COMP 312 Assignment 2

Daniel Braithwaite

March 14, 2016

1 Python

1.1 Dice Problem

1.1.1 Part A

```
import random
random.seed(123)


def roll_die(n):
    """Rolls n die and returns the results as array"""

    rolls = [];
    for i in range(0, n):
        rolls.append(random.randint(1, 6))

    return rolls;

if __name__ == '__main__':
    n = 1000000
    events = 0

    for i in range(0, n):
        o = False


        for j in range(0, 24):
            roll = roll_die(2)

            if roll[0] == 6 and roll[1] == 6:
                o = True

        if o:
            events += 1

    p = float(events)/float(n)

    print p
```

The output of this program is 0.491206 so yes the probability of the event occurring is less than $\frac{1}{2}$. The value of n was very large so we can be confident in this finding.

1.1.2 Part B

```
import random
random.seed(123)
```

```python
def roll_die(n):
	"""Rolls n die and returns the results as array"""

	rolls = [];
	for i in range(0, n):
		rolls.append(random.randint(1, 6))

	return rolls;

if __name__ == '__main__':
	n = 1000000
	min_num_throws = 24
	max_num_throws = 26
	events = {}

	for i in range(0, n):

		for j in range(min_num_throws, max_num_throws + 1):

			o = False
			events.setdefault(j, 0)

			for k in range(0, j):
				roll = roll_die(2)

				if roll[0] == 6 and roll[1] == 6:
					o = True

			if o:
				events[j] += 1

	for i in range(min_num_throws, max_num_throws+1):
		events[i] = float(events[i])/float(n)

	print events
```

The modified code calculates the probability of the event occurring with a different number of throws. Over the range 24 to 26 it outputs the the following $\{24 : 0.491443, 25 : 0.506207, 26 : 0.518883\}$
From this we see that it requires 25 throws for the probability to be $> \frac{1}{2}$ but as close to $\frac{1}{2}$ as possible

1.2 Optimized Table Look-up

```python
import random
```

```python
def tablelookup(y,p):
        """Sample from y[i] with probabilities p[i]"""
        u = random.random()
        sumP = 0.0
        b = 0

        for i in range(len(p)):
                sumP += p[i]
                b += 1
                if u < sumP:
                        return {'val': y[i], 'count': b}

def run_lookup(y, p):
        """Runs a lookup for given probabilities"""

        m = 1000000
        b = 0.0
        valuetotal = 0.0

        for k in range(m):
                d = tablelookup(y,p)

                b += d['count']
                valuetotal += d['val']

        print valuetotal/m
        print b/m
        print "\n"

random.seed(123)

y1 = [0,1,2,3,4,5]
p1 = [1.0/1024, 15.0/1024, 90.0/1024, 270.0/1024, 405.0/1024, 243.0/1024]

y2 = [5,4,3,2,1,0]
p2 = [243.0/1024, 405.0/1024, 270.0/1024, 90.0/1024, 15.0/1024, 1.0/1024]

y3 = [4,3,5,2,1,0]
p3 = [405.0/1024, 270.0/1024, 243.0/1024, 90.0/1024, 15.0/1024, 1.0/1024]


print "Part I"
run_lookup(y1, p1)

print "Part II"
```

```
run_lookup(y2, p2)

print "Part III"
run_lookup(y3, p3)
```

The output from running this code is

Part I	Part II	Part III
3.750354	3.750114	3.750704
4.750354	2.249886	2.064436

As we can see from this all results are mostly the same. We also see that option 2 and 3 take about half as many steps as option 1. However option 3 is just slightly faster

2 Queues

2.1 Problem 3

We know that the service times are distributed exponentially. And we assume that all the servers have equal average service times, let this time be m. Once one of the other services has finished you can begin being served (you are now 'racing' against the remaining 6 people). As the exponential distribution has the property of being memory-less all the services have the same probability of taking time m. This means that $\frac{6}{7}$ probability of finishing before atleast one of the other customers being served.

2.2 Problem 4

state $\leftarrow$ number of active pizza shops

For this problem I used python to compute the steady state vector. The following code is what I used to do this.

```
coeffs = {}

# Compute the coefficients
l = 1
u = 1

for i in range(1, 32 + 1):
        price = max(0, 16 - 0.5 * (i-1))

        l = l * price
        u = u * (i * 1.0/(10 + price))

        coeffs[i] = float(l)/float(u)
```

```python
# Compute pi 0
pi0 = 0
s = 1

for i in range(1, 32 + 1):
        s = s + coeffs[i]

pi0 = 1/s

pies = {}
pies[0] = pi0


# Compute the rest of the pis
for i in range(1, 32 + 1):
        pies[i] = coeffs[i] * pi0


# Ensure that the sum of pies is 1
piSum = 0
for i in range(32 + 1):
        piSum += pies[i]

if not piSum == 1.0:
        raise Exception("Sum of pies should be 1")


## Solve Assignment Problems ##
###############################

# Compute average number of pizza restrants
avg = 0
for i in range(32 + 1):
        avg += i * pies[i]

print "Average Num Pizza Shops: " + str(avg)

# Compute fraction of time more than 20 restrants
timeM20 = 0.0
for i in range(20 + 1, 32 + 1):
        timeM20 += pies[i]

print "Fraction of time with more than 20 restrants: " + str(timeM20)
```

To get the average state of the system we take the following sum

$$\sum_{n=0}^{32} n\pi_n$$

for which we get the following result

$$Avg\ number\ of\ pizza\ shops:\ 27.6082698882$$

To get the fraction of time spent in state i we know this value is π_i so if we want to find the fraction of time with 20 or more restaurants this is

$$\sum_{n=20}^{32} \pi_n = 0.999741852504 \tag{1}$$

$$Fraction\ of\ time\ with\ >\ 20\ restrants:\ 0.999741852504 \tag{2}$$