Investigating Coding Theory Using Python

Jose Unpingco, PhD
junpingco@hpti.com
Half the job is finding the problem.
The purpose of communication?

Uncertainty (entropy)

message

Uncertainty (entropy)

sender

receiver
The Monty Hall problem is a probability puzzle based on the American television game show *Let's Make a Deal*. The name comes from the show's host, Monty Hall. The problem is also called the Monty Hall paradox, as it is a veridical paradox in that the result appears absurd but is demonstrably true.

A well-known statement of the problem was published in *Parade* magazine:

Suppose you're on a game show, and you're given the choice of three doors: Behind one door is a car; behind the others, goats. You pick a door, say No. 1, and the host, who knows what's behind the doors, opens another door, say No. 3, which has a goat. He then says to you, "Do you want to pick door No. 2?" Is it to your advantage to switch your choice?  
(Whitaker 1990)
How has the information environment changed?

Should you change your guess or not?

In search of a new car, the player picks a door, say 1. The game host then opens one of the other doors, say 3, to reveal a goat and offers to let the player pick door 2 instead of door 1.
Plot Summary for
"Star Trek" The Galileo Seven (1967)

A shuttle craft under Mr. Spock's command is forced to land on a hostile planet. His emotionless approach to command does not sit well with some crew members, particularly Mr. Boma who challenges Spock at every opportunity. The Enterprise and Captain Kirk meanwhile have only a short time to find the lost shuttle craft as they must deliver urgent medical supplies to Markus III in only a few days. Written by garykmcd
What information must be sent?

Interference environment
Repetition code

11111

Interference

10101

contains errors
What errors can I recover from?

```
p = [ "\{0:05b\}".format(i) for i in range(2**5) ]
pp=map(lambda x: map(eval,list(x)),p)
```
What errors can I recover from?

11111 errors 10001

11111 errors 11001

def \texttt{ham}(x,y): return sum(map(operator.ne, x,y))
Within 1 hamming distance unit

for i in p if ham(i, p[-1]) <= 1
for i in p if ham(i, p[0]) <= 1
Within 2 hamming distance unit

for i in p if ham(i,p[-1])<=2
for i in p if ham(i,p[0])<=2
Binary Repetition Codes

11111
00000

Transmit 0 or 1

\[
\begin{align*}
    i_1 + p_1 &= 0 \\
    i_1 + p_2 &= 0 \\
    i_1 + p_3 &= 0 \\
    \vdots & \quad \vdots \\
    i_1 + p_{n-1} &= 0
\end{align*}
\]

def chk(x):
    return [(x[0]+x[1])%2, (x[0]+x[2])%2, (x[0]+x[3])%2, (x[0]+x[4])%2]
Binary Repetition Codes (5,1)

In [480]: [(i,chk(i),sum(i)) for i in pp if sum(i)<=2]
Out[480]:
([(0, 0, 0, 0, 0), (0, 0, 0, 0), 0),
 (0, 0, 0, 0, 1), (0, 0, 0, 1), 1),
 (0, 0, 0, 1, 0), (0, 0, 1, 0), 1),
 (0, 0, 1, 0, 1), (0, 0, 1, 1), 2),
 (0, 0, 0, 1, 0), (0, 0, 1, 0), 1),
 (0, 0, 1, 0, 1), (0, 0, 1, 1), 2),
 (0, 0, 1, 1, 0), (0, 1, 0, 1), 2),
 (0, 0, 1, 1, 1), (0, 1, 1, 0), 2),
 (0, 0, 1, 0, 0), (0, 0, 0, 0), 0),
 (1, 0, 0, 0, 1), (1, 0, 0, 1), 2),
 (0, 0, 1, 0, 1), (0, 1, 0, 0), 2),
 (0, 0, 1, 1, 0), (0, 1, 0, 0), 2),
 (0, 1, 0, 0, 1), (0, 0, 0, 1), 2),
 (0, 1, 1, 0, 0), (0, 1, 0, 0), 2),
 (1, 0, 0, 0, 1), (1, 0, 0, 1), 1),
 (1, 0, 0, 1, 0), (1, 0, 1, 0), 2),
 (1, 0, 1, 0, 0), (1, 1, 0, 0), 2),
 (1, 0, 1, 0, 1), (1, 0, 1, 1), 2),
 (1, 0, 1, 1, 0), (1, 1, 0, 0), 2),
 (1, 1, 0, 0, 0), (0, 1, 1, 1), 2),
 (1, 1, 1, 0, 0), (0, 1, 1, 1), 2)]
Make code corrections

```python
k = dict([(tuple(chk(i)), i) for i in pp if sum(i) <= 2])

def fix(x):
    return [(i+j)%2 for i, j in zip(k[tuple(chk(x))], x)]
```

In [480]: [(i, chk(i), sum(i)) for i in pp if sum(i) <= 2]
Out[480]:

```python
[[[0, 0, 0, 0, 0], [0, 0, 0, 0, 1], [0, 0, 0, 1, 1], [0, 0, 0, 1, 0], [0, 0, 1, 0, 0], [0, 0, 1, 0, 1], [0, 0, 1, 1, 0], [0, 0, 1, 1, 1], [0, 1, 0, 0, 0], [0, 1, 0, 0, 1], [0, 1, 0, 1, 0], [0, 1, 0, 1, 1], [0, 1, 1, 0, 0], [0, 1, 1, 0, 1], [0, 1, 1, 1, 0], [0, 1, 1, 1, 1], [1, 0, 0, 0, 0], [1, 0, 0, 0, 1], [1, 0, 0, 1, 0], [1, 0, 0, 1, 1], [1, 0, 1, 0, 0], [1, 0, 1, 0, 1], [1, 0, 1, 1, 0], [1, 0, 1, 1, 1], [1, 1, 0, 0, 0], [1, 1, 0, 0, 1], [1, 1, 0, 1, 0], [1, 1, 0, 1, 1], [1, 1, 1, 0, 0], [1, 1, 1, 0, 1], [1, 1, 1, 1, 0], [1, 1, 1, 1, 1]]
```
Applying corrections

In [48]: for i in pp:
    ....:     (i, fix(i))
    ....:

Out[48]: ([0, 0, 0, 0, 0], [0, 0, 0, 0, 0])
Out[48]: ([0, 0, 0, 0, 1], [0, 0, 0, 0, 0])
Out[48]: ([0, 0, 0, 1, 0], [0, 0, 0, 0, 0])
Out[48]: ([0, 0, 0, 1, 1], [0, 0, 0, 0, 0])
Out[48]: ([0, 0, 1, 0, 0], [0, 0, 0, 0, 0])
Out[48]: ([0, 0, 1, 0, 1], [0, 0, 0, 0, 0])
Out[48]: ([0, 0, 1, 1, 0], [0, 0, 0, 0, 0])
Out[48]: ([0, 0, 1, 1, 1], [1, 1, 1, 1, 1])
Out[48]: ([0, 1, 0, 0, 0], [0, 0, 0, 0, 0])
Out[48]: ([0, 1, 0, 0, 1], [0, 0, 0, 0, 0])
Out[48]: ([0, 1, 0, 1, 0], [0, 0, 0, 0, 0])
Out[48]: ([0, 1, 0, 1, 1], [1, 1, 1, 1, 1])
Out[48]: ([0, 1, 1, 0, 0], [0, 0, 0, 0, 0])
Out[48]: ([0, 1, 1, 0, 1], [0, 0, 0, 0, 0])
Out[48]: ([0, 1, 1, 1, 0], [0, 0, 0, 0, 0])
Out[48]: ([0, 1, 1, 1, 1], [1, 1, 1, 1, 1])
(7, 4) Binary Hamming Code

Figure 5.0.2 Venn diagram illustration of (7, 4) binary Hamming code.
(7,4) Binary Hamming Code

\[ p_1 + i_1 + i_2 + i_3 = 0, \]
\[ p_2 + i_2 + i_3 + i_4 = 0, \]
\[ p_3 + i_1 + i_2 + i_4 = 0, \]

\[ i_1 i_2 i_3 i_4 p_1 p_2 p_3 \]
(7,4) Binary Hamming Code

In [361]: p = ['\{0:07b\}'.format(i) for i in range(2**7)]

In [362]: pp=map(lambda x: map(eval,list(x)),p)

In [363]: [(i,chk(i),sum(chk(i))) for i in pp]
Out[363]:
[([0, 0, 0, 0, 0, 0, 0], [0, 0, 0], 0),
 ([0, 0, 0, 0, 0, 0, 1], [0, 0, 1], 1),
 ([0, 0, 0, 0, 0, 1, 0], [0, 1, 0], 1),
 ([0, 0, 0, 0, 0, 1, 1], [0, 1, 1], 2),
 ([0, 0, 0, 0, 1, 0, 0], [1, 0, 0], 1),
 ([0, 0, 0, 0, 1, 0, 1], [1, 0, 1], 2),
 ([0, 0, 0, 0, 1, 1, 0], [1, 1, 0], 2),
 ([0, 0, 0, 0, 1, 1, 1], [1, 1, 1], 3),]
(7,4) Binary Hamming Code

In [372]: [(i,chk(i),sum(chk(i))) for i in pp if sum(chk(i))==0]

Out[372]:

[[[0, 0, 0, 0, 0, 0, 0], [0, 0, 0], 0),
 ([0, 0, 0, 1, 0, 1, 1], [0, 0, 0], 0),
 ([0, 0, 1, 0, 1, 1, 0], [0, 0, 0], 0),
 ([0, 0, 1, 1, 1, 0, 1], [0, 0, 0], 0),
 ([0, 1, 0, 0, 1, 1, 1], [0, 0, 0], 0),
 ([0, 1, 0, 1, 1, 0, 0], [0, 0, 0], 0),
 ([0, 1, 1, 0, 0, 0, 1], [0, 0, 0], 0),
 ([0, 1, 1, 1, 0, 1, 0], [0, 0, 0], 0),
 ([1, 0, 0, 0, 1, 0, 1], [0, 0, 0], 0),
 ([1, 0, 0, 1, 1, 1, 0], [0, 0, 0], 0),
 ([1, 0, 1, 0, 0, 1, 1], [0, 0, 0], 0),
 ([1, 0, 1, 1, 0, 0, 0], [0, 0, 0], 0),
 ([1, 1, 0, 0, 0, 1, 0], [0, 0, 0], 0),
 ([1, 1, 0, 1, 0, 0, 1], [0, 0, 0], 0),
 ([1, 1, 1, 0, 0, 1, 0], [0, 0, 0], 0),
 ([1, 1, 1, 1, 0, 0, 0], [0, 0, 0], 0),
 ([1, 1, 1, 1, 1, 1, 1], [0, 0, 0], 0)]
(7,4) Binary Hamming Code

In [387]: [(i,chk(i),sum(chk(i))) for i in pp if sum(i)==1]
Out[387]:
([(0, 0, 0, 0, 0, 0, 1], [0, 0, 1], 1),
 ([0, 0, 0, 0, 0, 1, 0], [0, 1, 0], 1),
 ([0, 0, 0, 0, 1, 0, 0], [1, 0, 0], 1),
 ([0, 0, 0, 1, 0, 0, 0], [0, 1, 1], 2),
 ([0, 0, 1, 0, 0, 0, 0], [1, 1, 0], 2),
 ([0, 1, 0, 0, 0, 0, 0], [1, 1, 1], 3),
 ([1, 0, 0, 0, 0, 0, 0], [1, 0, 1], 2)"
In [402]: [[(i,chk(i),sum(i)) for i in pp if sum(i)==2]
Out[402]:
[([0, 0, 0, 0, 1, 1], [0, 1, 1], 2),
 ([0, 0, 0, 0, 1, 0, 1], [1, 0, 1], 2),
 ([0, 0, 0, 0, 1, 1, 0], [1, 1, 0], 2),
 ([0, 0, 0, 1, 0, 1, 0], [0, 1, 0], 2),
 ([0, 0, 0, 1, 1, 0, 0], [1, 1, 1], 2),
 ([0, 0, 1, 0, 0, 0, 1], [1, 1, 1], 2),
 ([0, 0, 1, 0, 0, 1, 0], [1, 1, 0], 2),
 ([0, 0, 1, 0, 0, 1, 0], [1, 0, 0], 2),
 ([0, 0, 1, 0, 1, 0, 0], [0, 1, 0], 2),
 ([0, 0, 1, 0, 1, 0, 0], [0, 1, 0], 2),
 ([0, 0, 1, 0, 0, 0], [1, 0, 1], 2),
 ([0, 1, 0, 1, 0, 0], [1, 1, 1], 2),
 ([0, 1, 0, 0, 0, 1, 0], [1, 1, 0], 2),
 ([0, 1, 0, 0, 0, 1, 0], [1, 0, 1], 2),
 ([0, 1, 0, 0, 1, 0, 0], [0, 1, 1], 2),
 ([0, 1, 0, 0, 0, 0], [1, 0, 0], 2),
 ([0, 1, 1, 0, 0, 0], [0, 1, 0], 2),
 ([1, 0, 0, 0, 0, 1], [1, 0, 0], 2),
 ([1, 0, 0, 0, 0, 1], [1, 1, 0], 2),
 ([1, 0, 0, 1, 0, 0], [0, 0, 1], 2),
 ([1, 0, 0, 0, 0, 0], [1, 1, 0], 2),
 ([1, 0, 1, 0, 0, 0], [0, 1, 1], 2),
 ([1, 1, 0, 0, 0, 0], [0, 1, 0], 2),
 ([1, 1, 0, 0, 0, 0], [0, 0, 1], 2),
 ([1, 1, 0, 0, 0, 0], [0, 1, 0], 2)]
(7,4) Binary Hamming Code
Python Iterators

An **iterator** is an object representing a stream of data

```python
>>> L = [1,2,3]
>>> it = iter(L)
>>> print(it)
<iterator object at 0x8116870>
>>> it.next()
1
>>> it.next()
2
>>> it.next()
3
>>> it.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
StopIteration
```
class Count(object):
    
def __init__(self):
        self.x = 10

    def __iter__(self):
        return self

    def next(self):
        print 'am in next'
        self.x -= 1

        if self.x < 0:
            raise StopIteration
        else:
            return self.x
Python List Comprehensions

>>> seq1 = 'abc'

>>> seq2 = (1, 2, 3)

>>> [(x, y) for x in seq1 for y in seq2]
[('a', 1), ('a', 2), ('a', 3), ('b', 1), ('b', 2), ('b', 3), ('c', 1), ('c', 2), ('c', 3)]
Generators

- Produces a stream of on-demand values
- Only executes on `next()`
- `yield()` produces a value, but saves the function’s state for later
- Consumable (use once-only)

```python
def generate_ints(N):
    for i in range(N):
        yield i

gen=( i for i in range(3) )
```
Generators allow non-linear looping

For loop

Loop statements
generator

See itertools

Loop statements
other code

More Loop statements
Generator: Permutation Example

```python
>>> n = 0
>>> for i in itertools.permutations(range(10)):
    n += 1

>>> scipy.factorial(10)
array(3628800)

>>> scipy.factorial(10)*4
14515200
```

Each element generated on-demand, not stored in a list (would not fit!)
HPCMP Functional Areas (CTAs)

Climate/Weather/Ocean Modeling (CWO)
Computational Chemistry and Materials Science (CCM)
Computational Electromagnetics and Acoustics (CEA)
Computational Electronics and Nanoelectronics (CEN)
Computational Fluid Dynamics (CFD)
Computational Structural Mechanics (CSM)
Environmental Quality Modeling and Simulation (EQM)
Forces Modeling and Simulation (FMS)
Integrated Modeling and Test Environments (IMT)
Signal/Image Processing (SIP)
*From the “System Engineering Fundamentals” book from the Defense Acquisition University press, 2001*
90% costs defined by requirements.

Minimize Risk!