UNIVERSITEIT VAN AMSTERDAM

KANSREKENING EN STATISTIEK

LAB-3

Authors:
Abe WIERSMA

20 april 2015

1 Exercise: Random Number Generators

1.1 Rekenen met stochasten

$$E(X) = \sum_x x P_X(x) = \sum_x \sum_y x P_{XY}(x, y)$$

$$E(X) + E(Y) = \sum_x \sum_y x P_{XY}(x, y) + \sum_y \sum_y y P_{XY}(x, y) =$$

$$\sum_x \sum_y (x + y) P_{XY}(x, y) = E(X + Y)$$

1.2 Uniform verdeelde stochasten

```
import random
import pylab
import numpy

def ibm_random(n, x0):
    random_ibm = [x0]
    m = numpy.power(2.0, 31)
    for i in range(n):
        random_ibm.append(((65539.0 * random_ibm[i]) + 0.0) % m)
    random_ibm.pop(0)

    random_ibm = random_ibm / m
    return random_ibm

n = 1000
pairs = []
for i in range(n):
    pairs.append([random.random(), random.random()])
pairs = numpy.array(pairs)
pylab.plot(pairs[:, 0], pairs[:, 1], 'ro')
pylab.show()

ibm_pairs = numpy.column_stack((ibm_random(n, random.random()),
                                ibm_random(n, random.random())))
ibm_pairs = numpy.array(ibm_pairs)
pylab.plot(ibm_pairs[:, 0], ibm_pairs[:, 1], 'ro')
pylab.show()
# To the naked eye the pairs seem equally well distributed with both RNGs.
```

2.3

2.4 Numpy (as well as the random library) uses the Mersenne Twister pseudorandom generator. The domain of this generator is far larger than IBM's generator, so the chance of repetition is far smaller. Also, you will need far more consecutive numbers to spot the pattern and be able to predict the further output of the generator. But it is still possible with Mersenne Twister, so if you are concerned with safety another generator should be used.

1.3 De exponentile verdeling

3.1 de verwachting is $\lambda^{-}1$. De variantie is $\lambda^{-}2$
3.2 λ is de intensiteitsparameter. Met een hogere λ zal de functie met een hogere waarde beginnen, maar hij zal ook sneller afnemen.
3.3 Het gemiddelde van de waarden die eruit komen zouden (ongeveer) gelijk moeten zijn aan de verwachting. Dus je neem het gemiddelde van de gevonden waardes en verheft die tot de macht -1.
3.4

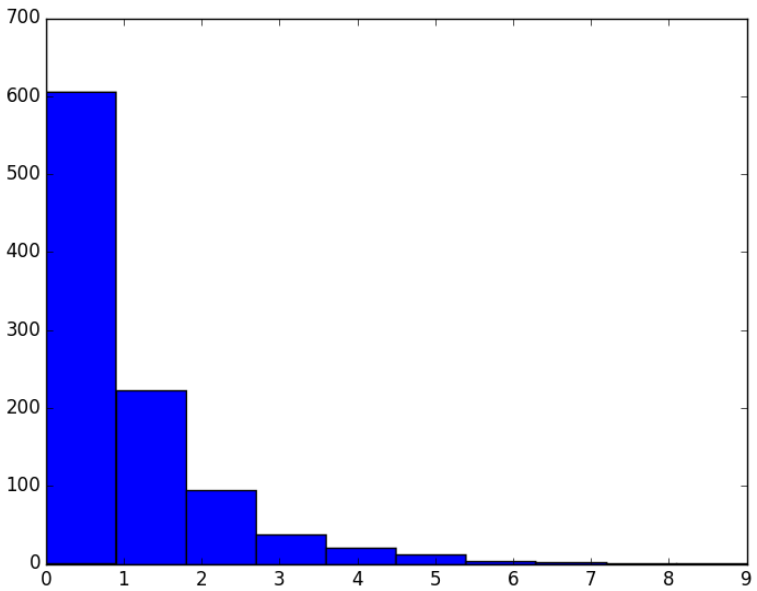
```
import random
import numpy
import matplotlib.pyplot as pyplot

def inverse_transform_sampling(lmbd, sample_size):
    """ CDF = 1 - e^(lambda x) it's inverse is -ln(-x+1)/lambda. """
    samples = []
    for i in range(sample_size):
        u = random.random()
        samples.append(- numpy.log(-u + 1) / lmbd)
    return samples

samples = inverse_transform_sampling(1, 1000)
n, bins, patches = pyplot.hist(samples)
pyplot.show()

lmbd = 1 / numpy.mean(samples)
print("lambda = ", lmbd)
```

3.5



Figuur 1: Exponentile verdeling

3.6 Lambda = 0.973995802533

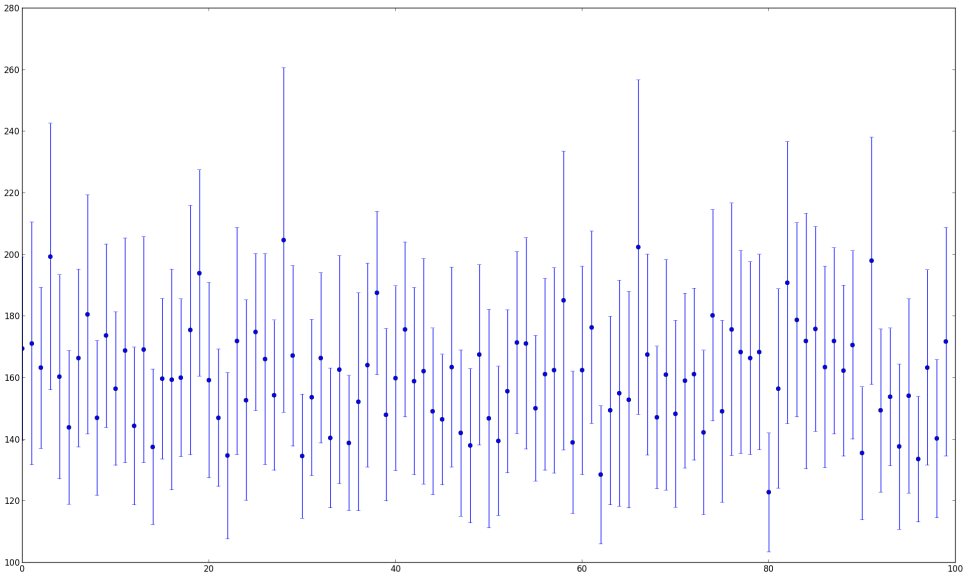
2 Exercise: Betrouwbaarheids Intervallen

1. Voor deze oefening is het de bedoeling een interval te bepalen uit een random selectie van 50 waarvoor het gemiddelde van de hele populatie met 95% zekerheid binnen het interval ligt.

2. $Z = \frac{\bar{X}-\mu}{\sigma/\sqrt{n}}$
   $P(-z \leq Z \leq z) = 1 - \alpha = 0,95.$
   $P(Z \leq z) = 1 - \frac{\alpha}{2} = 0{,}975$
   $z = \Phi^{-1}(0.975) = 1.96$

   $\hat{\mu} = \bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$
   Lower endpoint $= \bar{\mu} - 1.96\frac{\sigma}{\sqrt{n}}$
   Upper endpoint $= \bar{\mu} + 1.96\frac{\sigma}{\sqrt{n}}$

3. result:



Figuur 2: results

2.1 The Code

```
import math
import numpy
import re
import random
import matplotlib.pyplot as plt
from scipy.stats import norm

within = 0
not_in = 0.05
enddata = []
yerr = []

# reading of data, with regex :)
rawdata = []
```

```
ptrn = '([1-9][0-9]*.[0-9]*)$'
lines = [line.strip() for line in open('tijden-medium.log')]
for line in lines:
    searchResult = re.search(ptrn, line)
    if searchResult:
        if searchResult.group(1):
            rawdata.append(float(searchResult.group(1)))
    else:
        print('line did not pass regex')

for pla in range(100):
    # transform array to numpy array, for further numpy use.
    data = numpy.array(rawdata)
    data_mean = numpy.mean(data)

    # draw fifty random numbers from list.
    fifty_random = []
    for i in range(50):
        random_index = random.randrange(len(data))
        fifty_random.append(data[random_index])
        data = numpy.delete(data, random_index)

    fifty_random = numpy.array(fifty_random)

    random_mean = numpy.mean(fifty_random)
    random_sigma = numpy.sqrt(numpy.var(fifty_random))

    # definition of Cumulative distributive function of x, in this case:
    # 1-(a/2)
    z = norm.ppf(1.0 - (0.5 * not_in))
    lower = random_mean - (z * random_sigma /
                           numpy.sqrt(50))
    upper = random_mean + (z * random_sigma /
                           numpy.sqrt(50))
    if data_mean >= lower and data_mean <= upper:
        within += 1

    enddata.append(random_mean)
    yerr.append([math.fabs(lower - random_mean),
                 math.fabs(upper - random_mean)])

enddata = numpy.array(enddata)
yerr = numpy.array(yerr)
plt.errorbar(range(100), enddata, yerr=yerr.T, fmt='o')
plt.show()
print('full collection mean within: ' + str(within) + ' of: ' + str(100))
```