# Question1

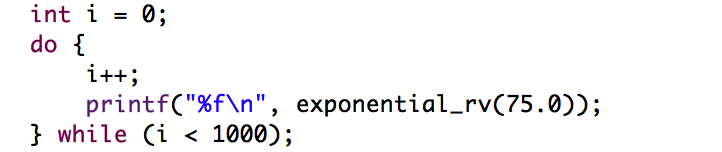
What is the mean and variance of the 1000 random variables you generated?

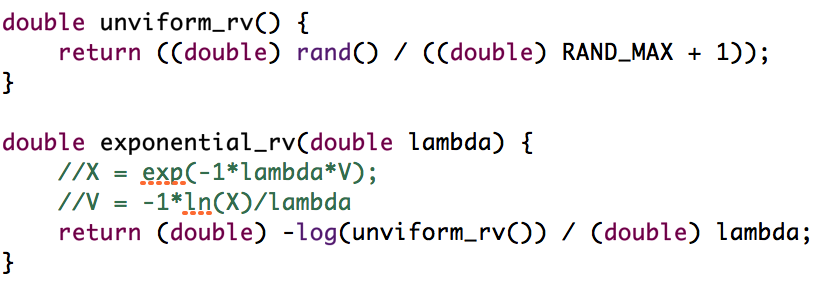
F(x) = exp(-1\*lambda \* x)  
x = -ln(F(x))/lambda

(calculated) => (observed)

Mean: 1/lambda = 0.133- => 0.0128652

Variance: 1/lambda^2 = 0.000177- (calculated) => 0.00017793





# Question 2

To create the simulator I defined a class Simulator which encapsulates the event scheduler (ES) and all of the functional requirements. Each stage of the simulation is performed by a self describing function which accepts the relevant parameters. The re-used parameters: link rate, queue size, duration (T), and average packet size (L) are defined in the constructor to avoid misuse. The event scheduler itself is defined as a list of simEvent structs. These structs are assigned a type (EventType) corresponding to arrivals, departures and observations. The statistic calculations are stored within the main simulator class and are public so as to be accessible when the simulation is complete. As can be seen in the main, the simulator usage is as follows.

Sim::Simulator\* sim = **new** Sim::Simulator(C, queue\_size, T, L);

//Generate observation events, poisson parameter alpha ( if less than T)

sim->generate\_observations(alpha);

//Packet arrival times (parameter lambda)

sim->generate\_arrivals(lambda);

sim->order\_events();

//Find departure times

sim->calculate\_departures();

//packet arrivals so far, packets departures so far

sim->order\_events();

sim->observe\_events();

## Question 2 Code Listing

**namespace** Sim {

**enum** EventType {

*ARRIVAL*, *DEPARTURE*, *OBSERVATION*

};

**struct** simEvent {

EventType type;

**int** id;

**double** packetLength;

**double** time;

**bool** dropped;

} SimEvent;

**bool** **compare\_times**(simEvent& first, simEvent& second) {

**return** first.time < second.time;

}

;

**class** Simulator {

**int** Na; //Arrivals

**int** Nd; //Departures

**int** No; //Observations

**int** linkRate;

**int** queueSize;

**int** durationT;

**int** avgPacketSize;

list<simEvent> ES;

**public**:

/////////STATS/////

**int** num\_observations;

**int** num\_packets;

**double** num\_packets\_in\_buffer;

**double** sojourn\_time;

**double** pIdle;

**double** pLoss;

//////////////////

**Simulator**(**int** linkRate, **int** queueSize, **int** durationT, **int** L) {

**this**->linkRate = linkRate;

**this**->queueSize = queueSize;

**this**->durationT = durationT;

**this**->avgPacketSize = L;

**this**->num\_packets\_in\_buffer = 0;

**this**->sojourn\_time = 0;

**this**->pIdle = 0;

**this**->pLoss = 0;

}

**~Simulator**() {

ES.clear();

}

**void** **generate\_observations**(**int** alpha) {

**double** time = 0;

**int** i = 1;

**while** (**true**) {

simEvent e;

e.type = *OBSERVATION*;

e.id = i;

e.dropped = **false**;

time += exponential\_rv(alpha);

e.time = time;

**if** (e.time > durationT) {

**break**;

}

i++;

ES.push\_back(e);

}

num\_observations = i;

}

**void** **generate\_arrivals**(**int** lambda) {

**double** time = 0;

**int** i = 1;

**while** (**true**) {

simEvent e;

e.type = *ARRIVAL*;

e.id = i;

e.dropped = **false**;

e.packetLength = exponential\_rv(1.0 / (**double**) avgPacketSize);

time += exponential\_rv(lambda);

e.time = time;

**if** (e.time > durationT) {

**break**;

}

i++;

ES.push\_back(e);

}

num\_packets = i;

}

**void** **calculate\_departures**() {

**double** current\_time = 0;

**int** packet\_count = 0;

**for** (list<simEvent>::iterator it = ES.begin(); it != ES.end(); ++it) {

**if** (it->type != *ARRIVAL* || it->dropped) {

**continue**;

}

//How long has this packet been waiting?

//printf("Packet at time %f (delta)%f: id %d size: %f \n", it->time, (current\_time - it->time), it->id, it->packetLength);

**if** (it->time > current\_time) {

current\_time = it->time;

}

//add departure event at

**double** departureTime = (**double**) current\_time

+ (it->packetLength / (**double**) linkRate);

**if** (queueSize > 0) {

//if finite queue, mark dropped packets by considering the number left in the queue while servicing

//count # events between current time and departureTime, if > queueSize, drop packet

**int** numQueued = 0;

**for** (list<simEvent>::iterator dq = it; dq != ES.end(); ++dq) {

**if** (dq->type != *ARRIVAL* || dq->dropped) {

//only check arrivals and packets that have not yet been dropped

**continue**;

}

**if** (dq->time > departureTime) {

**break**;

//We have moved beyond the affected interval

}

**if** (numQueued == queueSize) {

//Start dropping packets

//printf("Packet id %d dropped (time:%f)\n", dq->id, dq->time);

dq->dropped = **true**;

pLoss++;

} **else** {

numQueued++;

}

}

}

//add departure event

simEvent e;

e.id = it->id;

e.type = *DEPARTURE*;

e.time = departureTime;

//printf("Packet departs at time %f id %d \n", e.time, e.id);

ES.push\_back(e);

sojourn\_time += it->time;

packet\_count++;

//time advances to take new packet

current\_time = departureTime;

}

//average sojourn time

sojourn\_time /= (**double**) packet\_count;

pLoss /= (**double**) num\_packets;

}

**void** **order\_events**() {

ES.sort(compare\_times);

}

**void** **observe\_event**(simEvent\* se) {

**switch** (se->type) {

**case** *ARRIVAL*:

**if** (!se->dropped) {

Na++;

}

**break**;

**case** *DEPARTURE*:

Nd++;

**break**;

**case** *OBSERVATION*:

No++;

num\_packets\_in\_buffer += Na - Nd;

**if** (Na == Nd) {

pIdle++;

}

//printf("Time: %f No %d Na %d Nd %d\n", se->time, No, Na, Nd);

**break**;

}

}

**void** **observe\_events**() {

Na = 0;

Nd = 0;

No = 0;

**for** (list<simEvent>::iterator it = ES.begin(); it != ES.end(); ++it) {

observe\_event(&\*it);

}

pIdle /= num\_observations;

num\_packets\_in\_buffer /= (**double**) num\_observations;

}

};

}

**int** **main**(**void**) {

Sim::Simulator\* sim = **new** Sim::Simulator(C, queue\_size, T, L);

//Generate observation events, poisson parameter alpha ( if less than T)

sim->generate\_observations(alpha);

//Packet arrival times (parameter lambda)

sim->generate\_arrivals(lambda);

sim->order\_events();

//Find departure times

sim->calculate\_departures();

//packet arrivals so far, packets departures so far

sim->order\_events();

sim->observe\_events();

**printf**("%f\t%d\t%f\t%f\t%f\t%f\t%f\t%f\t\n", p, T, lambda, alpha,

sim->num\_packets\_in\_buffer, sim->sojourn\_time, sim->pIdle,

sim->pLoss);

//number of observations

**delete** sim;

}

}

**return** EXIT\_SUCCESS;

}

;