box "red"
]
|> R.lines
```



The histogram reveals that the sampled data from the rejection process seems to have generated data that follow the normal dist N(0,1) fairly well.

Performing statistical check on the sample distribution to see if it follows the normal dist.

```
let ks_res =
    namedParams [
        "x", box rnorm
        "y", box "pnorm"
        "mean",box 0
        "sd",box 1
    ]
    |> R.ks_test
    |> fun r -> match r.AsList().["p.value"].Value with
                | :? array<float> as a -> a
                | _ -> [|0.0|]
    |> Seq.head
```

Outputs...

```
val ks_res : float = 0.9356775536
```

we get a p-value higher than the significant interval and thus reject the null hypothesis that the generated numbers doesn't follow a normal dist.

2.4 One throws a needle of length l at random on a parquet oor with planck width l. Study by simulation this experiment and estimate the probability of the needle to intersect a line between two planks?

This can be solved by generating a dataset consisting of random coordinates and angles, to represent the droped needles. For the coordinate, we only need to calculate the x part, since we only are interested in the x value of the ending coordinate too see if it overlaps one of the plank gaps.

Sampling the needed random data to represent the droped needles, and defining the funciton that checks for intersection.

```
let l = 1.0 //length of the needles
let needle_simulations = 100000
let needle_head_x = R.runif(needle_simulations, min=0, max=l).AsNumeric()
let needle_angle = R.runif(needle_simulations, min=0, max=360).AsNumeric()

//function that checks if a given needle intersects with any planck gap
let intersects x angle =
    let x' = x + l * Math.Cos angle
    if x' < 0.0 then
        true
    else if x' > l then
        true
    else
        false
```

Calculating the needle intersections and the resulting probability for intersection.

```
let simulations =
    Seq.zip needle_head_x needle_angle
    |> Seq.map (fun (x,a) -> intersects x a)

let needle_intersect_prob =
    let intersections =
        simulations
        |> Seq.filter id //'id' is the identity function
        |> Seq.length
    float(intersections) / float(needle_simulations)
```

Outputs

```
val needle_intersect_prob : float = 0.63534
```

thus we can conclude that the probability for intersection can be estimated to around 63%.

3 Exercises II

3.1 You want to sell your car. You receive a rst oer at a price of 12 (say in K Euro) Since it is the rst oer, you decide to reject it and wait a little bit to smell the market. You reject oers and wait until you get the rst oer strictly larger than 12. Study by simulation the waiting time (counted in number of oers untill the sell). You can for instance estimate the expectation of the waiting time. Assume that the Xis are i.i.d Poisson(10). Consider the solution conditionally on X1 = 12, then unconditionally (i.e. in average over all possible X1 values under the assumption that the Xis are i.i.d Poisson(10)). The use of rpois is allowed in this exercise.

This can be calculated by generating a random sequence from the poison(10) dist, identifying the values above 12 and then calculating the average number of values in the sequence between the values that are above 12.

```
//sample n random values from the poisson(10) dist, and divide into offers
   less than and greater than 12, and then extracting the indexes of the
   offers less than 12
let n = 1000000.0
let rpois = R.rpois(n,10).AsNumeric()

let avg_watingtime =
    rpois
    |> Seq.mapi (fun i x -> i, x > 12.0) //mapping to tuple of index and
       true/false
    |> Seq.filter snd
    |> Seq.map fst //extracting indexes
    |> Seq.windowed 2
    |> Seq.map (fun x -> match x with
                         | [|x1;x2|] -> x2 - x1 - 1
                         | _         -> 0)
    |> Seq.map float
    |> Seq.average
```

Outputs...

```
val avg_watingtime : float = 3.803399877
```

thus we can conclude that there is an average waiting time between bids higher than 12, at about 4 bids.

3.2 Variance in the estimation of the median of a Poisson distribution

- Simulate a single dataset of say n = 200 observations from a Poisson distribution with parameter 10 and estimate the theoretical median from your sample.

```
//Estimate the theoretical median from your sample
let n = 200.0
let rpois = R.rpois(n,10).AsNumeric()
let median_val1 = R.median(Array.ofSeq rpois).Print()
```

Outputs...

```
val median_val1 : string = "[1] 10
```

the median is estimated to be 10.

- Implement the bootstrap estimation technique to evaluate the variance of your estimator.

```
//sampling 500 times from the previously random poision distributed sample
   and calculating the mean and median
let bootstrapped =
    [|0 .. 500|]
    |> Array.map
        (fun _ ->
            namedParams [
                "x", box rpois
                "size", box n
                "replace", box true
            ]
            |> R.sample
            |> fun r -> r.AsNumeric()
            |> Array.ofSeq)

let bootstrapped_medians =
    bootstrapped
    |> Array.map (fun xs -> R.median(xs).AsNumeric()
                            |> Seq.head)

R.mean(bootstrapped_medians).Print()
let variance =
    R.sd(bootstrapped_medians).AsNumeric()
    |> Seq.head
    |> fun x -> x**2.0
```

Outputs for the mean median of all the samples...

```
val it : string = "[1] 9.795409
```

Outputs for the standard deviation of the medians of all the samples...

```
val variance : float = 0.1835588822
```

Thus by using the bootstrap method we have fund that estimating the median based on a 200 sample, has a variance of 0.183

3.3 Simulate a sample of size 50000 from a bivariate centred, standardized Gaussian vector with correlation coeffcient rho = 0.7. Plot this sample. Extract a sample of a random variable approximately distributed as Y2—Y1 = .5. Estimate its mean and variance. Plot its empirical histogram and compare it to a normal density with same parameters as the one simulated above.

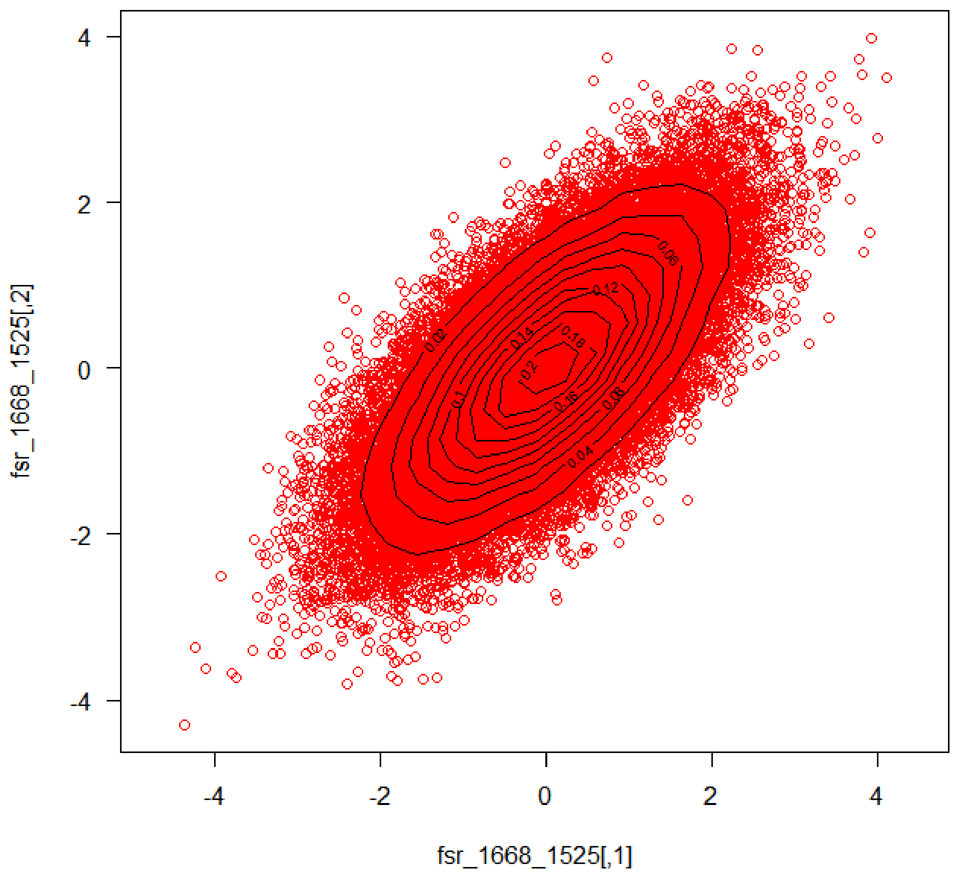
sampling random bivariate normal dist data and performing the cholesky decomposition of the given covariance matrix.

```
let n = 50000
//target covariance matrix
let M = array2D [[1.0; 0.7]
                 [0.7; 1.0]]

//performing cholesky decomposition
let U = R.chol(M).AsNumericMatrix().ToArray()
let L = R.t(U).AsNumericMatrix().ToArray()
```

```
//generate bivariate random normal distributed data set
let x1 = R.rnorm(n,0,1).AsNumeric()
let x2 = R.rnorm(n,0,1).AsNumeric()
let X = array2D [x1
                 x2]

//performing matrix multiplication between the cholesky decomposed matrix
   and the random bivariate sample
let Y = R.''%*%''(L, X).AsNumericMatrix().ToArray()
let Y' = R.t(Y).AsNumericMatrix().ToArray()
```

Plotting the resulting bivariate random sample

```
namedParams [
    "x", box Y'
    "asp", box 1
    "las", box 1
    "cex", box 1
    "col", box 2
]
|> R.plot
namedParams [
    "x", box <| R.kde2d(Y.[0,0..], Y.[1,0..])
    "add", box true
]
|> R.contour
```



Testing that the covariance matrix of the sample is approximately the same as the target covariance matrix.

```
//verifying that the covariance matrix is similiar to the initial covariance
    matrix
R.cov(Y').AsNumericMatrix().ToArray()
```

Outputs...

```
val it : float [,] = [[1.002158123; 0.7031856323]
                       [0.7031856323; 0.9995219499]]
```

The covariance matrix of the sample is very close to the target covariance matrix, so the sampling has been done successfully.

Extracting a slice of the multivariate data at Y1=0.5, and estimating the mean and variance of the slice.

```
//Extract a sample of a random variable approximately distributed as Y2|Y1
   =.5
//no Y1 values are exactly 0.5
//so we have to sample from an interval, for example 0.49 to 0.51
let interval x = 0.49 <= x && x <= 0.51
Y.[0,0..]
|> Seq.filter (fun x -> interval x)
|> Seq.length
```

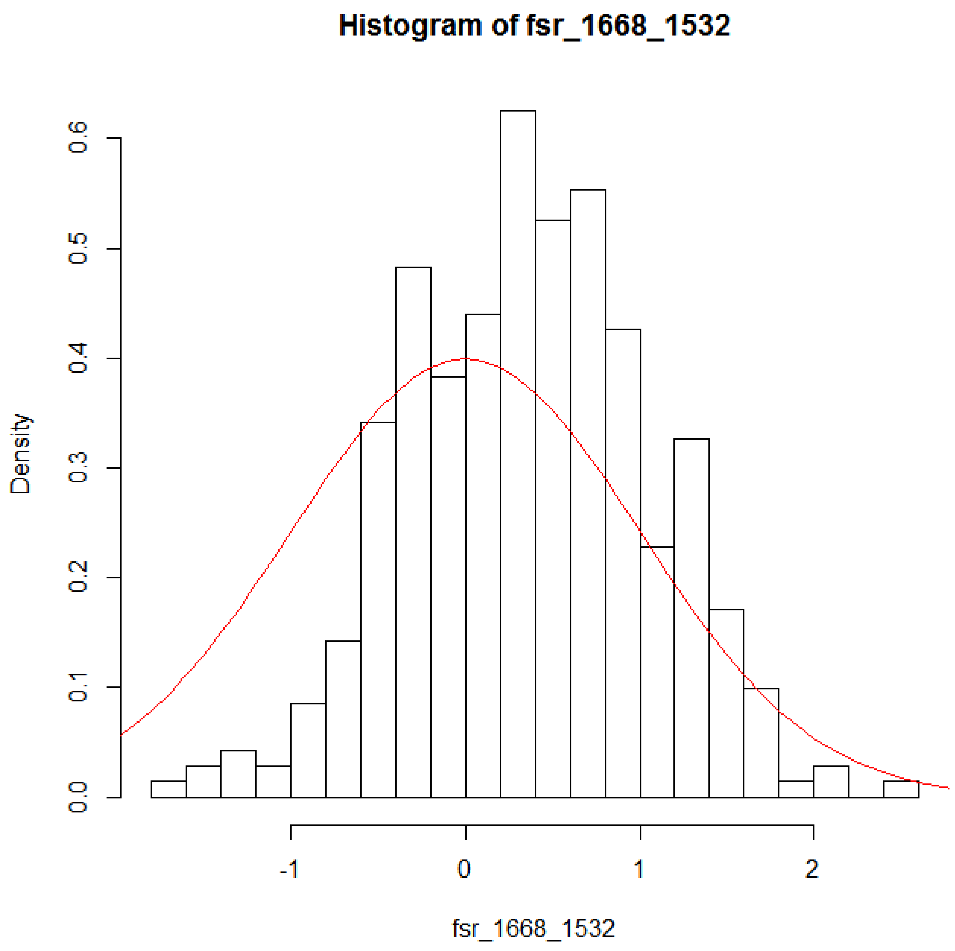
```
//extracting values
let sample_slice =
    [0 .. Array2D.length1 Y' - 1]
    |> Seq.map (fun i -> interval Y'.[i,0], Y'.[i,1])
    |> Seq.filter fst
    |> Seq.map snd

let mean = R.mean(sample_slice).AsNumeric() |> Seq.head
let varaince = R.var(sample_slice).AsNumeric() |> Seq.head
```

Outputs...

```
val mean : float = 0.3587044098
val varaince : float = 0.4812317728
```

Plotting the data slice.

```
namedParams [
    "x", box sample_slice
    "breaks", box 25
    "probability", box true
]
|> R.hist
let xs = [-4.0 .. 0.1 .. 4.0]
let normal_fun =
    xs
    |> List.map (fun x -> Seq.head(R.dnorm(x, 0, 1).AsNumeric()))
namedParams [
    "x", box xs
    "y",box normal_fun
    "col", box "red"
]
|> R.lines
```



The data in the histogram is slightly off the plotted normal distribution, however, the skewness makes sense when observing the position of the slice at the multivariate distribution plot.