Kernel: SageMath 10.4

Pseudorandom Numbers

Math 242 Modern Computational Math

How do computers generate "random" numbers? Today we will consider two algorithms that produce *pseudorandom* numbers.

Middle-Square Method

One early approach for creating pseudorandom numbers is the *middle-square method*. This method is simple. Start with a number with an even number of digits. Suppose the number is N and it has k digits. Square N, producing a number with $2k$ digits. Extract the middle k of these digits and repeat. This produces a sequence of k-digit numbers, which we can regard as a longer sequence of decimal digits. For some starting values, this sequence of decimal digits will appear approximately random.

In [1]:
```
# determine how many digits are in the decimal representation of a
number
def howManyDigits(num):
    return ceil(log(num, 10))  # log base 10 of num
```

In [2]:
```
howManyDigits(1234)
```

Out[2]: 4

In [5]:
```
# compute the middle square sequence
# seed = starting value
# numIter = number of iterations to perform
def middleSquare(seed, numIter):
    # initializations
    num = seed
    seq = [num]
    k = howManyDigits(num)

    for i in range(numIter):
        # square the number
        num = num^2

        # extract the middle k digits
        num = num // (10^(k//2))   # forgets the last k/2 digits
        num = num % (10^k)         # take the next k digits
        #print(num)

        # append the middle k digits to the sequence
        seq.append(num)

    return seq
```

In [4]:
```
middleSquare(5146, 10)
```

Out[4]:
```
4813
1649
7192
7248
5335
4622
3628
1623
6341
2082

[5146, 4813, 1649, 7192, 7248, 5335, 4622, 3628, 1623, 6341, 2082]
```

It would be useful to have a function that converts a list of numbers into a list of decimal digits. Here is such a function:

In [6]:
```
# convert a list of big numbers to a list of digits in order
def extractDigits(seq):
    # initialize a list for the digits
    allDigits = []

    # use the initial value to determine how many digits per number
    digitsPerNum = howManyDigits(seq[0])

    # loop over all numbers in the sequence
    for num in seq:
        # initialize a list for the digits in num
        digits = []

        # extract the digits in num, starting with the ones place
        for i in range(digitsPerNum):
            digits.append(num % 10)
            num = num // 10

        # reverse the digits, then concatenate digits to the end of
allDigits
        digits.reverse()
        allDigits += digits

    # return the big list of digits
    return allDigits
```

Observe how `extractDigits` works:

In [7]:
```
extractDigits([1234, 77, 12345, 9876])
```

Out[7]:
```
[1, 2, 3, 4, 0, 0, 7, 7, 2, 3, 4, 5, 9, 8, 7, 6]
```

In [8]:
```
seq = middleSquare(5146, 10)
print(seq)
print(extractDigits(seq))
```

Out[8]:
```
[5146, 4813, 1649, 7192, 7248, 5335, 4622, 3628, 1623, 6341, 2082]
[5, 1, 4, 6, 4, 8, 1, 3, 1, 6, 4, 9, 7, 1, 9, 2, 7, 2, 4, 8, 5, 3, 3, 5,
4, 6, 2, 2, 3, 6, 2, 8, 1, 6, 2, 3, 6, 3, 4, 1, 2, 0, 8, 2]
```

In [9]:
```
seq = middleSquare(2500, 10)
print(seq)
print(extractDigits(seq))
```

Out[9]:
```
[2500, 2500, 2500, 2500, 2500, 2500, 2500, 2500, 2500, 2500, 2500]
[2, 5, 0, 0, 2, 5, 0, 0, 2, 5, 0, 0, 2, 5, 0, 0, 2, 5, 0, 0, 2, 5, 0, 0,
2, 5, 0, 0, 2, 5, 0, 0, 2, 5, 0, 0, 2, 5, 0, 0, 2, 5, 0, 0]
```

In [10]:
```
seq = middleSquare(5140, 100)
print(seq)
print(extractDigits(seq))
```

Out[10]:
```
[5140, 4196, 6064, 7720, 5984, 8082, 3187, 1569, 4617, 3166, 235, 552,
3047, 2842, 769, 5913, 9635, 8332, 4222, 8252, 955, 9120, 1744, 415,
1722, 9652, 1611, 5953, 4382, 2019, 763, 5821, 8840, 1456, 1199, 4376,
1493, 2290, 2441, 9584, 8530, 7609, 8968, 4250, 625, 3906, 2568, 5946,
3549, 5954, 4501, 2590, 7081, 1405, 9740, 8676, 2729, 4474, 166, 275,
756, 5715, 6612, 7185, 6242, 9625, 6406, 368, 1354, 8333, 4388, 2545,
4770, 7529, 6858, 321, 1030, 609, 3708, 7492, 1300, 6900, 6100, 2100,
4100, 8100, 6100, 2100, 4100, 8100, 6100, 2100, 4100, 8100, 6100, 2100,
4100, 8100, 6100, 2100, 4100]
[5, 1, 4, 0, 4, 1, 9, 6, 6, 0, 6, 4, 7, 7, 2, 0, 5, 9, 8, 4, 8, 0, 8, 2,
3, 1, 8, 7, 1, 5, 6, 9, 4, 6, 1, 7, 3, 1, 6, 6, 0, 2, 3, 5, 0, 5, 5, 2,
3, 0, 4, 7, 2, 8, 4, 2, 0, 7, 6, 9, 5, 9, 1, 3, 9, 6, 3, 5, 8, 3, 3, 2,
4, 2, 2, 2, 8, 2, 5, 2, 0, 9, 5, 5, 9, 1, 2, 0, 1, 7, 4, 4, 0, 4, 1, 5,
1, 7, 2, 2, 9, 6, 5, 2, 1, 6, 1, 1, 5, 9, 5, 3, 4, 3, 8, 2, 2, 0, 1, 9,
0, 7, 6, 3, 5, 8, 2, 1, 8, 8, 4, 0, 1, 4, 5, 6, 1, 1, 9, 9, 4, 3, 7, 6,
1, 4, 9, 3, 2, 2, 9, 0, 2, 4, 4, 1, 9, 5, 8, 4, 8, 5, 3, 0, 7, 6, 0, 9,
8, 9, 6, 8, 4, 2, 5, 0, 0, 6, 2, 5, 3, 9, 0, 6, 2, 5, 6, 8, 5, 9, 4, 6,
3, 5, 4, 9, 5, 9, 5, 4, 4, 5, 0, 1, 2, 5, 9, 0, 7, 0, 8, 1, 1, 4, 0, 5,
9, 7, 4, 0, 8, 6, 7, 6, 2, 7, 2, 9, 4, 4, 7, 4, 0, 1, 6, 6, 0, 2, 7, 5,
0, 7, 5, 6, 5, 7, 1, 5, 6, 6, 1, 2, 7, 1, 8, 5, 6, 2, 4, 2, 9, 6, 2, 5,
6, 4, 0, 6, 0, 3, 6, 8, 1, 3, 5, 4, 8, 3, 3, 3, 4, 3, 8, 8, 2, 5, 4, 5,
4, 7, 7, 0, 7, 5, 2, 9, 6, 8, 5, 8, 0, 3, 2, 1, 1, 0, 3, 0, 0, 6, 0, 9,
3, 7, 0, 8, 7, 4, 9, 2, 1, 3, 0, 0, 6, 9, 0, 0, 6, 1, 0, 0, 2, 1, 0, 0,
4, 1, 0, 0, 8, 1, 0, 0, 6, 1, 0, 0, 2, 1, 0, 0, 4, 1, 0, 0, 8, 1, 0, 0,
6, 1, 0, 0, 2, 1, 0, 0, 4, 1, 0, 0, 8, 1, 0, 0, 6, 1, 0, 0, 2, 1, 0, 0,
4, 1, 0, 0, 8, 1, 0, 0, 6, 1, 0, 0, 2, 1, 0, 0, 4, 1, 0, 0]
```

Questions to investigate

1. Experiment with the middle-square method using various starting values. What do you observe?

2. Do certain starting values produce sequences of digits that you would consider random?

3. Do certain starting values produce sequences of digits that you would consider definitely not random?

4. What cycles of numbers do you observe?

In [0]:

In [0]:

Linear Congruential Method

This method generates a sequence of values $S_0, S_1, S_2, \ldots, S_M$ using modular arithmetic. First we select a multiplier α, an increment β, and a modulus N. We choose a starting value S_0 and iterate the rule:

$$S_n = \alpha S_{n-1} + \beta \pmod{N}$$

In [11]:
```
def linearCongruential(multiplier, increment, modulus, seed, numVals):
    # initializations
    vals = [0]*numVals
    vals[0] = seed

    # repeat numVals times
    for i in range(numVals - 1):
        # compute the next number and append it to the list
        next = ( multiplier*vals[i] + increment ) % modulus
        vals[i+1] = next

    # return the list
    return vals
```

In [14]:
```
print( linearCongruential(37, 1, 100, 17, 100) )
```

Out[14]:
```
[17, 30, 11, 8, 97, 90, 31, 48, 77, 50, 51, 88, 57, 10, 71, 28, 37, 70,
91, 68, 17, 30, 11, 8, 97, 90, 31, 48, 77, 50, 51, 88, 57, 10, 71, 28,
37, 70, 91, 68, 17, 30, 11, 8, 97, 90, 31, 48, 77, 50, 51, 88, 57, 10,
71, 28, 37, 70, 91, 68, 17, 30, 11, 8, 97, 90, 31, 48, 77, 50, 51, 88,
57, 10, 71, 28, 37, 70, 91, 68, 17, 30, 11, 8, 97, 90, 31, 48, 77, 50,
51, 88, 57, 10, 71, 28, 37, 70, 91, 68]
```

In [0]:

Questions to investigate

1. Experiment with the linear congruential method using various choices for the multiplier, increment, and modulus. What do you observe?
2. Do certain parameters produce sequences of digits that you would consider random?
3. Do certain parameters produce sequences of digits that you would consider definitely not random?
4. What cycles of numbers do you observe?

In [0]:

In [0]: