Kernel: SageMath 10.7

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(224892)
```

Out[2]: 6

In [3]:
```
# 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))
        #print(num)
        num = num % (10^k)
        print(num)

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

    return seq
```

In [4]:
```
middleSquare(4223, 10)
```

Out[4]:
```
8337
5055
5530
5809
7444
4131
651
4238
9606
2752
```

```
[4223, 8337, 5055, 5530, 5809, 7444, 4131, 651, 4238, 9606,
2752]
```

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 [5]:
```
# 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 [6]:
```
extractDigits([1234, 77, 12345, 9876])
```

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

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 [7]:
```
seq = middleSquare(4223, 20)
print(extractDigits(seq))
```

Out[7]:
```
8337
5055
5530
5809
7444
4131
651
4238
9606
2752
5735
8902
```

```
2456
319
1017
342
1169
3665
4322
6796
[4, 2, 2, 3, 8, 3, 3, 7, 5, 0, 5, 5, 5, 5, 3, 0, 5, 8, 0, 9,
7, 4, 4, 4, 4, 1, 3, 1, 0, 6, 5, 1, 4, 2, 3, 8, 9, 6, 0, 6, 2,
7, 5, 2, 5, 7, 3, 5, 8, 9, 0, 2, 2, 4, 5, 6, 0, 3, 1, 9, 1, 0,
1, 7, 0, 3, 4, 2, 1, 1, 6, 9, 3, 6, 6, 5, 4, 3, 2, 2, 6, 7, 9,
6]
```

In [11]:

```
seq = middleSquare(4671, 100)
print(extractDigits(seq))
```

Out[11]:

```
8182
9451
3214
3297
8702
7248
5335
4622
3628
1623
6341
2082
3347
2024
965
9312
7133
8796
3696
6604
6128
5523
5035
3512
3341
1622
6308
7908
5364
7724
6601
5732
8558
```

2393
7264
7656
6143
7364
2284
2166
6915
8172
7815
742
5505
3050
3025
1506
2680
1824
3269
6863
1007
140
196
384
1474
1726
9790
8441
2504
2700
2900
4100
8100
6100
2100
4100
8100
6100
2100
4100
8100
6100
2100
4100
8100
6100
2100
4100
8100
6100
2100
4100

```
8100
6100
2100
4100
8100
6100
2100
4100
8100
6100
2100
4100
8100
6100
2100
4100
[4, 6, 7, 1, 8, 1, 8, 2, 9, 4, 5, 1, 3, 2, 1, 4, 3, 2, 9, 7,
8, 7, 0, 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, 3, 3, 4, 7, 2, 0, 2, 4, 0, 9,
6, 5, 9, 3, 1, 2, 7, 1, 3, 3, 8, 7, 9, 6, 3, 6, 9, 6, 6, 6, 0,
4, 6, 1, 2, 8, 5, 5, 2, 3, 5, 0, 3, 5, 3, 5, 1, 2, 3, 3, 4, 1,
1, 6, 2, 2, 6, 3, 0, 8, 7, 9, 0, 8, 5, 3, 6, 4, 7, 7, 2, 4, 6,
6, 0, 1, 5, 7, 3, 2, 8, 5, 5, 8, 2, 3, 9, 3, 7, 2, 6, 4, 7, 6,
5, 6, 6, 1, 4, 3, 7, 3, 6, 4, 2, 2, 8, 4, 2, 1, 6, 6, 6, 9, 1,
5, 8, 1, 7, 2, 7, 8, 1, 5, 0, 7, 4, 2, 5, 5, 0, 5, 3, 0, 5, 0,
3, 0, 2, 5, 1, 5, 0, 6, 2, 6, 8, 0, 1, 8, 2, 4, 3, 2, 6, 9, 6,
8, 6, 3, 1, 0, 0, 7, 0, 1, 4, 0, 0, 1, 9, 6, 0, 3, 8, 4, 1, 4,
7, 4, 1, 7, 2, 6, 9, 7, 9, 0, 8, 4, 4, 1, 2, 5, 0, 4, 2, 7, 0,
0, 2, 9, 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, 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, 8, 1, 0, 0, 6, 1, 0, 0, 2, 1,
0, 0, 4, 1, 0, 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 [9]:

```
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
        vals[i+1] = (multiplier*vals[i] + increment) %
modulus

    # return the list
    return vals
```

In [10]:
```
linearCongruential(37, 1, 100, 17, 20)
```

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

In [ ]:

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 [ ]:

In [ ]: