الجامعة المصرية اليابانية للعلوم والتكنولوجيا
E-JUST
Egypt - Japan University of Science and Technology
エジプト日本科学技術大学

Random Number Generator
A Hybrid Approach

To: Prof. Sherif I. Rabia

Methods of Random Numbers Generation

1- True (hardware) random number generators:

These methods take their values from a measurement of some real chaotic physical phenomena rather than a known algorithm. Some examples of these phenomena are thermal noise, the photoelectric effect, and some other quantum phenomena. They are theoretically unpredictable in comparison with the other methods followed. Therefore, they are widely used in fields like cryptography (e.g., internet protocols) and simulation (to imitate real world noises).

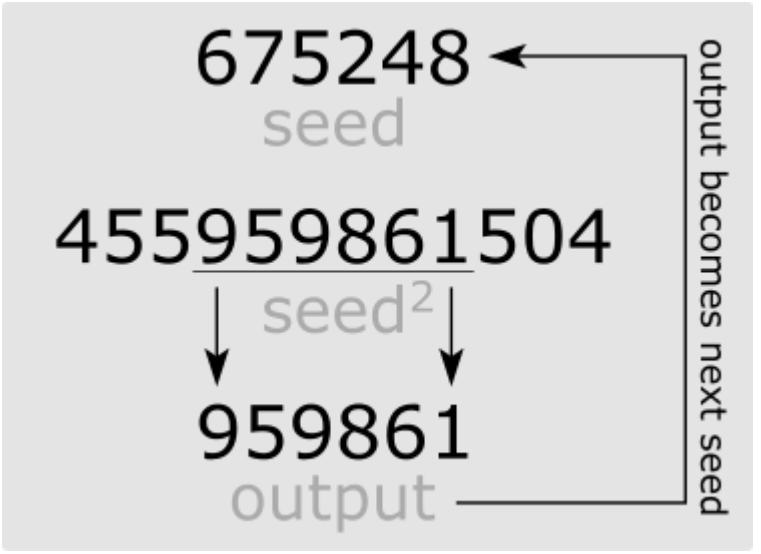
Methods of Random Numbers Generation

2- Pseudo random number generators:

These methods depend mainly on a followed algorithm. There are a lot of known algorithms to generate random numbers like Middle-square (from the oldest) and Linear congruential generator (LFG). These algorithms are used for random number generators in most of the programming languages. They cannot be used for security purposes as they can be predicted. For instance, in the Middle square method, the seed is the key to predicting the random numbers generated from the algorithm.

Middle-square method

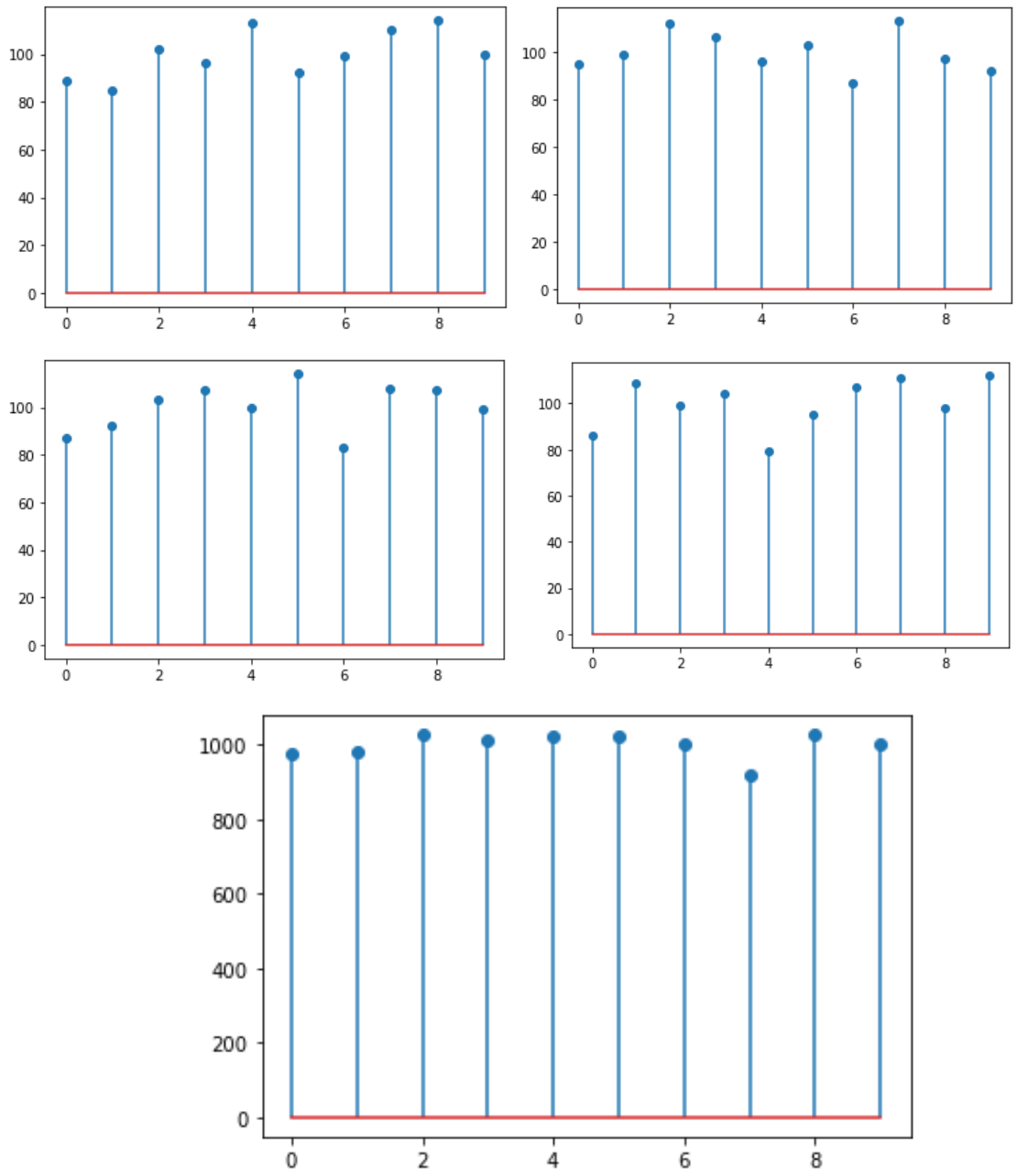
To get n-digit random number, an n-digit value is chosen (the seed) and squared, generating a number of 2n-digits at maximum. If the result has less than 2n-digits, leading zeroes are added to compensate. The middle n-digits will be the taken as the seed for the next number and the sequence continues. Notably, the value of n must be even for the middle n numbers to be defined. However, sometimes it is acceptable to add leading zeroes to create even digits number. This method is of no practical use because it gets stuck. For example, if the middle numbers are all zeroes, the algorithm outputs zeroes forever.



A hybrid method

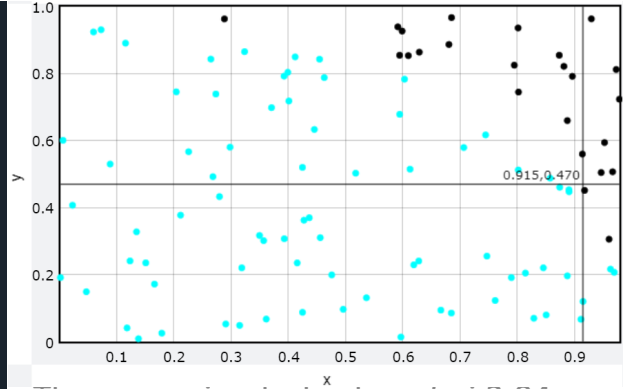
Our method is a modified implementation of the middle-square algorithm but with a theoretical true random seed. As stated in Python's official documentation, time.time() function returns the number of seconds elapsed since epoch in decimal format. The number on the right side of the output represents the milliseconds between the seconds (which are completely unpredictable numbers). As stated, we made use of the milliseconds part to generate the seed of the algorithm.

```
'''
This is a two part program that can run seperately
First part is a code to generating random integers using A Hybrid Approach
Second part is finding an approximate value of Pi using random numbers
'''
program = input('Type "R" to run the random integers generator or "P" to run Pi value program: ')
if program == "R" or program=='r':


  #generating random integers using A Hybrid Approach

  # from sys import argv
  from matplotlib import pyplot
  import time
  from collections import Counter


  def get_seed():
      seed_in = str(time.time())
      list1= seed_in.split(".")
      seed = list1[1]
      return (seed)


  def mdl_sqr_method(seed):
      sqrd= int(seed)**2
      sqrd_str = str(sqrd)
      return (sqrd_str)


  def get_mdl_char(s,n,rslt_list):
      d=len(s)-n
      if n:
          l=int(s[(d+1)//2:len(s)-(d//2)])
          while len(str(l)) != n:
              seed = int(get_seed())
              sqrd_str =mdl_sqr_method(seed)
              try:
                  get_mdl_char(sqrd_str,length_random)
              except:
                  l= int('1'+str(l))
          rslt_list.append(l)
```

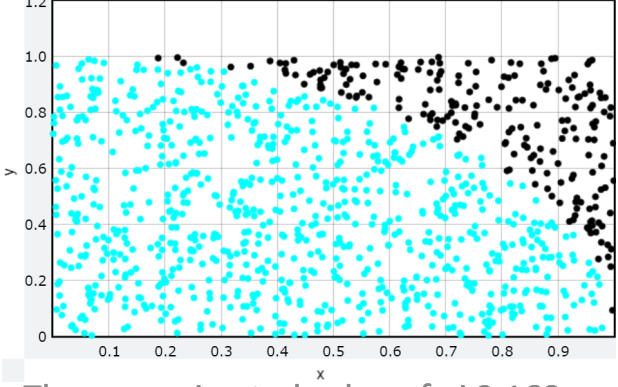


Different samples of random numbers from 0 to 9

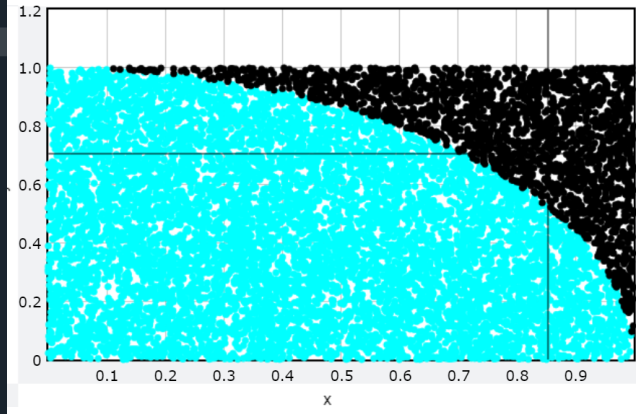
```
#Approximating Pi

import random
plot = input('Do you want to see visual plot of the data (Y/N)?')
if plot =='Y' or plot =='y':
  from vpython import*

  tgraph=graph(xtitle="x",ytitle="y")
  p1=gdots(color=color.cyan)
  p2=gdots(color=color.black)

  points_total= 0
  PointsSmallerThanOne=0


  while points_total<100000: #A bigger number will return a more accurte approximation of Pi
                             #but will take more time to return the result, so change as you like
    rate(1000)    #To visualize the process
    x=random()
    y=random()
    r=vector(x,y,0)
    if mag(r)<1:
      PointsSmallerThanOne+=1
      p1.plot(r.x,r.y)
    else:
      p2.plot(r.x,r.y)
    points_total+=1

  pi_approx = 4*PointsSmallerThanOne/points_total
  print("The approximated value of pi based on the size of points chosen is about",pi_approx)
else:
  points_total= 0
  PointsSmallerThanOne=0
  while points_total<100000: #A bigger number will return a more accurte approximation of Pi
                             #but will take more time to return the result, so change as you like
    x=random.random()
    y=random.random()
    r=((x**2)+(y**2))**0.5
    if r<1:
      PointsSmallerThanOne+=1
    points_total+=1

  pi_approx = 4*PointsSmallerThanOne/points_total
  print("The approximated value of pi based on the size of points chosen is about",pi_approx)
```



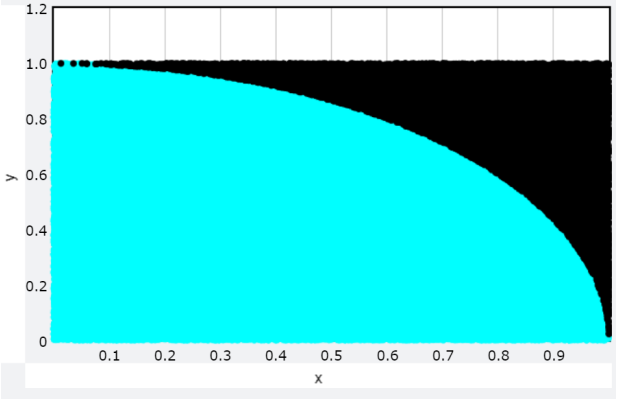
The approximated value of pi 3.04, based on 10^2 points



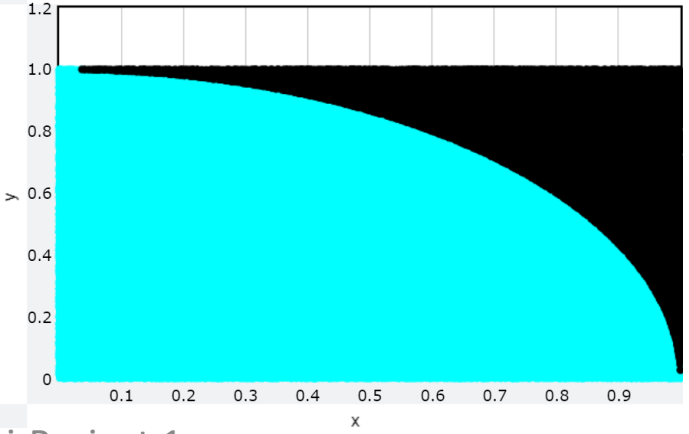
The approximated value of pi 3.168, based on 10^3 points



The approximated value of pi 3.1512, based on 10^4 points



The approximated value of pi 3.1422, based on 10^5 points



The approximated value of pi 3.144684, based on 10^6 points and value of 3.1418812 based on 10^7 points