Program Design

Invasion Percolation: Randomness

Copyright © Software Carpentry 2010
This work is licensed under the Creative Commons Attribution License
See http://software-carpentry.org/license.html for more information.
Need a 2D grid of random values
Need a 2D grid of random values
Uniformly distributed
in some range 1..Z

<table>
<thead>
<tr>
<th>5</th>
<th>3</th>
<th>7</th>
<th>2</th>
<th>6</th>
<th>1</th>
<th>1</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>9</td>
<td>3</td>
<td>9</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>5</td>
<td>9</td>
<td>7</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>8</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>8</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1</td>
<td>9</td>
<td>7</td>
<td>8</td>
<td>6</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>
Need a 2D grid of random values
Uniformly distributed in some range 1..Z
Need to check the science on that...

<table>
<thead>
<tr>
<th>5</th>
<th>3</th>
<th>7</th>
<th>2</th>
<th>6</th>
<th>1</th>
<th>1</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>9</td>
<td>3</td>
<td>9</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>5</td>
<td>9</td>
<td>7</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>8</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>8</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1</td>
<td>9</td>
<td>7</td>
<td>8</td>
<td>6</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>
assert N > 0, "Grid size must be positive"
assert N%2 == 1, "Grid size must be odd"
grid = []
for x in range(N):
    grid.append([])
    for y in range(N):
        grid[-1].append(1)
from random import seed, randint
assert N > 0, "Grid size must be positive"
assert N%2 == 1, "Grid size must be odd"
assert Z > 0, "Range must be positive"
seed(S)
grid = []
for x in range(N):
    grid.append([])
    for y in range(N):
        grid[-1].append(randint(1, Z))
from random import seed, randint
assert N > 0, "Grid size must be positive"
assert N%2 == 1, "Grid size must be odd"
assert Z > 0, "Range must be positive"
seed(S)
grid = []
for x in range(N):
    grid.append([])
    for y in range(N):
        grid[-1].append(randint(1, Z))
>>> from random import seed, randint
>>> seed(4713983)
>>> for i in range(5):
    ...   print randint(1, 10),
    ...
    7 2 6 6 5
>>> from random import seed, randint
>>> seed(4713983)
>>> for i in range(5):
...   print randint(1, 10),
...
7 2 6 6 5

Standard Python
random number library
>>> from random import seed, randint

>>> seed(4713983)

>>> for i in range(5):
    ...   print randint(1, 10),
    ...

7 2 6 6 5

Initialize the sequence of “random” numbers
```python
>>> from random import seed, randint
>>> seed(4713983)
>>> for i in range(5):
    ...     print randint(1, 10),
...     print
7 2 6 6 5
```

Produce the next “random” number in the sequence.
Here’s a simple “random” number generator:

```python
>>> base = 17  # a prime
>>> value = 4  # anything in 0..base-1
>>> for i in range(20):
...   value = (3 * value + 5) % base
...   print value,
...   print value,
...
0 5 3 14 13 10 1 8 12 7 9 15 16 2 11 4 0 5 3 14
```
>>> base = 17  # a prime
>>> value = 4  # anything in 0..base-1
>>> for i in range(20):
...     value = (3 * value + 5) % base
...     print value,
...
0 5 3 14 13 10 1 8 12 7 9 15 16 2 11 4 0 5 3 14

base controls how long before values repeat
>>> base = 17  # a prime
>>> value = 4  # anything in 0..base-1
>>> for i in range(20):
...    value = (3 * value + 5) % base
...    print value, ...
0 5 3 14 13 10 1 8 12 7 9 15 16 2 11 4 0 5 3 14

base controls how long before values repeat

Once they do, values appear in exactly the same order as before
>>> base = 17  # a prime
>>> value = 4  # anything in 0..base-1
>>> for i in range(20):
...   value = (3 * value + 5) % base
...   print value,
...   print value,

0 5 3 14 13 10 1 8 12 7 9 15 16 2 11 4 0 5 3 14

The *seed* controls where the sequence starts
>>> base = 17  # a prime
>>> value = 9  # anything in 0..base-1
>>> for i in range(20):
...   value = (3 * value + 5) % base
...   print value,
...
15 16 2 11 4 0 5 3 14 13 10 1 8 12 7 9 15 16 2 11

The seed controls where the sequence starts
Changing the seed slides values left or right
```python
>>> base = 17  # a prime
>>> value = 9  # anything in 0..base-1
>>> for i in range(20):
...   value = (3 * value + 5) % base
...   print value,
...
15 16 2 11 4 0 5 3 14 13 10 1 8 12 7 9 15 16 2 11

The seed controls where the sequence starts
Changing the seed slides values left or right
(We’ll use this fact when testing our program)
```
This is a lousy “random” number generator
This is a lousy “random” number generator

Did you notice that 6 never appeared?
This is a lousy “random” number generator
Did you notice that 6 never appeared?
That would probably distort our results...
What happens when 6 *does* appear?

```python
>>> base = 17
>>> value = 6
>>> for i in range(20):
...   value = (3 * value + 5) % base
...   print value,
... 
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
```
What happens when 6 \textit{does} appear?

```python
>>> base = 17
>>> value = 6
>>> for i in range(20):
...   value = (3 * value + 5) % base
...   print value,
...
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
```

How can we prove this won’t ever happen?
What happens when 6 *does* appear?

```python
>>> base = 17
>>> value = 6
>>> for i in range(20):
...     value = (3 * value + 5) % base
...     print value,
...     print
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
```

How can we prove this won't ever happen?

Or that something subtler won't go wrong?
Computers can’t generate real random numbers
Computers can’t generate real random numbers

But they *can* generate numbers with many of the same statistical properties as the real thing.
Computers can’t generate real random numbers
But they *can* generate numbers with many of
the same statistical properties as the real thing

This is very hard to get right
Computers can’t generate real random numbers
But they *can* generate numbers with many of the same statistical properties as the real thing
This is very hard to get right

*Never* try to do it yourself
Computers can’t generate real random numbers
But they *can* generate numbers with many of
the same statistical properties as the real thing
This is very hard to get right
*Never* try to do it yourself

*Always* use a good library
Computers can’t generate real random numbers
But they *can* generate numbers with many of
the same statistical properties as the real thing
This is very hard to get right
*Never* try to do it yourself
*Always* use a good library
...like Python’s
Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin. For, as has been pointed out several times, there is no such thing as a random number. There are only methods to produce random numbers, and a strict arithmetic procedure of course is not such a method.

– John von Neumann