Day1 - Generating pseudo 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 Exercise on pseudo RNG simulation

2.1 congruence method

- Implement the congruence method a = 16807,b = 0 and M = 2147483647 and Simulate 1000 realisations on [0,1]

```
//[1] - Implement the congruence method a = 16807,b = 0 and M = 2147483647.
let congruential_method (initial: int) (a: int) (b: int) (M: int) =
    let rec loop prev = seq {
        let x = (a * prev + b) % M
        yield float x / float M
        yield! loop x
    }
    seq {
        yield float initial / float M
        yield! loop initial
    }

//[2] - Simulate 1000 realisations on [0,1]
let seed = 3
let a = 16807
let b = 0
let M = 2147483647
let U1 = congruential_method seed a b M
         |> Seq.take 1000
         |> List.ofSeq
```

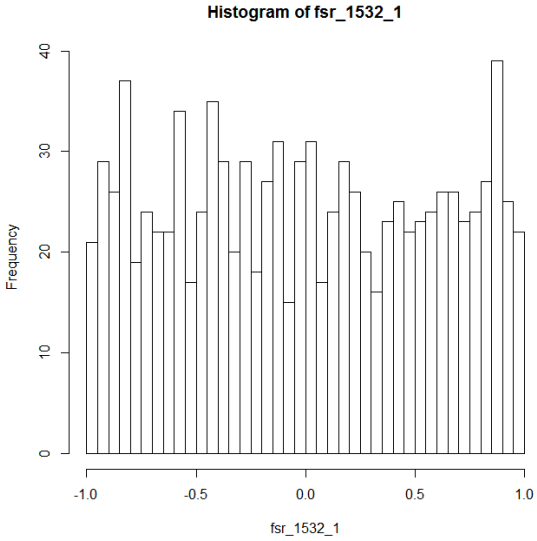
Outputs...

```
val U1 : float list =
  [1.396983863e-09; 2.347910778e-05; 0.3946133644; 0.2668128783;
     0.3240438268;
   0.204594771; 0.6243147196; 0.8574880389; -0.1985371994; -0.8147085066;
   -0.8058640872; -0.157707525; -0.5903718609; -0.3798620605; -0.3416475418;
   -0.07023194203; -0.388249098; 0.6974123072; -0.5913577888; -0.9503519367;
   -0.5649930078; 0.1625217; -0.4977896276; -0.3502678253; -0.9513378432;
   0.8648771298; -0.01008551429; 0.4927614804; -0.1578030126; -0.1952314876;
   0.7443894389; 0.9532934753; 0.003431917635; -0.3197603278; -0.2118275376;
   -0.1854229398; -0.4033477741; 0.9339633388; -0.8781718676; 0.5654275443;
   -0.8592669656; 0.3001161466; 0.05207336836; -0.8028983603; -0.3127356904;
   -0.1487463616; 0.01990243188; 0.5001724686; 0.3986758606; 0.5451852631;
   0.9287119303; 0.8614045558; -0.3736376601; 0.2718493739; 0.9724258128;
   -0.4393711665; -0.5111917544; 0.4001882325; -0.0363796265; 0.5676176425;
   -0.05028708142; 0.8250230056; 0.161649251; 0.8389597264; 0.396115543;
   -0.4860712343; 0.6007684262; -0.8850647639; 0.7165201473; 0.554109494;
   0.9182612244; -0.7836092691; -0.1209793971; 0.6992730748; 0.6825633141;
   -0.1583858259; 0.009425626606; 0.4165063004; 0.2213877082; 0.8632099358;
   -0.03061622243; -0.5668501312; 0.9498497639; 0.1249737912; 0.4345084594;
   0.783673386; -0.8014078205; 0.7387679246; 0.4725029187; -0.6434496397;
   -0.4580899987; 0.8813959238; -0.378714754; 0.9411332113; -0.3741243842;
   0.09147778391; -0.5328866232; -0.2254717402; 0.4964637549; 0.06632490692;
   ...]
```

2.2 Make some checks of the randomness of the sequence produced

- Plot a histogram of the sample

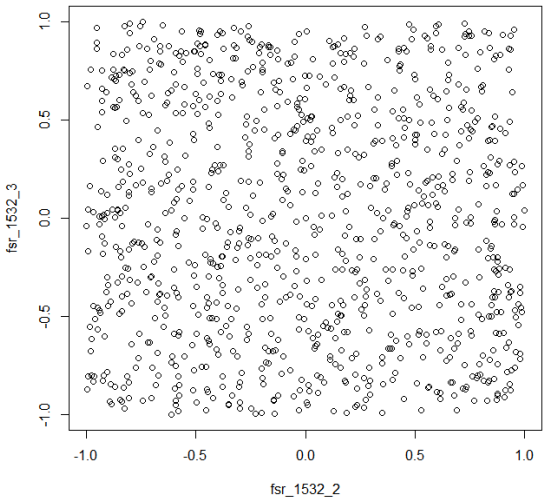
```
//Plot a histogram of the sample
namedParams [
    "x", box U1
    "breaks", box 50
]
|> R.hist
```



Based on the histogram, it looks like the sample size is too small to make out a proper uniform random distribution

- Illustrate graphically the distribution of (xk,xk+2) by a scatter plot

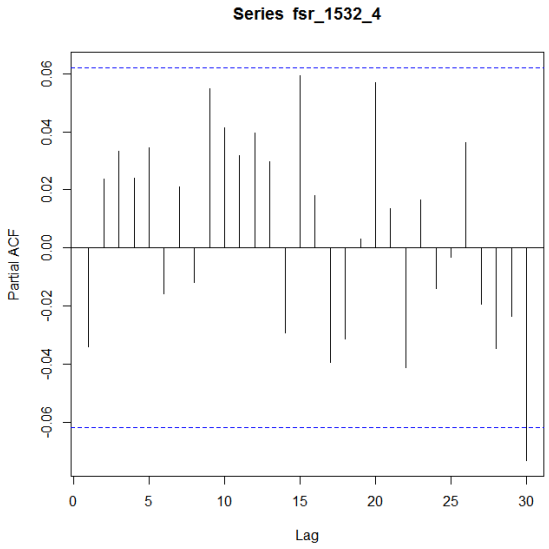
```
//Illustrate graphically the distribution of (xk,xk+2) by a scatter plot
R.plot(Seq.take (U1.Length-1) U1, List.tail U1)
```



The scatter plot of the data seems well distributed, and doesn't reveal any flaws to the randomness of the data

- Plot the auto-correlation function of the sample

```
//Plot the auto-correlation function of the sample
R.pacf U1
```



The autocorrelation diagram doesn't reveal any significant correlations in the sample

- Perform a statistical test of the hypothesis

```
//Perform a statistical test of the hypothesis that the simulated values
namedParams [
    "x", box U1
    "y", box "punif"
    "min",box -1
    "max",box 1
]
|> R.ks_test
|> fun r -> r.Print()
```

Outputs...

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

data:  fsr_1532_5
D = 0.017819, p-value = 0.9086
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 uniform dist

2.3 congruence method - with chosen values

- Repeat the previous steps with parameters a,b and M of your choice

```
//[3] - Repeat the previous steps with parameters a,b and M of your choice
let seed2 = 5
let a2 = 1373
let b2 = 899
let M2 = 7777
let U2 = congruential_method seed2 a2 b2 M2
         |> Seq.take 10000
         |> List.ofSeq
```

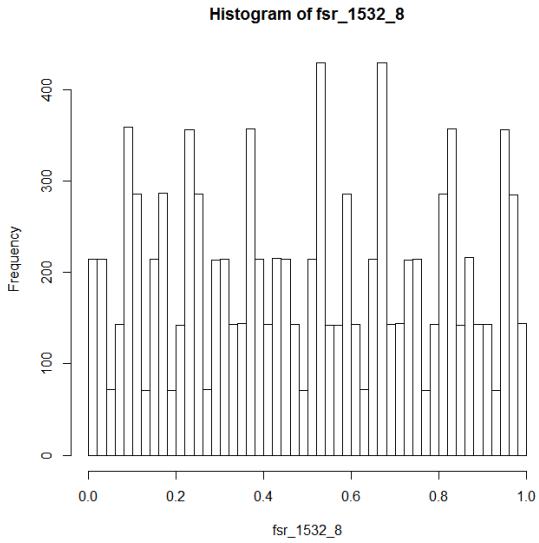
Outputs...

```
val U2 : float list =
  [0.000642921435; 0.9983284043; 0.8204963353; 0.6570657066; 0.2668123955;
   0.4490163302; 0.6150186447; 0.5361964768; 0.3133599074; 0.3587501607;
   0.6795679568; 0.1624019545; 0.09348077665; 0.4647036132; 0.153658223;
   0.08833740517; 0.4028545712; 0.2349234923; 0.6655522695; 0.9188633149;
   0.7149286357; 0.7126141186; 0.5347820496; 0.3713514209; 0.9810981098;
   0.1633020445; 0.329304359; 0.2504821911; 0.02764562171; 0.07303587502;
   0.3938536711; 0.8766876688; 0.8077664909; 0.1789893275; 0.8679439373;
   0.8026231195; 0.1171402855; 0.9492092066; 0.3798379838; 0.6331490292;
   0.42921435; 0.4268998328; 0.2490677639; 0.08563713514; 0.6953838241;
   0.8775877588; 0.04359007329; 0.9647679054; 0.741931336; 0.7873215893;
   0.1081393854; 0.5909733831; 0.5220522052; 0.8932750418; 0.5822296515;
   0.5169088337; 0.8314259997; 0.6634949209; 0.09412369808; 0.3474347435;
   0.1435000643; 0.1411855471; 0.9633534782; 0.7999228494; 0.4096695384;
   0.5918734731; 0.7578757876; 0.6790536196; 0.4562170503; 0.5016073036;
   0.8224250997; 0.3052590973; 0.2363379195; 0.6075607561; 0.2965153658;
   0.231194548; 0.545711714; 0.3777806352; 0.8084094124; 0.06172045776;
   0.8577857786; 0.8554712614; 0.6776391925; 0.5142085637; 0.1239552527;
   0.3061591873; 0.4721615019; 0.3933393339; 0.1705027646; 0.2158930179;
   0.5367108139; 0.01954481162; 0.9506236338; 0.3218464704; 0.01080108011;
   0.9454802623; 0.2599974283; 0.09206634949; 0.5226951267; 0.776006172;
      ...]
```

2.4 Make some checks of the randomness of the sequence produced

- Plot a histogram of the sample

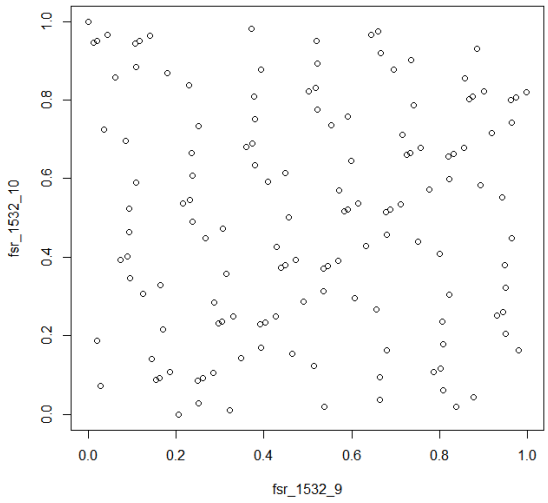
```
namedParams [
    "x", box U2
    "breaks", box 50
]
|> R.hist
```



The histogram reveals that this data most likely is not uniformly dist

- Illustrate graphically the distribution of (xk,xk+2) by a scatter plot

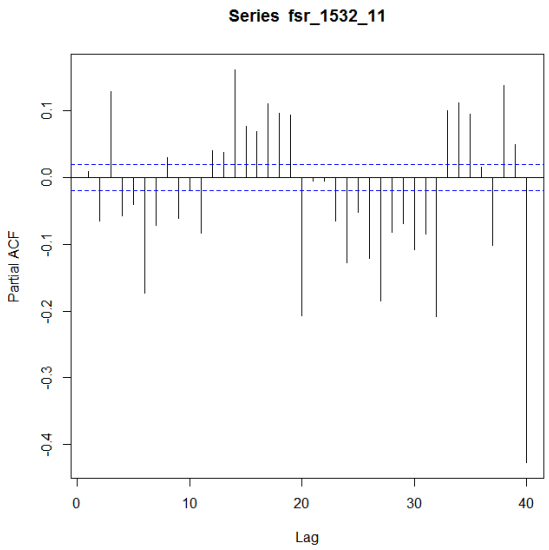
```
//Illustrate graphically the distribution of (xk,xk+2) by a scatter plot
R.plot(Seq.take (U2.Length-1) U2, Seq.skip 1 U2)
```



It seems a lot of the data points are stacked on the scatter plot, which might entail that the distribution isn't uniform

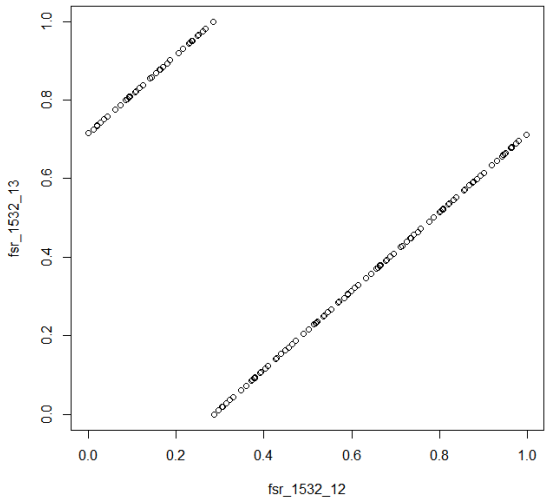
- Plot the auto-correlation function of the sample

```
//Plot the auto-correlation function of the sample
R.pacf U2
```



The autocorrelation diagram reveals that there is autocorrelation between multiple of the time lags and therefore this data cannot be uniformly dist

To get a sense of the autocorrelation, we plot a scatter plot with time lag set to 20



The scatter plot clearly visualizes the autocorrelation for time lag 20

- Perform a statistical test of the hypothesis

```
//Perform a statistical test of the hypothesis that the simulated values
namedParams [
    "x", box U2
    "y", box "punif"
    "min",box -1
    "max",box 1
]
|> R.ks_test
|> fun r -> r.Print()
```

Outputs...

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

data:  fsr_1532_14
D = 0.50032, p-value < 2.2e-16
alternative hypothesis: two-sided
"
```

we get a very low p-value and can thus accept the null hypothesis that the generated numbers doesn't follow a uniform dist