Pseudo-random number generators

Contents


```
import numpy as np
np.set_printoptions(precision=3)
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set(rc={"figure.dpi":100, 'savefig.dpi':300})
sns.set_context('notebook')
sns.set_style("ticks")
```

Objectives

- To understand how computers generate “random” numbers.

Readings

- These notes.
- Middlesquare algorithm.
- Linear congruential generator.

Pseudo-random number generators (PRNG)

Random number generation is the backbone of Bayesian inference. Computers are deterministic. So, how can they generate random numbers? Well they cannot! But they can produce sequence of numbers that look like random numbers! These “fake” random number generators are called Pseudo-random number generators (PRNG). They are used to generate random numbers between zero and a maximum integer, say m. As we will argue later this is sufficient to generate pretty much any random variable you want.

The middlesquare algorithm (Von Neumann)

The middlesquare algorithm is the simplest PRNG.

1. Take a number and square it.
2. Pad the result with zeros to get to the desired number of digits.
3. Take the middle digits of the resulting number.
4. Repeat.

Here is an implementation:

```
def middlesquare(s : int, digits : int = 4):
    """Sample random numbers using the middle square algorithm.

    Arguments:
        s      -- The initial seed.
        digits -- How many digits do you want.
    """
    # Square the number
    s2 = s ** 2
    # Turn the resulting number into a string padding
    # with zeros to get to the desired number of digits
    s2_str = str(s2).zfill( 2 * digits)
    # Keep only the middle
    middle_str = s2_str[int(np.floor(digits/2)):][:-int(np.floor(digits/2))]
    return int(middle_str)
```

Pads so the resulting string is 8 characters long

Let's draw some random numbers:

```
seed = 1234
s = seed
for _ in range(20):
    s = middlesquare(s, digits=4)
    print(s)
```

```
5227
3215
3362
3030
1809
2724
4201
6484
422
1780
1684
8358
8561
2907
4506
3040
2416
8370
569
3237
```

Unfortunately, the middlesquare algorithms results in periodic sequences with very small period. For example:

```
seed = 540
s = seed
for _ in range(20):
    s = middlesquare(s, digits=4)
    print(s)
```

```
2916
5030
3009
540
2916
5030
3009
540
2916
5030
3009
540
2916
5030
3009
540
2916
5030
3009
540
```

Questions

- What is the minimum number that you can get from the middle square algorithm with 4 digits?
- What is the maximum number that you can get from the middle square algorithm with 4 digits?

Linear congruential generator (LCG)

The linear congruential generator works as follows. You pick three big integers a, b and m. Pick a seed x_0. Then iterate:

$$x_{i+1} = (ax_i + b) \mod m$$

Here is a simple implementation:

```
def lcg(
    x : int,
    a : int = 123456,
    b : int = 978564,
    m : int = 6012119
):
    """Sample random numbers using a linear congruential generator.

    Arguments:
        x     -   The previous number in the sequence.
        a     -   A big integer.
        b     -   Another big integer.
        m     -   Another big integer.
    """
    return (a * x + b) % m
```

Let's draw some random numbers:

```
seed = 1234
s = seed
for _ in range(20):
    s = lcg(s)
    print(s)
```

```
3020293
2650792
5494308
965075
3115541
1883116
317849
243995
2909094
134725
4067010
1658958
451558
4155644
2001482
3861575
4605659
1061643
2982572
5159241
```

Questions

- What is the minimum number that you can get from LCG with $m = 10$?
- What is the maximum number that you can get from LCG with $m = 10$?
- What about the general case of LCG with arbitrary m?
- Picking $a = 2$ and $b = 0$ is a bad choice. But let's do it. Pick also $m = 10$ and go ahead and play with the algorithm. See for yourself that these numbers must be very big and ideally prime numbers.

Mersenne Twister PRNG

Numpy uses the Mersenne Twister to generate random numbers. Its details are more complicated than LCG, but it is still initialized by an integer seed. You can test it as follows:

integer seed. You can test it as follows:

```
np.random.seed(12345)

for _ in range(5):
    print(np.random.randint(0, 6012119))
```

```
1396132
2993577
1134974
5664101
3555874
```

If you rerun the code above, you will get a different set of random numbers:

```
for _ in range(5):
    print(np.random.randint(0, 6012119))
```

```
5290753
4246897
3579195
3692649
3755099
```

But if you refix the seed, you will get exactly the same sequence as the first time:

```
np.random.seed(12345)

for _ in range(5):
    print(np.random.randint(0, 6012119))
```

```
1396132
2993577
1134974
5664101
3555874
```

So, resetting the seed gives you the same sequence. In your numerical simulations you should always set the seed by hand in order to ensure the reproducibility of your work.

Questions

- What is the maximum number that you can get from the Mersenne Twister PRNG? Hint: Google it.

By Ilias Bilionis (ibilion[at]purdue.edu)