Reinforcement Learning: Homework #3

Due on April 16, 2020 at 11:59pm

Professor Ziyu Shao

Yiwei Yang
2018533218

Problem 1

Problems 3 For Lemma

1. First we have $Y_n = 1$, which means that from n_{th} to $(n+1)_{th}$ has its state changed. So because there's only 2 state, we can list the possibility as $X_n = A \bigwedge X_{n+1} = B$ and $X_n = B \bigwedge X_{n+1} = B$. Then notice that given $Y_{n-1} = 1$, we have $X_{n-1} = B$, the future state Y_{n+1} and X_{n+1} are impacted both. So, we can say that, the future is affetcted by the previous state. So Y_n isn't Markov Chain.
2. **Proofs by Induction and Contradiction:** Suppose Z_n is not Markov Chain for $m \in \mathbb{N}$. First, we have the base condition $m = 1$ is true since Y_n is (X_n, X_{n+1}) by the previous proof. Also, for $m+1$ $(Z_n)_n$ isn't a Markov Chain. So, we get that $Z_n = \{the\ (n-m+1)_{st}\ to\ n_{th}\ terms\ of\ the\ Y_n\ chain\}$

Problems 2

1. (a) From the question we can get that $\forall n \in \mathbb{N}$, we have the marginal distribution of the X_n, which is s. We can write the distribution as an expected value: $p_X(x) = \int_s p_{X|S}(x|s)p_S(s)ds$, since we have that X_0 s and that the stationary distribution for the chain.

   (b) Denote the n_{th} variable of s is s_n. By the linearity of expectation, we have the average number of the variables that are valued in 3 is $10s_3$, considering 10 variables.

2. Given $X_n \in \{1, 2, 3\}$ and corresponding Y_n. we have symmetric input:

$$Q = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & 1 & 0 \end{bmatrix}$$

   When $X_n = 1$, we have $Y_n = 0$. When $X_n = 2$, we have $Y_n = 0$. When $X_n = 3$, we have $Y_n = 2$. For state of 1 and 2, they can be merged into mutual state zero. Also, state 3 is transfromed in 2. By *3(b)*, we have $\forall i, j \in \mathbb{N}, q_i j = \frac{1}{3}$.

   The above input Q can get the result $Y_1 = 0, Y_2 = 2$, which means that the future state relies on the zero state. So, Y_n do not have the markov property.

Problems 4

1. First, given the graph, we have $\forall n \in \mathbb{N}$ $P(X_{n+1} = 1|X_n = 1) = P(X_{n+1} = 2|X_n = 2)p$ and $P(X_{n+1} = 1|X_n = 2) = P(X_{n+1} = 2|X_n = 1) = 1 - p$. Therefore, we have:

$$Q = \begin{bmatrix} p & 1-p \\ 1-p & p \end{bmatrix}$$

2. Let's introduce a stationary distribution $\pi = \begin{bmatrix} \pi_1 \\ \pi_2 \end{bmatrix}$, a probability distribution that remains unchanged in the Markov chain. By defination, we have $\pi = \pi Q$, amd π is the invariant by the matrix Q. Take Q in the equation $(I - Q)\pi = 0$, we have

$$\begin{bmatrix} p & 1-p \\ 1-p & p \end{bmatrix} \pi = 0$$

$$\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \pi = 0$$

$$\pi_1 - \pi_2 = 0$$

   We assume the distribution is normalized, so we have $\pi_1 + \pi_2 = 1$, so we have the stationary distribution $\pi = \begin{bmatrix} \pi_1 \\ \pi_2 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \end{bmatrix}$

Problem 1 continued on next page...

3. Let $n \in \mathbb{N}$ be arbitary. Suppose $p_n \in (0,1)$,we have the property that

$$Q^n = \begin{bmatrix} p_n & 1-p_n \\ 1-p_n & p_n \end{bmatrix}$$

$$\begin{bmatrix} p_{n+1} & 1-p_{n+1} \\ 1-p_{n+1} & p_{n+1} \end{bmatrix} = Q^{n+1} = Q^n Q = \begin{bmatrix} p_n & 1-p_n \\ 1-p_n & p_n \end{bmatrix} \begin{bmatrix} p & 1-p \\ 1-p & p \end{bmatrix}$$

$$\begin{cases} p_{n+1} = p_n p + (1-p_n)(1-p) \\ 1-p_{n+1} = p_n(1-p) + (1-p_n)p \\ 1-p_n = (1-p_{n+1})p + p_n(1-p) \\ p_{n+1} = (1-p_n)(1-p) + p_n p \end{cases}$$

$$p_{n+1} - (2p-1)p_n = 1-p$$

By the thereom of the characteristic equation, suppose the final recursive is $p_{n+1} = p_n^H + p_n^P$. We have the characteristic polynomial is $\lambda - (2p-1) = 0$ and the particular solution can be written as $p_n^P = \frac{1}{2}$. So $p_n = A(2p-1)^{n+1}$

To compute A, we have to apply the convergence properties as $n \to \infty$, p_n is convergent. $lim_{n\to\infty} p_n = lim_{n\to\infty}(A(2p-1)^n + \frac{1}{2}) = Alim_{n\to\infty}(2p-1)^n + \frac{1}{2} = \frac{1}{2}$ So, we have $lim_{n\to\infty} Q^n = lim_{n\to\infty} \begin{bmatrix} p_n & 1-p_n \\ 1-p_n & p_n \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}$

Problems 6

1. s is the stationary distribution for chains three chains. As we've considered in *2(2)*, s is the marginal distribution for $(X_n)_n, (Y_n)_n, (Z_n)_n$. So, the possibility of some Drogon at their home is s_0. Applying the linearity property, we have that the average number of times of Drogon visiting home is $25s_0$.
2. $(W_n)_n$ is a Markov chain and here's the proof.
   (a) All we need is to prove its markoc property. $\forall n \in \mathbb{N}, x_n, y_n, z_n \in M$ $P((X_{n+1}, Y_{n+1}, Z_{n+1}) = (x_{n+1}, y_{n+1}, z_{n+1}) | \{(X_0, Y_0, Z_0) = (x_0, y_0, z_0), (X_1, Y_1, Z_1) = (x_1, y_1, z_1), (X_2, Y_2, Z_2) = (x_2, y_2, z_2)$ $, ...(X_n, Y_n, Z_n) = (x_n, y_n, z_n)\}$)(Denote the $RHS = Ans$,$\{(X_0, Y_0, Z_0) = (x_0, y_0, z_0), (X_1, Y_1, Z_1) = (x_1, y_1, z_1), (X_2, Y_2, Z_2) = (x_2, y_2, z_2), ...(X_n, Y_n, Z_n) = (x_n, y_n, z_n)\} = A_n$)
   (b) By bayesian theory, we have $Ans = P(X_{n+1} = x_{n+1}|A_n)P(Y_{n+1} = y_{n+1}|X_{n+1} = x_{n+1}, A_n)P(Z_{n+1} = z_{n+1}|X_{n+1} = x_{n+1}, Y_{n+1} = y_{n+1}, A_n)$
   (c) By LOTE and MDP, we have $Ans = P(X_{n+1} = x_{n+1}|X_n = x_n)P(Y_{n+1} = y_{n+1}|Y_n = y_n)P(Z_{n+1} = z_{n+1}|Z_n = z_n) = LHS$
3. Given that 3 dragons start at home at time 0, we have they have an average time $\frac{1}{s_0}$ to get home again. Applyiing the independency of three dragons' momvements, the expectaiton time wil be $\frac{1}{s_0^3}$.

Problems 8

1. First, denote M be the state space. Denote $q_{i,j} = 0$ when there's no edge between the i and the j, $\forall\, different\, i, j \in M$. Suppose there's an edge, we have $q_{i,j} = \frac{1}{d_i} min(\frac{d_i}{d_j}, 1) = \begin{cases} \frac{1}{d_i} & d_i > d_j \\ \frac{1}{d_j} & d_i \leq d_j \end{cases}$. For all the accepting state we have $q_{acc,acc} = 1 - \Sigma_{i \neq j} q_{i,j}$
2. Then

(a) we observe that $q_{i,j}$ is symmetric, which means that $q_{j,i} = \begin{cases} \frac{1}{d_j}\frac{d_j}{d_i} & with\ edge \\ 0 & without\ edge \end{cases} = q_{i,j}$.

(b) we observe that the chain is reversible, so we have $s = (\frac{1}{M}, \frac{1}{M}, \frac{1}{M}, \frac{1}{M}..., \frac{1}{M})$

Problems 14

1. The transition probabilites is equal to find the probability $P(X_{n+1} = i|X_n = i)$ of $\forall i \in 0, 1, ..N\ X_n = i$.
   (a) If we don't have any black balls of the previous state the next state will be that some of the white balls from the first urn and the black balls from the second urn. We have the start state $P(X_{n+1} = 1|X_n = 0) = 1$ and the accepting state $P(X_{n+1} = N-1|X_n = N) = 1$.
   (b) $\forall i \in \{1, 2, ...N-1\}$. Take $X_{n+1} = i-1$ It's equal to choose the black ball from the first urn and white ball from the second urm. So we have $P(X_{n+1} = i-1)|X_n = i) = \frac{i}{N}^2$. Similarly, when $X_{n+1} = i+1$ we have $P(X_{n+1} = i+1)|X_n = i) = \frac{N-i}{N}^2$
   (c) Therefore, teh number of black balls in teh first urn remains the same as we picked the different colors. So, we have $P(X_{n+1} = i|X_n = i) = \frac{i}{N}\frac{N-i}{N} + \frac{N-i}{N}\frac{i}{N} = 2\frac{i(N-i)}{N^2}$
2. **Proofs by Induction**
   (a) For $i = 0$, we have when $j = 1$, $\frac{\binom{N}{0}\binom{N}{N}}{\binom{2N}{N}} * 1 = \frac{\binom{N}{1}\binom{N}{N-1}}{\binom{2N}{N}N^2}$. So $s_0 q_{0,1} = s_1 q_{1,0}$ is true.
   (b) Suppose $i = 1, ...N-1$ is for $j = i-1$ and $j = i+1$. Then for $1 < i \leq N-1$ and $j = i-1$, we have to prove $s_i q_{i,i-1} = s_{i-1} q_{i-1,i}$ $\frac{\binom{N}{i}\binom{N}{N}}{\binom{2N}{N}} = \frac{\binom{N}{i-1}\binom{N}{N-i+1}}{N^2}$.
   (c) To prove $LHS = RHS$. We have $\frac{N!}{(i-1)!(N-i)!} = \frac{N!}{i!(N-i)!}i = \binom{N}{i}\binom{N}{N-i}i^2 = \binom{N}{i-1}\binom{N}{N-i+1}(N-i+1)^2 = \frac{N!}{(i-1)1!(N-i+1)!}(N-i+1) = \frac{N!}{(i-1)!(N-i)!}$. End of proof.
   (d) So, we have showed that the chain is reversible, s is stationary distribution.

Problems 16

1. Suppose the space state is M. We have to going to prove that $v = (v_1, ...v_N)$ statisfy the equation $v = vQ$. And we have the i_{th} equation $v_i = \Sigma_{j=1}^N v_j P(X_{n+1} = i|X_n = j)$, applying $P(X_{n+1} = i|X_n = j) = \frac{\omega_{i,j}}{v_j}$, so $v_i = \Sigma_{j=1}^N v_j \frac{\omega_{i,j}}{v_j} = \Sigma_{j=1}^N \omega_{i,j}$. So, we prove that the stationary distribution is proportional to v.
2. For arbitary reversible Markov Chain with transposition matrix Q on the space state M. For the distribution $s = (s_1, s_2, ...s_N)$. we have $s_i q_{i,j} = s_j q_{j,i}$. By *16(1)*, the $s_i q_{i,j}$ is symmetric and undirected, so we have $v_i = \Sigma_j s_i q_{i,j} = \Sigma_j s_j q_{j,i} = \Sigma_j P(X_0 = j)P(X_1 = i|X_o = j) = P(X_1 = i) = s_i$. By 16(1), v is not proportional to s. So, $v = s$.

Problems 17

1. Given $Q_C = \begin{bmatrix} 0.2 & 0.8 \\ 0.8 & 0.2 \end{bmatrix}$. Because the matrix is symmmetric, we have the uniform $s_C = (\frac{1}{2}, \frac{1}{2})$. So, we have $Q_M = \begin{bmatrix} 0.7 & 0.3 \\ 0.6 & 0.4 \end{bmatrix}$. Solve the equation, we have $s_M = (\frac{2}{3}, \frac{1}{3})$.
2. It's Markov Chain. Because the cat and mouse have the fact their past and future are independent. Hence, the movements of the cat' and mouse's are Markov Chain/

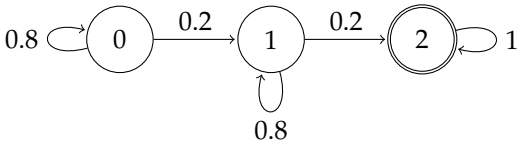
Problem 1 continued on next page...

3. Let a and b as desired value. we have

$$\begin{cases} a = 0.8 * 0.4 + 0.2 * 0.6 + 0.2 * 0.4 * (1 + a) + 0.6 * 0.8 * (1 + b) \\ b = 0.8 * 0.7 + 0.2 * 0.3 + 0.8 * 0.3 * (1 + a) + 0.2 * 0.7 * (1 + b) \end{cases}$$

So, we have $\begin{cases} a = \frac{335}{169} \doteq 1.98 \\ b = \frac{290}{169} \doteq 1.72 \end{cases}$

Problems 18

1. 

2. Notice that state zero is transient. Finally, when we get to the state two, the chain stays there forever, the state 2 is the accepting state.

   So, our variable T which is the time of arrival in state two has the distribution of $k \to \binom{k+r-1}{k}(1-p)^r p^k$, which is the negative binomial distribution. The parameter $r = 2$ and $p = 0.8$

   So, we have the Mean and the Variance: $E(T) = \frac{pr}{1-p} = \frac{0.8*2}{1-0.8} = 8$ and $Var(T) = \frac{pr}{(1-p)^2} = \frac{0.8*2}{(1-0.8)^2} = 40$

Problems 19

1. First, we notice that the state 1 and 2 is transient, which the random N equals 0 iff first step to state 2 and second step to state 3. Hence, we have $P(N = 0) = \frac{1}{2}^2 = \frac{1}{4}$

   $\forall k \in N$, we have 2 conditions for letting the state 1 be visited k times before leaves state 2:

   (a) The chain comes to state one in advance of $k_t h$ visit.

   (b) $N = 0$ happens when the chain comes to $k_t h$ visits.

   Denote the times to visit state 2 are t. We just first pick the $k + l$ from $2l$ state and choose the state 2 by the probability of $\frac{1}{2}$ for $k + 1$ times followed by state 1.

   So, $P(N = k) = \frac{1}{4}\Sigma_{t=0}^{k}\binom{k+t}{2t}(\frac{1}{2})^{k+t}$.

   Eventually, we have $P(N = k) = \begin{cases} \frac{1}{4} & k = 0 \\ \frac{1}{4}\Sigma_{t=0}^{k}\binom{k+t}{2t}(\frac{1}{2})^{k+t} & k \in N \end{cases}$. For validation, we have $\Sigma x P(x) = \frac{1}{4}\Sigma_{k=0}^{\infty}\Sigma_{t=0}^{k}\binom{k+t}{2t}(\frac{1}{2})^{k+t} = \frac{1}{4}\Sigma_{t=0}^{k}\Sigma_{k=0}^{\infty}\binom{k+t}{2t}(\frac{1}{2})^{k+t} + \frac{1}{4}$ Applying Mathmetica, we have $Ans = \frac{1}{4} + \frac{1}{9}\Sigma 4^{-t}(3 * 2^{2k+1}n + 5 * 2^{2k+1} - 1) \doteq 1$

2. Notice state 1 and 2 is reducible and after it's reduced to only state 1, the new chain has stationary distribution. The average time that the chain spends in state three is equal to S_3. Also, the new chain has symmetric transposition matrix

$$\begin{bmatrix} 0.5 & 0.5 & 0 & 0 & 0 \\ 0.5 & 0.5 & 0.3 & 0.2 & 0 \\ 0 & 0.3 & 0 & 0 & 0.7 \\ 0 & 0.2 & 0 & 0.8 & 0 \\ 0 & 0.5 & 0.7 & 0 & 0.3 \end{bmatrix}$$

   because we have $qi, j = q_{j,i} \forall$ states i and j. Hence, it is reversible, hence $s_3 = \frac{1}{4}$

Three Server Organizations

I. basic settings

In [1]:
```
import heapq
import random
import matplotlib.pyplot as plt
import numpy as np
import queue
import copy
```

1. Denotions

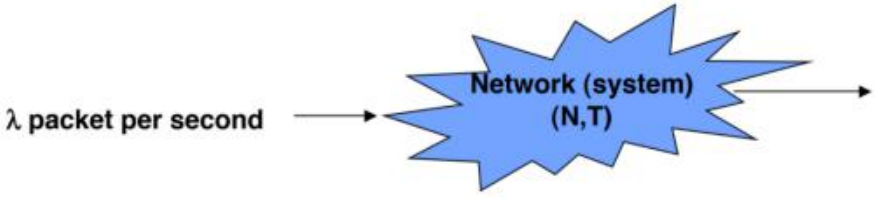
a. Random events

- Arrival process
  - Packets arrive according to a random process
  - Typically the arrival process is modeled as Poisson
- The Poisson process
  - Arrival rate of λ packets per second
  - Over a small interval δ, we have $P_{exactly\ one\ arrival} = \lambda\delta + o(\delta)$, $P_{0\ arrivals} = 1 - \lambda\delta + o(\delta)$, $P_{more\ than\ one\ arrival} = O(\delta)$ Where $\frac{O(\delta)}{\delta} \to 0$ as $\delta \to 0$.
- Also the process overall can be described as:
  - $P_{n\ arrivals\ in\ interval\ T} = \frac{(\lambda T)^n e^{-\lambda T}}{n!}$
  - where, $n = number\ of\ arrivals\ in\ T$,

b. Customer (Server)

Denote the customer have the init variables.

- The customer obeys the Markov property(memoryless):
  - $P(T \le t_0 + t | T > t_0) = P(T \le t)$
- The arriving of the customer obeys the rule of Little's theorem



- **N = average number of packets in system**
- **T = average amount of time a packet spends in the system**
- **λ = arrival rate of packets into the system (not necessarily Poisson)**

- **Little's theorem: N = λT**
  - **Can be applied to entire system or any part of it**
  - **Crowded system -> long delays**
    On a rainy day people drive slowly and roads are more congested!

In [43]:
```
class Customer:
    def __init__(self,arrival_time,service_start_time,service_time,fdm=0,k=0):
        self.arrival_time=arrival_time
        self.service_start_time=service_start_time
        self.service_time=service_time
        self.service_end_time=self.service_start_time+self.service_time
        if fdm==0:
            self.wait=self.service_start_time-self.arrival_time
        else:
            self.wait=(self.service_start_time-self.arrival_time)/k
```

II. Three Simulations

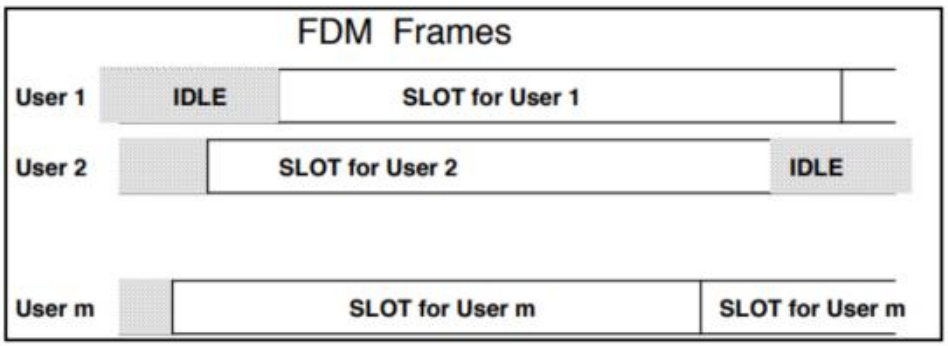
1. FDM Queue Simulation

FDM is supposed to be a kind of signal mux technology widely used in dem and modem. In the category of Queuing theory, it's actually very similar to `M-D-1 Queue` with vacations, where $D = \frac{1}{\mu} + \frac{\rho}{\mu-\lambda}$, and vacations means server goes on vacation for m time units when there is nothing to transmit

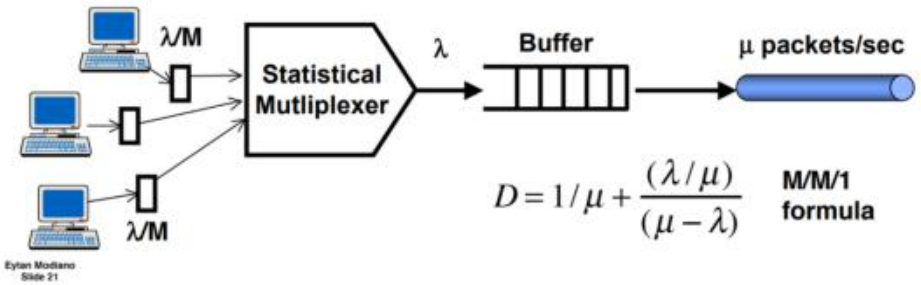
The state space is the set $\{0, 1, 2, 3, \dots\}$ where the value corresponds to the number of entities in the system, including any currently in service.Arrivals occur at rate λ according to a Poisson process and move the process from state i to $i + 1$.Service times are deterministic time D (serving at rate $\mu = \frac{1}{D}$).A single server serves entities one at a time from the front of the queue, according to a first-come, first-served discipline. When the service is complete the entity leaves the queue and the number of entities in the system reduces by one.

We have the transition Matrix defined as

$$P = \begin{pmatrix} a_0 & a_1 & a_2 & a_3 \dots \\ a_0 & a_1 & a_2 & a_3 \dots \\ 0 & a_0 & a_1 & a_2 \\ 0 & 0 & a_0 & a_1 \end{pmatrix}, a_n = \frac{\lambda^n}{n!} e^{-\lambda}, n = 0, 1, \dots$$



The buffer is of infinite size, so there is no limit on the number of entities it can contain.



In [44]:
```
def FDM(lambd,mu,k):
    # Simulation Time
    simulation_time = 300
    simclock=0
```

```
    Customers=[]
    kqueue=[]
    last_k_queue_Customer = [[]]
    last_k_Customer = []
    while simclock<simulation_time:
        if len(Customers)<=k:
            arrival_time=1/mu
            service_start_time=arrival_time
        else:
            arrival_time+=1/mu
            k_customer = last_k_Customer[len(last_k_Customer)-k:]
            service_start_time=max(arrival_time,min(k_customer))

        service_time = random.expovariate(mu)
        end_time = service_start_time + service_time
        last_k_Customer.append(end_time)

        # Add new Customer
        if arrival_time <=simulation_time and end_time <= simulation_time:
            Customers.append(Customer(arrival_time,service_start_time,service_ti

        # Increment clock till next end of service
        if arrival_time <= simulation_time:
            simclock=arrival_time
        else:
            simclock = simulation_time

    Waits=[a.wait for a in Customers]
    avgQdelay=sum(Waits)/len(Waits)

    served = len(Customers)
    Total_Times = [a.wait+a.service_time for a in Customers]
    Mean_Time = sum(Total_Times)/len(Total_Times)
    avgQlength = (Mean_Time/float(simclock)) * served
    Service_Times=[a.service_time for a in Customers]

    if sum(Service_Times) >= simclock:
        util = 1
    else:
        util=sum(Service_Times)/simclock

    avgQlength/=k
    avgQdelay/=k

    # Output Result
    print ('FDM Results: lambda = %lf, mu = %lf, k = %d' % (lambd,mu,k))
    print ('FDM Total customer served: %d' % served)
    print ('FDM Average queue length: %lf' % avgQlength)
    print ('FDM Average customer delay in queue: %lf' % avgQdelay)
    return avgQdelay
```

In [78]:
```
FDM(5.0/60, 8.0/60,1)
```

```
FDM Results: lambda = 0.083333, mu = 0.133333, k = 1
FDM Total customer served: 40
FDM Average queue length: 1.017350
FDM Average customer delay in queue: 2.918219
```

Out[78]:
```
2.918219374149586
```

The validation of the average delay computed

Equilibrium analysis

- Assume m Poisson streams of fixed length packets of arrival rate λ/k each multiplexed by FDM on m subchannels.
- Suppose it takes k time units to transmit a packet, so μ=1/k.
- The total system load: $\rho = \lambda$

Average customer delay in queue

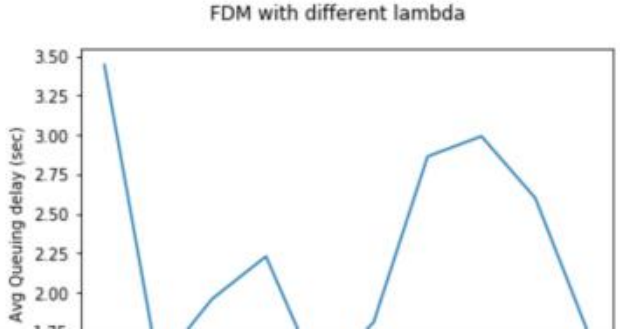
For `M-D-1 queue` , we havet the average the estimate of customer delay is $W = \frac{\lambda E[x^2]}{2(1-\rho)}$

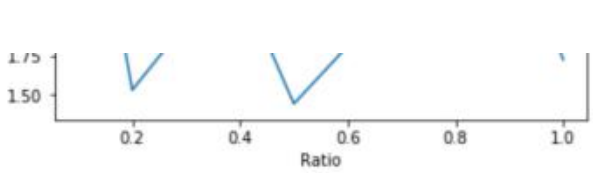
$$W_{FDM} = \frac{\rho k^2}{2(1-\rho)}$$

In this case $\frac{\frac{1}{12} * 1^2}{2(1-\frac{1}{12})} = \frac{11}{6} \doteq 2.9182193$

In [47]:
```
mu = 1000.0 / 60
ratios = [u / 10.0 for u in range(1, 11)]
avgdelay = []
for ro in ratios:
    delay= FDM(mu*ro, mu, 1)
    avgdelay.append(delay)
plt.plot(ratios, avgdelay)
plt.xlabel('Ratio')
plt.ylabel('Avg Queuing delay (sec)')
plt.suptitle('FDM with different lambda', fontsize=12)
plt.show()
```

```
FDM Results: lambda = 1.666667, mu = 16.666667, k = 1
FDM Total customer served: 4927
FDM Average queue length: 57.535673
FDM Average customer delay in queue: 3.442672
FDM Results: lambda = 3.333333, mu = 16.666667, k = 1
FDM Total customer served: 4995
FDM Average queue length: 26.509387
FDM Average customer delay in queue: 1.532317
FDM Results: lambda = 5.000000, mu = 16.666667, k = 1
FDM Total customer served: 4890
FDM Average queue length: 32.899873
FDM Average customer delay in queue: 1.958436
FDM Results: lambda = 6.666667, mu = 16.666667, k = 1
FDM Total customer served: 4950
FDM Average queue length: 37.765467
FDM Average customer delay in queue: 2.228333
FDM Results: lambda = 8.333333, mu = 16.666667, k = 1
FDM Total customer served: 4939
FDM Average queue length: 24.782987
FDM Average customer delay in queue: 1.445057
FDM Results: lambda = 10.000000, mu = 16.666667, k = 1
FDM Total customer served: 4938
FDM Average queue length: 30.808183
FDM Average customer delay in queue: 1.812072
FDM Results: lambda = 11.666667, mu = 16.666667, k = 1
FDM Total customer served: 4954
FDM Average queue length: 48.270771
FDM Average customer delay in queue: 2.862641
FDM Results: lambda = 13.333333, mu = 16.666667, k = 1
FDM Total customer served: 4909
FDM Average queue length: 49.930818
FDM Average customer delay in queue: 2.990681
FDM Results: lambda = 15.000000, mu = 16.666667, k = 1
FDM Total customer served: 4903
FDM Average queue length: 43.436982
FDM Average customer delay in queue: 2.597720
FDM Results: lambda = 16.666667, mu = 16.666667, k = 1
FDM Total customer served: 4933
FDM Average queue length: 29.376556
FDM Average customer delay in queue: 1.726151
```



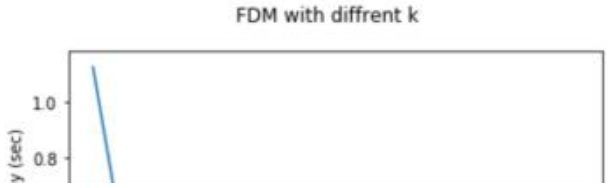


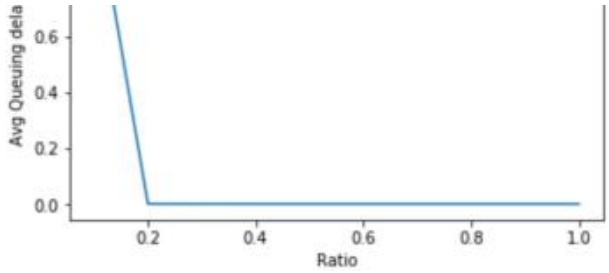
MMK with different k

As we can see in the graphe. The relation fluctuates a lot. On average, the Queing delay is lager than 2.0.

In [48]:
```
mu = 1000.0 / 60
ratios = [u / 10.0 for u in range(1, 11)]
avgdelay = []
k = 1
for ro in ratios:
    delay = FDM(mu*ro, mu, k)
    avgdelay.append(delay)
    k=k+1
plt.plot(ratios, avgdelay)
plt.xlabel('Ratio')
plt.ylabel('Avg Queuing delay (sec)')
plt.suptitle('FDM with diffrent k', fontsize=12)
plt.show()
```

```
FDM Results: lambda = 1.666667, mu = 16.666667, k = 1
FDM Total customer served: 4974
FDM Average queue length: 19.652515
FDM Average customer delay in queue: 1.126007
FDM Results: lambda = 3.333333, mu = 16.666667, k = 2
FDM Total customer served: 5000
FDM Average queue length: 0.499841
FDM Average customer delay in queue: 0.000486
FDM Results: lambda = 5.000000, mu = 16.666667, k = 3
FDM Total customer served: 5001
FDM Average queue length: 0.334560
FDM Average customer delay in queue: 0.000007
FDM Results: lambda = 6.666667, mu = 16.666667, k = 4
FDM Total customer served: 5002
FDM Average queue length: 0.255556
FDM Average customer delay in queue: 0.000000
FDM Results: lambda = 8.333333, mu = 16.666667, k = 5
FDM Total customer served: 5004
FDM Average queue length: 0.201994
FDM Average customer delay in queue: 0.000000
FDM Results: lambda = 10.000000, mu = 16.666667, k = 6
FDM Total customer served: 5004
FDM Average queue length: 0.171261
FDM Average customer delay in queue: 0.000000
FDM Results: lambda = 11.666667, mu = 16.666667, k = 7
FDM Total customer served: 5006
FDM Average queue length: 0.142356
FDM Average customer delay in queue: 0.000000
FDM Results: lambda = 13.333333, mu = 16.666667, k = 8
FDM Total customer served: 5007
FDM Average queue length: 0.125355
FDM Average customer delay in queue: 0.000000
FDM Results: lambda = 15.000000, mu = 16.666667, k = 9
FDM Total customer served: 5007
FDM Average queue length: 0.113930
FDM Average customer delay in queue: 0.000000
FDM Results: lambda = 16.666667, mu = 16.666667, k = 10
FDM Total customer served: 5009
FDM Average queue length: 0.100388
FDM Average customer delay in queue: 0.000000
```





FDM with different k

As we can see in the graphe. The Queuing delay is negatively related to average Queuing delay. The trend drop sharply during the range of $0 - 0.2$, which is in accordance to the previous induction.

On average, the Queing delay is lager than 0.2.

2. M-M-1 Queue Simulation

The state space is the set $\{0, 1, 2, 3, \dots\}$, where the value corresponds to the number of customers in the system. We let Pn denote the probability of n customers in the system.

Arrivals occur at rate λ according to a Poisson process, which moves the system from state i to state $i + 1$ (Since every time the package come). Service times are exponentially distributed with rate parameter μ so that $\frac{1}{\mu}$ is the mean service time.

A single server serves customers one at a time from the front of the queue, according to a first-come, first-served discipline. When the service is complete the customer leaves the queue and the number of customers in the system reduces by one, i.e., the system moves from state i to $i - 1$.

Suppose the system is in state n. Then, the balance equation reads

$(\lambda + \mu)P_n = \lambda P_{n-1} + \mu P_{n+1}$

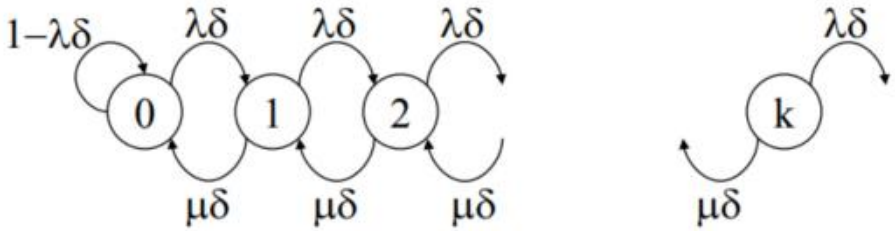
Essentially, $\lambda + \mu$: rate of an arrival or departure to P_n. λ: rate of an arrival to P_{n-1}. μ: rate of a departure from P_{n+1}

The boundary condition (near an empty queue) is that $\lambda P_0 = \mu P_1$.

Thus, $P_1 = \frac{\lambda}{\mu} P_0$

$P_2 = \frac{\lambda}{\mu} P_1 + \frac{1}{\mu}(\mu P_1 - \lambda P_0) = \frac{\lambda}{\mu} P_1 = \lambda \mu^2 P_0$

$Pn = \frac{\lambda}{\mu}^n P_0$



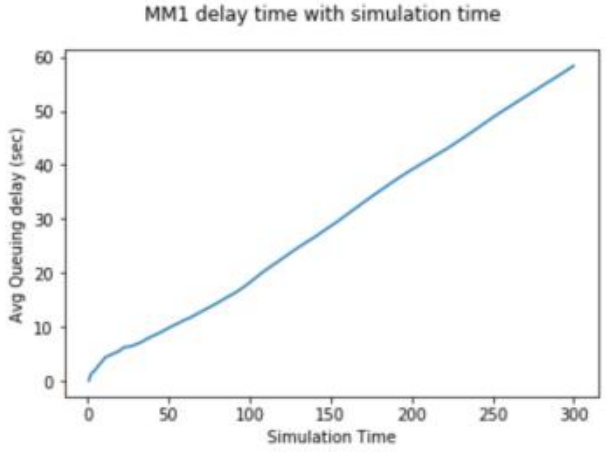
In [3]:
```
def MM1(lambd,mu,k):
    simulation_time = 300
    mu = mu*k
    ratio = lambd/mu


    # Initialize Parameters
    qu = queue.Queue()
    curr_process = None
    simulation_qu = []
```

```
    service = []
    arrival = []
    wait_time = []
    server_busy = False
    list_wait = []
    list_delay = []
    served=0

    num_processes = int(np.random.poisson(lambd)* simulation_time)
    num_processes_served = 0
    for i in range(num_processes):
        temp = np.random.exponential(1/lambd)
        if i==0:
            simulation_qu.append(0)
        else:
            simulation_qu.append(int(temp - temp%1))

    while not len(service) == num_processes:
        temp = np.random.exponential(1/mu)
        if not int(temp- temp%1)<1:
            service.append(int(temp - temp%1))

    service_copy = copy.deepcopy(service)

    for i in range(num_processes):
        if i == 0:
            arrival.append(0)
        else:
            arrival.append(arrival[i-1] + simulation_qu[i])
        wait_time.append(0)

    # Simulation of M/M/1 Queue (i represents current time)
    for i in range(simulation_time):
        if server_busy:
            for item in list(qu.queue):
                wait_time[item] = wait_time[item] + 1
            service[curr_process] = service[curr_process] - 1
            if service[curr_process] == 0:
                server_busy = False
                served+=1
                num_processes_served = num_processes_served + 1

        for j in range(num_processes):
            if i== arrival[j]:
                qu.put(j)

        if not server_busy and not qu.empty():
            curr_process = qu.get()
            server_busy = True


        sum_wait = 0
        sum_delay = 0

        for i in range(num_processes_served):
            sum_wait = sum_wait + wait_time[i]
            sum_delay = sum_delay + wait_time[i] + service_copy[i]

        if num_processes_served == 0:
            list_wait.append(0)
            list_delay.append(0)
        else:
            list_wait.append(sum_wait/(num_processes_served))
            list_delay.append(sum_delay/(num_processes_served))

    # Output Result
    print ('MM1 Results: lambda = %lf, mu = %lf, k = %d' % (lambd,mu,k))
    print ('MM1 Average queue length: %d' % served)
    print ('MM1 Average customer wait in queue:'+ str(sum(list_wait)/len(list_wa
    print ('MM1 Average customer delay in queue:' + str(sum(list_delay)/len(list
    return list_delay,simulation_time
```

In [8]:
```
delay_sim = MM1(1,2,3)
```

```
plt.plot([i+1 for i in range(sim)], delay)
plt.ylabel("Avg Queuing delay (sec)")
plt.xlabel("Simulation Time")
plt.suptitle('MM1 delay time with simulation time', fontsize=12)
plt.show()
```

```
MM1 Results: lambda = 1.000000, mu = 6.000000, k = 3
MM1 Average queue length: 299
MM1 Average customer wait in queue:0.28171046780770603
MM1 Average customer delay in queue:0.2916771344743727
```



The validation of the average delay computed

Equilibrium analysis

- We want to obtain $P(n) = the\ probability\ of\ being\ in\ state\ n$
- At equilibrium $\lambda P(n) = \mu P(n+1)$ for all n
  - Local balance equations between two states $(n, n+1)$
  - $P(n+1) = (\frac{\lambda}{\mu})P(n) = \rho P(n), \rho = \frac{\lambda}{\mu}$
- It follows: $P(n) = \rho^n P(0)$
- By axiom of probability
  - $\Sigma_{i=0}^{\infty} P(n) = 1$
    - $\Sigma_{i=0}^{\infty} \rho^n P(0) = fracP(0)1 - \rho = 1$
    - $P(0) = 1 - \rho$
    - $P(n) = \rho^n(1 - \rho)$

Average number of customers in the system

$$N = \Sigma_{n=0}^{\infty} nP(n) = \Sigma_{n=0}^{\infty} n\rho^n(1-\rho) = \frac{\rho}{1-\rho} = \frac{\lambda}{\mu - \lambda}$$

where N = Average number of customers in the system. In this case $\frac{1}{6-1} = \frac{1}{5} \doteq 0.2125431$

Average customer wait in queue

The average amount of time that a customer spends in the system can be obtained from Little's formula ($N = \lambda T \Rightarrow T = \frac{N}{\lambda}$)

$$T = \frac{1}{\mu - \lambda}$$

In this case $\frac{1}{6-1} \doteq 0.2817104$

Average customer delay in queue

T includes the queueing delay plus the service time (Service time = $D_{TP} = \frac{1}{\mu}$). W = amount of time spent in queue = $T - \frac{1}{\mu}$

$$W = \frac{1}{\mu - \lambda} - \frac{1}{\mu}$$

In this case $\frac{1}{6-1} - \frac{1}{6} \doteq 0.2916771$

Average queue size

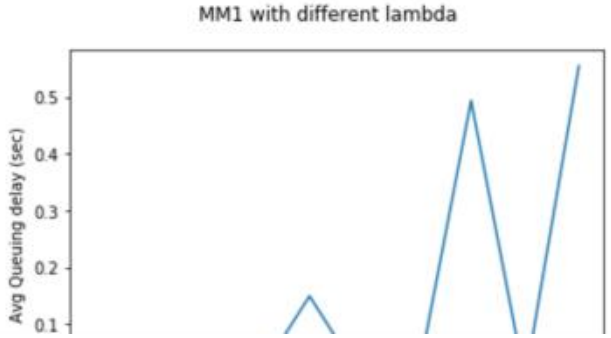
The average number of customers in the buffer can be obtained from little's formula

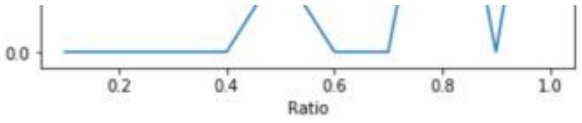
$$N_Q = \lambda W = \frac{\lambda}{\mu - \lambda} - \frac{\lambda}{\mu} = N - \rho$$

In this case $300 - \frac{1}{5} \doteq 299$

In [17]:
```
mu = 100.0 / 60
ratios = [u / 10.0 for u in range(1, 11)]
avgdelay = []
for ro in ratios:
    delay,sim= MM1(mu*ro, mu, 1)
    avgdelay.append(sum(delay)/len(delay)/100)
plt.plot(ratios, avgdelay)
plt.xlabel('Ratio')
plt.ylabel('Avg Queuing delay (sec)')
plt.suptitle('MM1 with different lambda', fontsize=12)
plt.show()
```

```
MM1 Results: lambda = 0.166667, mu = 1.666667, k = 1
MM1 Average queue length: 0
MM1 Average customer wait in queue:0.0
MM1 Average customer delay in queue:0.0
MM1 Results: lambda = 0.333333, mu = 1.666667, k = 1
MM1 Average queue length: 0
MM1 Average customer wait in queue:0.0
MM1 Average customer delay in queue:0.0
MM1 Results: lambda = 0.500000, mu = 1.666667, k = 1
MM1 Average queue length: 0
MM1 Average customer wait in queue:0.0
MM1 Average customer delay in queue:0.0
MM1 Results: lambda = 0.666667, mu = 1.666667, k = 1
MM1 Average queue length: 0
MM1 Average customer wait in queue:0.0
MM1 Average customer delay in queue:0.0
MM1 Results: lambda = 0.833333, mu = 1.666667, k = 1
MM1 Average queue length: 239
MM1 Average customer wait in queue:0.13716155151389758
MM1 Average customer delay in queue:0.1493398989137973
MM1 Results: lambda = 1.000000, mu = 1.666667, k = 1
MM1 Average queue length: 0
MM1 Average customer wait in queue:0.0
MM1 Average customer delay in queue:0.0
MM1 Results: lambda = 1.166667, mu = 1.666667, k = 1
MM1 Average queue length: 0
MM1 Average customer wait in queue:0.0
MM1 Average customer delay in queue:0.0
MM1 Results: lambda = 1.333333, mu = 1.666667, k = 1
MM1 Average queue length: 243
MM1 Average customer wait in queue:0.4810109782642805
MM1 Average customer delay in queue:0.4931651759766524
MM1 Results: lambda = 1.500000, mu = 1.666667, k = 1
MM1 Average queue length: 0
MM1 Average customer wait in queue:0.0
MM1 Average customer delay in queue:0.0
MM1 Results: lambda = 1.666667, mu = 1.666667, k = 1
MM1 Average queue length: 255
MM1 Average customer wait in queue:0.5424437181059363
MM1 Average customer delay in queue:0.554084189474754
```



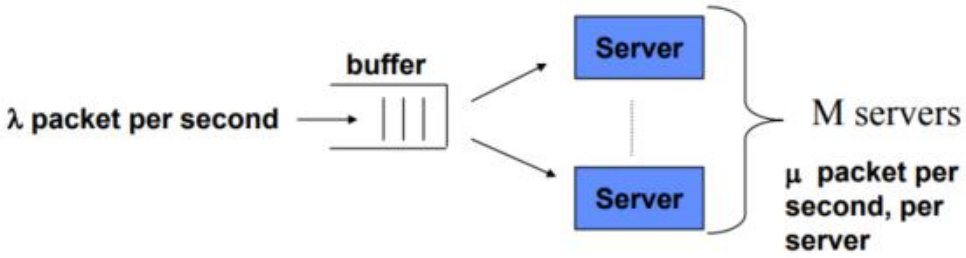


MM1 with different λ

As we can see in the graphe. The Queuing delay is roughly positively related to average Queuing delay. The relation fluctuates during the range of $0.8 - 1$, which is in accordance to the previous induction.

On average, the Queing delay is lager than 0.2.

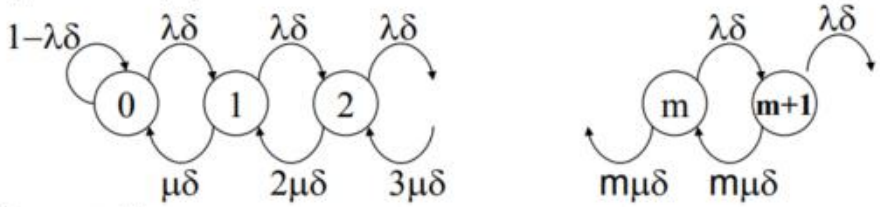
3. M-M-k(m) Queue Simulation



An $M/M/k$ queue is a stochastic process whose state space is the set $\{0, 1, 2, 3, \dots\}$ where the value corresponds to the number of customers in the system, including any currently in service.

Arrivals occur at rate λ according to a Poisson process and move the process from state i to $i + 1$. Service times have an exponential distribution with parameter μ. If there are fewer than c jobs, some of the servers will be idle. If there are more than c jobs, the jobs queue in a buffer. The buffer is of infinite size, so there is no limit on the number of customers it can contain. The model can be described as a continuous time Markov chain with transition rate matrix.

- Departure rate is proportional to the number of servers in use



- Balance equations:
  - $\lambda P(n-1) = n\mu P(n) n \leq m$
  - $\lambda P(n-1) = m\mu P(n) n > m$
  - $P(n) = \begin{cases} \frac{P(0)(m\rho)^n}{n!} & n \leq m \\ \frac{P(0)(m^m \rho^n)}{m!} & n > m \end{cases} \quad \rho = \frac{\lambda}{m\mu} \leq 1$

In [80]:
```
def MMK(lambd,mu,k):
    # Simulation Time
    simulation_time = 300
    simclock=0
    Customers=[]

    last_k_Customer = []
    while simclock<simulation_time:
        if len(Customers)<=k:
            arrival_time=random.expovariate(lambd)
            service_start_time=arrival_time
        else:
            arrival_time+=random.expovariate(lambd)
            k_customer = last_k_Customer[len(last_k_Customer)-k:]
            service_start_time=max(arrival_time,min(k_customer))
```

```
        service_time = random.expovariate(mu)
        end_time = service_start_time + service_time
        last_k_Customer.append(end_time)

        # Add new Customer
        if arrival_time <=simulation_time and end_time <= simulation_time:
            Customers.append(Customer(arrival_time,service_start_time,service_ti

        # Increment clock till next end of service
        if arrival_time <= simulation_time:
            simclock=arrival_time
        else:
            simclock = simulation_time

    Waits=[a.wait for a in Customers]
    avgQdelay=sum(Waits)/len(Waits)

    served = len(Customers)
    Total_Times = [a.wait+a.service_time for a in Customers]
    Mean_Time = sum(Total_Times)/len(Total_Times)
    avgQlength = (Mean_Time/float(simclock)) * served
    Service_Times=[a.service_time for a in Customers]

    if sum(Service_Times) >= simclock:
        util = 1
    else:
        util=sum(Service_Times)/simclock

    # Output Result
    print ('MMk Results: lambda = %lf, mu = %lf, k = %d' % (lambd,mu,k))
    print ('MMk Total customer served: %d' % served)
    print ('MMk Average queue length: %lf' % avgQlength)
    print ('MMk Average customer delay in queue: %lf' % avgQdelay)
    return avgQdelay
```

In [24]:
```
MMK(5.0/60, 8.0/60,1)
```

```
MMk Results: lambda = 0.083333, mu = 0.133333, k = 1
MMk Total customer served: 20
MMk Average queue length: 2.296409
MMk Average customer delay in queue: 22.903719
MMk Time-average server utility: 0.769494
```

In [25]:
```
MMK(1,2,1)
```

```
MMk Results: lambda = 1.000000, mu = 2.000000, k = 1
MMk Total customer served: 337
MMk Average queue length: 1.100553
MMk Average customer delay in queue: 0.489263
MMk Time-average server utility: 0.550947
```

In [87]:
```
MMK(1,2,3)
```

```
MMk Results: lambda = 1.000000, mu = 2.000000, k = 3
MMk Total customer served: 291
MMk Average queue length: 0.520516
MMk Average customer delay in queue: 0.002461
MMk Time-average server utility: 0.518129
```

The validation of the average delay computed

Equilibrium analysis

- The underlying Markov chain is again a birth-death process with $\lambda_k = \lambda \; for \; k = 0, 1, 2 \ldots$ and $\mu_k = \begin{cases} k\mu & 0 < k < m \\ m\mu & k \geq m \end{cases}$

$$P(0) = [\Sigma_{n=0}^{m-1} \frac{(m\rho)^n}{n!} + \frac{(m\rho)^m}{m!(1-\rho)}]^{-1}$$

$$P_Q = \Sigma_m^\infty P(n) = \frac{P(0)(m\rho)^m}{m!(1-\rho)}$$

In this case, we have $P(0) = (\frac{\frac{3}{2}^1}{1} + \frac{\frac{3}{2}^2}{2} + \frac{\frac{3}{2}^3}{6*0.5})^{-1} = \frac{4}{15}$ and $P_Q = \frac{4}{15} * \frac{3}{2}^3/(3! * 0.5) = 0.3$

Average customer wait in queue

Apply the equation W = amount of time spent in queue = $T - 1\mu$

$$T = W + \frac{1}{\mu}$$

Average customer delay in queue

Average customer delay can be defined as :

$$W = \frac{N_Q}{\lambda}$$

In this case $0.001/0.3 = 0.003333 \doteq 0.002461$

Average queue size

The average number of customers in the buffer can be obtained from little's formula

$$N_Q = \Sigma_0^\infty nP(n+m) = P_Q(\frac{\rho}{1-\rho})$$

In this case $300 * \frac{0.5}{0.5} = 291$

In [93]:
```
mu = 1000.0 / 60
ratios = [u / 10.0 for u in range(1, 11)]
avgdelay = []
```

```
for ro in ratios:
    delay= MMK(mu*ro, mu, 1)
    avgdelay.append(delay)
plt.plot(ratios, avgdelay)
plt.xlabel('Ratio')
plt.ylabel('Avg Queuing delay (sec)')
plt.suptitle('MMK with different lambda', fontsize=12)
plt.show()
```

```
MMk Results: lambda = 1.666667, mu = 16.666667, k = 1
MMk Total customer served: 530
MMk Average queue length: 0.117236
MMk Average customer delay in queue: 0.008060
MMk Time-average server utility: 0.102996
MMk Results: lambda = 3.333333, mu = 16.666667, k = 1
MMk Total customer served: 982
MMk Average queue length: 0.245563
MMk Average customer delay in queue: 0.015977
MMk Time-average server utility: 0.193263
MMk Results: lambda = 5.000000, mu = 16.666667, k = 1
MMk Total customer served: 1427
MMk Average queue length: 0.445915
MMk Average customer delay in queue: 0.030898
MMk Time-average server utility: 0.298942
MMk Results: lambda = 6.666667, mu = 16.666667, k = 1
MMk Total customer served: 1982
MMk Average queue length: 0.632129
MMk Average customer delay in queue: 0.036500
```

MMK with different λ

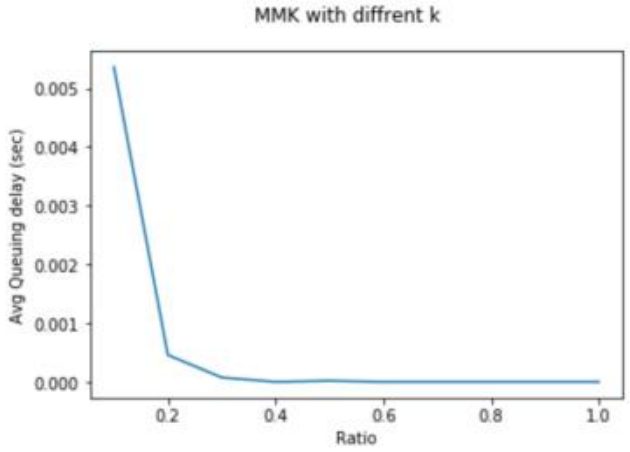
As we can see in the graphe. The Queuing delay is positively related to average Queuing delay. The relation rise aggressively during the range of $0.8 - 1$, which is in accordance to the previous induction.

On average, the Queing delay is lager than 0.4.

In [81]:
```
mu = 1000.0 / 60
ratios = [u / 10.0 for u in range(1, 11)]
avgdelay = []
k = 1
for ro in ratios:
    delay = MMK(mu*ro, mu, k)
    avgdelay.append(delay)
    k=k+1
plt.plot(ratios, avgdelay)
plt.xlabel('Ratio')
plt.ylabel('Avg Queuing delay (sec)')
plt.suptitle('MMK with diffrent k', fontsize=12)
plt.show()
```

```
MMk Results: lambda = 1.666667, mu = 16.666667, k = 1
MMk Total customer served: 494
MMk Average queue length: 0.098235
MMk Average customer delay in queue: 0.005362
MMk Results: lambda = 3.333333, mu = 16.666667, k = 2
MMk Total customer served: 1022
MMk Average queue length: 0.205646
MMk Average customer delay in queue: 0.000458
MMk Results: lambda = 5.000000, mu = 16.666667, k = 3
MMk Total customer served: 1549
MMk Average queue length: 0.306376
MMk Average customer delay in queue: 0.000074
MMk Results: lambda = 6.666667, mu = 16.666667, k = 4
MMk Total customer served: 2004
MMk Average queue length: 0.392111
MMk Average customer delay in queue: 0.000000
MMk Results: lambda = 8.333333, mu = 16.666667, k = 5
MMk Total customer served: 2480
MMk Average queue length: 0.494451
MMk Average customer delay in queue: 0.000019
MMk Results: lambda = 10.000000, mu = 16.666667, k = 6
MMk Total customer served: 3032
```

```
MMk Average queue length: 0.600825
MMk Average customer delay in queue: 0.000000
MMk Results: lambda = 11.666667, mu = 16.666667, k = 7
MMk Total customer served: 3483
MMk Average queue length: 0.715298
MMk Average customer delay in queue: 0.000000
MMk Results: lambda = 13.333333, mu = 16.666667, k = 8
MMk Total customer served: 3976
MMk Average queue length: 0.784441
MMk Average customer delay in queue: 0.000000
MMk Results: lambda = 15.000000, mu = 16.666667, k = 9
MMk Total customer served: 4590
MMk Average queue length: 0.911744
MMk Average customer delay in queue: 0.000000
MMk Results: lambda = 16.666667, mu = 16.666667, k = 10
MMk Total customer served: 5039
MMk Average queue length: 1.001046
MMk Average customer delay in queue: 0.000000
```



MMK with different k

As we can see in the graph. The Queuing delay is negatively related to average Queuing delay. The trend drop sharply during the range of $0 - 0.2$, which is in accordance to the previous induction.

On average, the Queing delay is lager than 0.001.

III. Comparison between three Simulations

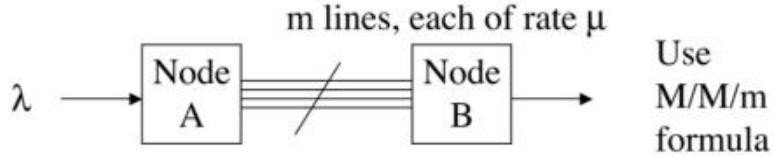
Admittedly, we have the limitation for just controlling the variable of `poisson process` for emulating the customer flow. The overall simulation is convincible and validated by the theororm induction.

In terms of the average delay time, we can jump to the conclution that `M-M-k` queue is the best strategy among three, not only for the lower delay time, but also for its algorithm's stability.

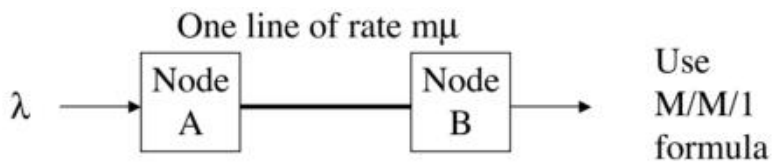
Reflection on the difference between `M-M-k` and `M-M-1`

We notice that, for some parameters, we have `M-M-1` over-performance the previous one. We can figure it out from the graph above.

- Bank with m tellers
- Network with parallel transmission lines



VS



When the system is lightly loaded, P_Q 0, and Single server is m times faster When system is heavily loaded, queueing delay dominates and systems are roughly the same

IV. The State of Art

From the paper [4], we can add a traffic equation to make noise in λ, such as:

$$\lambda_{ext} + (1 - \mathbb{E}[e^{-\theta D}])\lambda = \lambda < \mu$$

Assuming a FIFO service discipline, we can approximate delay as $D \sim Exponential(\mu - \lambda)$. Noticing that the expected recall rate is the moment-generating function for D,

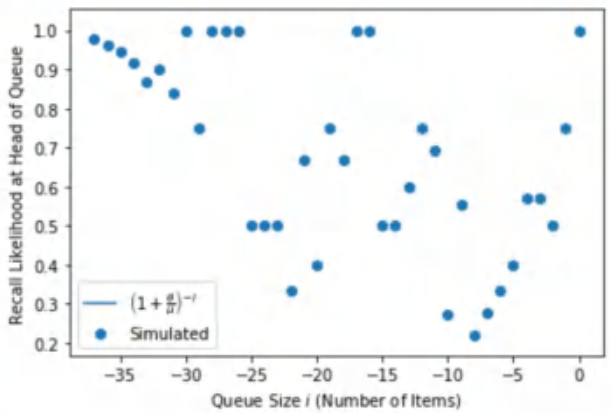
$$\mathbb{E}[e^{-\theta D}] = \frac{\mu - \lambda}{\mu - \lambda + \theta}$$

Plugging this expression into the traffic equation,

$$\lambda = \frac{\mu - \lambda + \theta}{\mu - \lambda} \cdot \lambda_{ext} < \mu$$

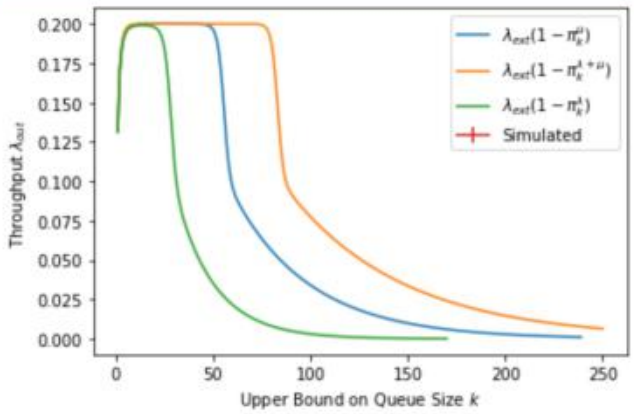
In [102]:
```
import numpy as np
from collections import defaultdict, Counter
T = 10000
theta = 0.01
arrival_rate = 0.2
service_rate = 0.4
using_clocked_delays = True
using_fifo = True
using_balking = False
flow_rate = (service_rate + arrival_rate - np.sqrt((service_rate + arrival_rate)
expected_recall_rate = (service_rate - flow_rate) / (service_rate - flow_rate +
feedback_threshold = np.log(service_rate / arrival_rate) / np.log(1 + theta / se
q = queue.Queue()
exits = []
delays = []
# initialize queue with some items
size = int(np.log(service_rate / arrival_rate) / np.log(1 + theta / service_rate
iits = np.random.exponential(1 / (arrival_rate + service_rate), size) # inter-ar
t = -sum(iits)
for iit in iits:
    t += iit
    q.put(t)
size = 0
sizes = [size]
t = 0
outcomes_at_queue_size = defaultdict(list)
for _ in range(T):
    t += np.random.exponential(1 / (arrival_rate + service_rate))
    if np.random.random() < arrival_rate / (arrival_rate + service_rate) and (no
        q.put(t)
        size += 1
    elif not q.empty():
        if using_clocked_delays:
            delay = t - q.get()
        else:
            q.get()
            delay = np.random.exponential(1 / (service_rate - flow_rate))
        delays.append(delay)
        if np.random.random() < np.exp(-theta*delay):
            outcomes_at_queue_size[size].append(1)
            exits.append(t)
            size -= 1
        elif True:
            outcomes_at_queue_size[size].append(0)
```

```
            q.put(t)

    sizes.append(size)
timesteps_at_queue_size = Counter(sizes)
plt.xlabel(r'Queue Size $i$ (Number of Items)')
plt.ylabel('Recall Likelihood at Head of Queue')
queue_sizes, outcomes = zip(*outcomes_at_queue_size.items())
plt.scatter(queue_sizes, [np.mean(x) for x in outcomes], label='Simulated')

queue_size_states = np.arange(0, max(queue_sizes)+1, 1)
plt.plot(queue_size_states, (1+theta/service_rate)**(-queue_size_states), label=

plt.legend(loc='lower left')
plt.show()

num_reps = 5
#arrival_rates = np.arange(2*service_rate/20, 2*service_rate, 2*service_rate/20)
#thresholds = [10, 50, 100]
arrival_rates = [0.1, 0.3]#[0.25, 0.3, 0.35]
thresholds = np.arange(1, 251, 1)
throughputs = [[[] for _ in arrival_rates] for _ in thresholds]
pis_approx = np.zeros(len(thresholds))
pis_ubound = np.zeros(len(thresholds))
pis_lbound = np.zeros(len(thresholds))
for i, threshold in enumerate(thresholds):
    queue_size_states = np.arange(0, threshold+1, 1)
    pis = ((arrival_rate / service_rate)**(queue_size_states))*((1+theta/service
    pis /= np.sum(pis)
    pis_approx[i] = pis[-1]

    pis = ((arrival_rate / service_rate)**(queue_size_states))*((1+theta/(arriva
    pis /= np.sum(pis)
    pis_ubound[i] = pis[-1]

    pis = ((arrival_rate / service_rate)**(queue_size_states))*((1+theta/(arriva
    pis /= np.sum(pis)
    pis_lbound[i] = pis[-1]
plt.xlabel(r'Upper Bound on Queue Size $k$')
plt.ylabel(r'Throughput $\lambda_{out}$')
plt.plot(thresholds, arrival_rate*(1-pis_approx), label=r'$\lambda_{ext}(1-\pi^{
plt.plot(thresholds, arrival_rate*(1-pis_ubound), label=r'$\lambda_{ext}(1-\pi^{
plt.plot(thresholds, arrival_rate*(1-pis_lbound), label=r'$\lambda_{ext}(1-\pi^{
plt.errorbar(thresholds, [np.mean(x) for x in throughputs], yerr=[np.std(x)/np.s
plt.legend(loc='best')
plt.show()
```



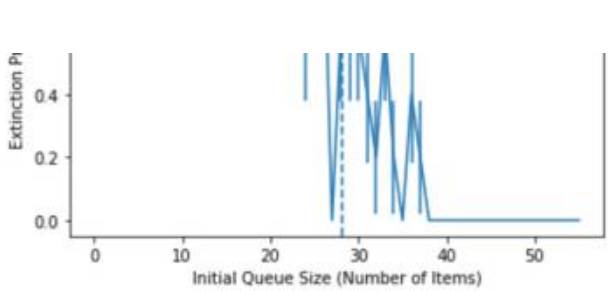

```
/opt/anaconda3/envs/my-rdkit-env/lib/python3.6/site-packages/ipykernel_launche
r.py:79: RuntimeWarning: overflow encountered in power
/opt/anaconda3/envs/my-rdkit-env/lib/python3.6/site-packages/ipykernel_launche
r.py:80: RuntimeWarning: invalid value encountered in true_divide
/opt/anaconda3/envs/my-rdkit-env/lib/python3.6/site-packages/ipykernel_launche
r.py:71: RuntimeWarning: overflow encountered in power
/opt/anaconda3/envs/my-rdkit-env/lib/python3.6/site-packages/ipykernel_launche
r.py:72: RuntimeWarning: invalid value encountered in true_divide
/opt/anaconda3/envs/my-rdkit-env/lib/python3.6/site-packages/numpy/core/fromnum
eric.py:3335: RuntimeWarning: Mean of empty slice.
  out=out, **kwargs)
```

```
/opt/anaconda3/envs/my-rdkit-env/lib/python3.6/site-packages/numpy/core/_method
s.py:161: RuntimeWarning: invalid value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)
/opt/anaconda3/envs/my-rdkit-env/lib/python3.6/site-packages/numpy/core/_method
s.py:217: RuntimeWarning: Degrees of freedom <= 0 for slice
  keepdims=keepdims)
/opt/anaconda3/envs/my-rdkit-env/lib/python3.6/site-packages/numpy/core/_method
s.py:186: RuntimeWarning: invalid value encountered in true_divide
  arrmean, rcount, out=arrmean, casting='unsafe', subok=False)
/opt/anaconda3/envs/my-rdkit-env/lib/python3.6/site-packages/numpy/core/_method
s.py:209: RuntimeWarning: invalid value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)
```

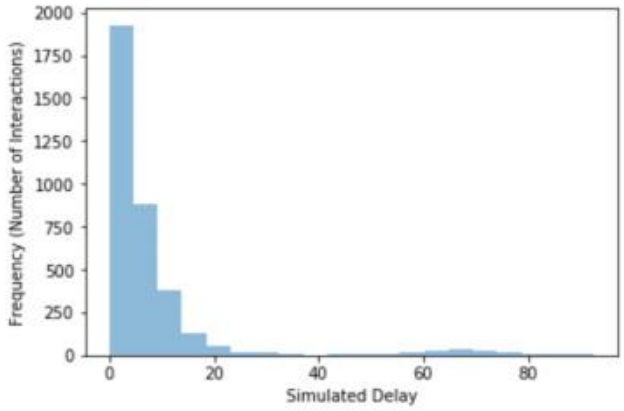


In [103]:

```
num_reps = 5
init_sizes = range(int(2 * feedback_threshold))
summ_sizes = [[] for _ in init_sizes]
for init_size_idx, init_size in enumerate(init_sizes):
    for _ in range(num_reps):
        q = queue.Queue()

        size = init_size
        iits = np.random.exponential(1 / (arrival_rate + service_rate), size)
        t = -sum(iits)
        for iit in iits:
            t += iit
            q.put(t)
        sizes = [size]

        t = 0
        for _ in range(T):
            t += np.random.exponential(1 / (arrival_rate + service_rate))
            if np.random.random() < arrival_rate / (arrival_rate + service_rate)
                q.put(t)
                size += 1
            elif not q.empty():
                if np.random.random() < np.exp(-theta*(t - q.get())):
                    size -= 1
                else:
                    q.put(t)

            sizes.append(size)

        summ_sizes[init_size_idx].append((np.max(sizes), np.min(sizes)))
plt.xlabel('Initial Queue Size (Number of Items)')
plt.ylabel('Extinction Probability')
plt.errorbar(init_sizes, [np.mean([1 if z[1] == 0 else 0 for z in x]) for x in s
plt.axvline(x=feedback_threshold, linestyle='--', label='Feedback Threshold')
plt.legend(loc='best')
plt.show()
```



In [104]:
```
plt.xlabel('Simulated Delay')
plt.ylabel('Frequency (Number of Interactions)')
plt.hist(delays, bins=20, linewidth=0, alpha=0.5)
plt.show()
```



This paper is more similar to the human learning, which make more tailored to the real world application.

V. Reference

1. M/D/1_queue
2. MIT Data communication fa02 Lec5&6
3. CSDN
4. leitnerq