ECE 358 Networking Lab 1

# Question 1

The equation of PDF for the exponential random variable is the following.

In CDF, the equation changes to the following.

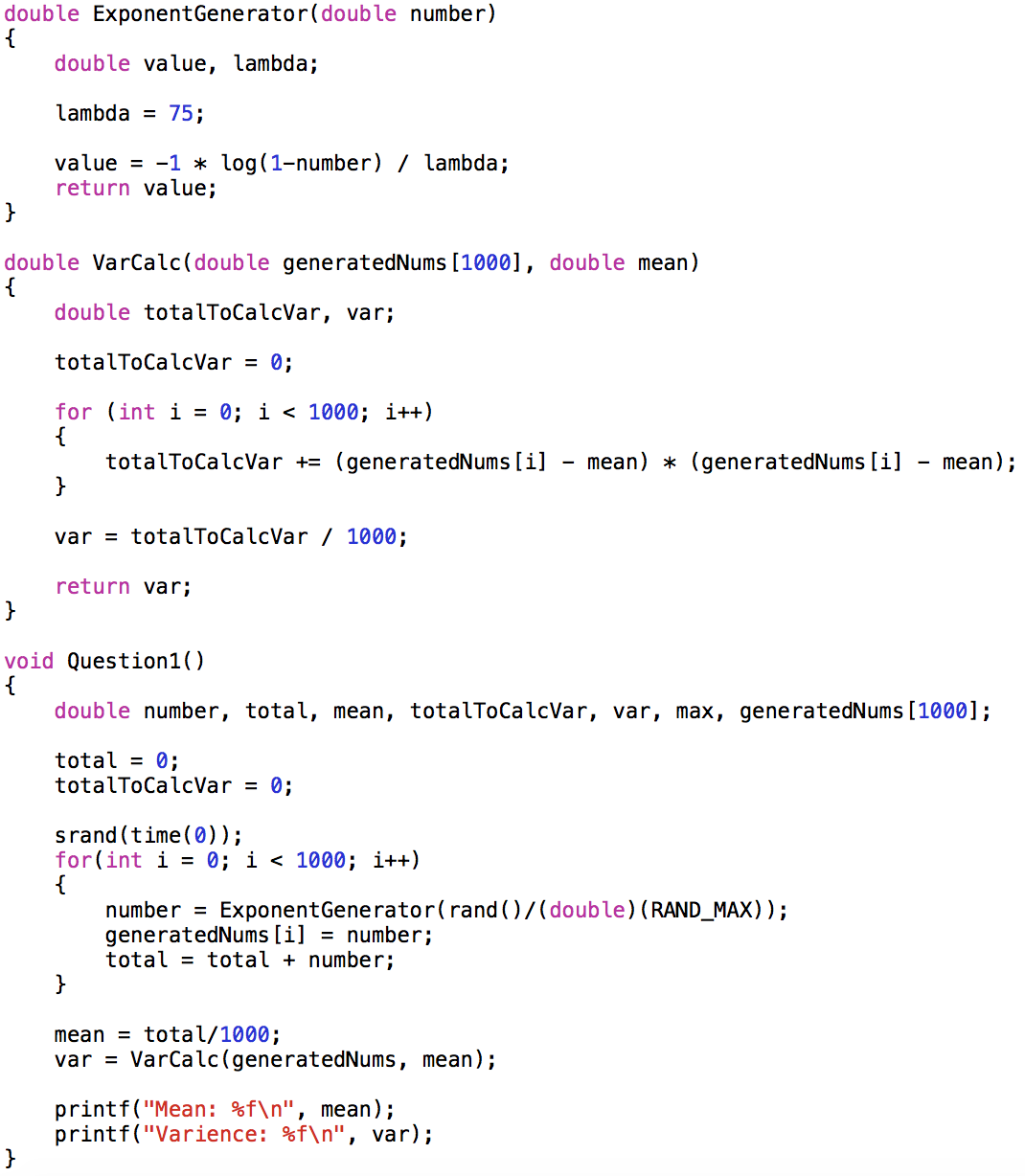


Figure 1. Exponential Generator



Figure 2. Mean and Variance of the Generator

I agree with the mean and variance. The expected mean of the exponential generator with is , which is 0.01333. The percent error is 2.82%. The expected variance of the exponential generator with is , which is . The percent error is 1.8123%.

# Question 2

enum EventType {a, d, o};

struct event

{

EventType TypeOfPacket;

double GenerationTime;

bool operator<(const event& rhs) const

{

return GenerationTime > rhs.GenerationTime;

}

bool operator>(const event& rhs) const

{

return GenerationTime < rhs.GenerationTime;

}

};

class q2

{

public:

q2 (int numberOfPackets);

void GenerateArrivalPackects(), GenerateObserverPackets(), GenerateDeparturePackets(), SortDES();

long Np(), No(), Ni();

~q2();

private:

double ExponentialRandomGenerator(double x);

long \_Na, \_No, \_Nd, \_Ni, \_Np, \_L, \_C, \_Time;

double \_Row, \_Lambda;

priority\_queue<event> \_DES;

};

q2::q2(int timeElapsed)

{

\_Na = 0;

\_Nd = 0;

\_No = 0;

\_Ni = 0;

\_Np = 0;

\_L = 12000;

\_C = 1000000;

\_Row = 20;

\_Lambda = \_Row \* (double)\_C / (double)\_L;

\_Time = timeElapsed;

srand(time(NULL));

}

void q2::GenerateArrivalPackects()

{

cout << "Beginning of Generation of Arrival Packets" << endl;

event arrival;

double ta = 0.0;

while (ta < \_Time)

{

ta += ExponentialRandomGenerator(\_Lambda);

arrival.TypeOfPacket = a;

arrival.GenerationTime = ta;

\_DES.push(arrival);

}

}

void q2::GenerateObserverPackets()

{

cout << "Beginning of Observer Packets" << endl;

event observer;

double alpha, to;

alpha = 3\*\_Lambda;

to = 0.0;

while (to < \_Time)

{

to += ExponentialRandomGenerator(alpha);

observer.TypeOfPacket = o;

observer.GenerationTime = to;

\_DES.push(observer);

}

}

void q2::SortDES()

{

cout << "Beginnig of SortDES" << endl;

event departure;

double packetLength, serviceTime, td;

td = 0.0;

while (!\_DES.empty())

{

if (\_DES.top().TypeOfPacket == a)

{

packetLength = ExponentialRandomGenerator(1.0/\_L);

serviceTime = packetLength/(double)\_C;

if (\_Na - \_Nd == 0)

{

td = \_DES.top().GenerationTime + serviceTime;

}

else

{

td += serviceTime;

}

departure.TypeOfPacket = d;

departure.GenerationTime = td;

\_DES.push(departure);

\_Na++;

}

else if (\_DES.top().TypeOfPacket == d)

{

\_Nd++;

}

else if (\_DES.top().TypeOfPacket == o)

{

\_No++;

\_Np += \_Na - \_Nd;

if (\_Na - \_Nd == 0)

{

\_Ni++;

}

}

\_DES.pop();

}

}

int main()

{

int timeElapsed;

timeElapsed = 10000;

q2 \* question2;

question2 = new q2(timeElapsed);

question2->GenerateArrivalPackects();

question2->GenerateObserverPackets();

question2->SortDES();

cout << "Average # of packets: " << (double)(question2 -> Np())/(double)(question2 -> No()) << endl;

cout << "Idle Probability: " << (double)(question2 -> Ni())/(double)(question2 -> No()) << endl;

question2 = NULL;

delete question2;

return 0;

}

First, I defined the struct of the event. In the event, there is enum, which defines the event time, and generation time, which specifies the timing the event, occurred. Then in the main function, the q2 class is initiated. Many variables are defined. They are

* \_Na to keep track of the number of arrival events,
* \_Nd to keep track of the number of departure events,
* \_No to keep track of the number of observer events,
* \_Ni to keep track of the number of idle moment,
* \_Np to keep track of the number of packets in the system counter,
* \_L, \_C, \_Row, \_Lambda constants, and
* \_Time, which will be time elapsed when running the program.

Times are declared in each function. Examples are

* ta for arrival event time,
* to for observer event time, and
* td for departure event time.

Then arrival packets are generated. After that, observer packets are generated. One thing to note is that for every arrival event, approximately 3 more observing event is generated. Then, through SortDES subroutine, departure packet is added and the records are kept for each observing event as the event in DES priority\_queue. Lastly, in main function, average number of packets in the system and the idle probability are calculated.

# Question 3

|  |  |  |
| --- | --- | --- |
| Ρ | Average # of Packets | Idle Probability |
| 0.25 | 0.331913 | 0.750841 |
| 0.35 | 0.532471 | 0.652414 |
| 0.45 | 0.818407 | 0.549584 |
| 0.55 | 1.2292 | 0.448846 |
| 0.65 | 1.86127 | 0.349281 |
| 0.75 | 2.99512 | 0.248603 |
| 0.85 | 5.6238 | 0.15201 |
| 0.95 | 19.3687 | 0.0498583 |

For each p value, I changed \_row and compiled the program. Then I copied the values into the table.

The results make sense. As rho increases, more events are produced. It means that more packets are in the system. Therefore, the system will be busier processing the packets. So the idle probability goes down and more and more packets wait for their turn to be processed since the system’s processing power does not increase. The processing time is constant, where as more events are generated.

# Question 4

If rho = 1.2, the idle probability is 9.99553e-07 and the average number of packets in the system is 83769.3. This indicates that the system is at its limit that the system is always busy and many of the packets are stored in the buffer. If the buffer is limited, many packets will be dropped.

# Question 5

class q2

{

public:

q2 ();

q2 (int numberOfPackets, double row, long sizeofbuffer);

void GenerateArrivalPackects(), GenerateObserverPackets(), GenerateDeparturePackets(), SortDES();

long Np(), No(), Ni(), Ngen(), Nloss();

~q2();

private:

double ExponentialRandomGenerator(double x);

void init(int timeElapsed, double row, long sizeofbuffer);

long \_Na, \_No, \_Nd, \_Ni, \_Np, \_L, \_C, \_Time, \_Ngen, \_Nloss, \_k;

double \_Row, \_Lambda;

priority\_queue<event> \_DES;

};

void q2::init(int timeElapsed, double row, long sizeofbuffer)

{

\_Row = row;

\_k = sizeofbuffer;

\_Time = timeElapsed;

\_Na = 0;

\_Nd = 0;

\_No = 0;

\_Ni = 0;

\_Np = 0;

\_Ngen = 0;

\_Nloss = 0;

\_L = 12000;

\_C = 1000000;

\_Lambda = \_Row \* (double)\_C / (double)\_L;

srand(time(NULL));

}

The codes are mostly similar to the last example codes. However, there are some differences. First is the addition of the following variables on top of the variables from question 2:

* \_Ngen to keep track of the number of generated packets,
* \_Nloss to keep track of the number of lost packets, and
* \_k for the buffer size. If k = 0, it is infinite buffer.

Also, get the following get functions are added:

* Ngen() to get the value of \_Ngen and
* Nloss() to get the value of \_Nloss.

while (!\_DES.empty())

{

if (\_DES.top().TypeOfPacket == a)

{

\_Ngen++;

if (\_k == 0 || \_Na - \_Nd < \_k)

{

packetLength = ExponentialRandomGenerator(1.0/\_L);

serviceTime = packetLength/(double)\_C;

if (\_Na - \_Nd == 0)

{

td = \_DES.top().GenerationTime + serviceTime;

}

else

{

td += serviceTime;

}

departure.TypeOfPacket = d;

departure.GenerationTime = td;

\_DES.push(departure);

\_Na++;

}

else

{

\_Nloss++;

}

}

SortDES() went much change. (Some part of the function is redundant, so it has been skipped in the report because of the formatting issue.) The highlighted lines are the changes implemented in the question 4. \_Ngen and \_Nloss are now implemented to check the generated and dropped packets. The if-condition has been added so that it checks whether the buffer is full or not. When the number of arrival packets minus the number of lost packets are greater than \_k (meaning the buffer is full), then the event is lost and \_Nloss is increased by 1.

int main()

{

int timeElapsed;

timeElapsed = 10000;

q2 \* question2;

question2 = new q2(timeElapsed, 0.5, 5);

question2->GenerateArrivalPackects();

question2->GenerateObserverPackets();

question2->SortDES();

cout << "Average # of Packets: " << (double)(question2 -> Np())/(double)(question2 -> No()) << endl;

cout << "Idle Probability: " << (double)(question2 -> Ni())/(double)(question2 -> No()) << endl;

cout << "Number of Lost Packets: " << question2 -> Nloss() << endl;

cout << "Number of Generated Packets: " << question2 -> Ngen() << endl;

question2 = NULL;

delete question2;

return 0;

}

Now the main function displays the number of packets lost and generated.

# Question 6

## Part 1

|  |  |
| --- | --- |
| Buffer of inf |  |
| Rho Value | Average # of Packets |
| 0.5 | 1.00235 |
| 0.6 | 1.50172 |
| 0.7 | 2.31705 |
| 0.8 | 4.01964 |
| 0.9 | 8.99023 |
| 1 | 702.289 |
| 1.1 | 41533.3 |
| 1.2 | 82667.7 |
| 1.3 | 124738 |
| 1.4 | 167631 |
| 1.5 | 207198 |
|  |  |
| Buffer of 5 |  |
| Rho Value | Average # of Packets |
| 0.5 | 0.90515 |
| 0.6 | 1.20855 |
| 0.7 | 1.53605 |
| 0.8 | 1.87365 |
| 0.9 | 2.19812 |
| 1 | 2.50116 |
| 1.1 | 2.77702 |
| 1.2 | 3.01808 |
| 1.3 | 3.23726 |
| 1.4 | 3.41559 |
| 1.5 | 3.58197 |
|  |  |
| Buffer of 10 |  |
| Rho Value | Average # of Packets |
| 0.5 | 0.994901 |
| 0.6 | 1.46222 |
| 0.7 | 2.09388 |
| 0.8 | 2.9625 |
| 0.9 | 3.96491 |
| 1 | 4.97961 |
| 1.1 | 5.95256 |
| 1.2 | 6.70259 |
| 1.3 | 7.31872 |
| 1.4 | 7.77429 |
| 1.5 | 8.13822 |
|  |  |
| Buffer of 15 |  |
| Rho Value | Average # of Packets |
| 0.5 | 1.00377 |
| 0.6 | 1.50741 |
| 0.7 | 2.24793 |
| 0.8 | 3.5831 |
| 0.9 | 5.3596 |
| 1 | 7.50862 |
| 1.1 | 9.39519 |
| 1.2 | 10.9596 |
| 1.3 | 11.9202 |
| 1.4 | 12.5656 |
| 1.5 | 13.0315 |

There is a rapid increase when the rho is greater than 1. For the buffer of 5, it was already at max, so it looks like it is a linear increase. However, for the buffer of 15, it rapidly stores more packets before entering into a plateau. The limit as the rho increases is its buffer size, since that is the maximum number of packet the server can store. For the infinite size of buffer (which is not included in the graph due to scaling issue) increases will increase infinitely without the limit as rho approaches infinite. (There are infinite numbers of packets that can be store in such a system.)

## Part 2

|  |  |
| --- | --- |
| Buffer of 5 |  |
| Rho Value | Average # of Packets |
| 0.4 | 0.00638115 |
| 0.5 | 0.015666 |
| 0.6 | 0.0313931 |
| 0.7 | 0.0585767 |
| 0.8 | 0.0883978 |
| 0.9 | 0.126741 |
| 1 | 0.166495 |
| 1.1 | 0.208255 |
| 1.2 | 0.250517 |
| 1.3 | 0.292387 |
| 1.4 | 0.328859 |
| 1.5 | 0.365851 |
| 1.6 | 0.398446 |
| 1.7 | 0.431001 |
| 1.8 | 0.458225 |
| 1.9 | 0.483487 |
| 2 | 0.506961 |
| 2.2 | 0.550246 |
| 2.4 | 0.586142 |
| 2.6 | 0.61753 |
| 2.8 | 0.64414 |
| 3 | 0.667377 |
| 3.2 | 0.688775 |
| 3.4 | 0.706256 |
| 3.6 | 0.722531 |
| 3.8 | 0.737182 |
| 4 | 0.749876 |
| 4.2 | 0.761613 |
| 4.4 | 0.773127 |
| 4.6 | 0.782688 |
| 4.8 | 0.791381 |
| 5 | 0.799875 |
| 5.4 | 0.814712 |
| 5.8 | 0.827562 |
| 6.2 | 0.838772 |
| 6.6 | 0.848603 |
| 7 | 0.857257 |
| 7.4 | 0.864616 |
| 7.8 | 0.871863 |
| 8.2 | 0.878009 |
| 8.6 | 0.883665 |
| 9 | 0.888903 |
| 9.4 | 0.893678 |
| 9.8 | 0.897955 |
|  |  |
| Buffer of 10 |  |
| Rho Value | Average # of Packets |
| 0.4 | 7.21E-05 |
| 0.5 | 0.00044357 |
| 0.6 | 0.00225604 |
| 0.7 | 0.00875278 |
| 0.8 | 0.0238192 |
| 0.9 | 0.0508898 |
| 1 | 0.0911412 |
| 1.1 | 0.140737 |
| 1.2 | 0.193094 |
| 1.3 | 0.243552 |
| 1.4 | 0.294274 |
| 1.5 | 0.337777 |
| 1.6 | 0.378256 |
| 1.7 | 0.41105 |
| 1.8 | 0.445793 |
| 1.9 | 0.473169 |
| 2 | 0.500692 |
| 2.2 | 0.5453 |
| 2.4 | 0.583626 |
| 2.6 | 0.615176 |
| 2.8 | 0.644049 |
| 3 | 0.6663 |
| 3.2 | 0.686802 |
| 3.4 | 0.705821 |
| 3.6 | 0.721657 |
| 3.8 | 0.736914 |
| 4 | 0.74997 |
| 4.2 | 0.762278 |
| 4.4 | 0.772272 |
| 4.6 | 0.783026 |
| 4.8 | 0.791355 |
| 5 | 0.800003 |
| 5.4 | 0.814515 |
| 5.8 | 0.827818 |
| 6.2 | 0.83864 |
| 6.6 | 0.848445 |
| 7 | 0.857316 |
| 7.4 | 0.864711 |
| 7.8 | 0.871683 |
| 8.2 | 0.878035 |
| 8.6 | 0.883737 |
| 9 | 0.888964 |
| 9.4 | 0.8935 |
| 9.8 | 0.897975 |
|  |  |
| Buffer of 20 |  |
| Rho Value | Average # of Packets |
| 0.4 | 0 |
| 0.5 | 0 |
| 0.6 | 4.00E-06 |
| 0.7 | 0.000161007 |
| 0.8 | 0.00245945 |
| 0.9 | 0.0137438 |
| 1 | 0.0481775 |
| 1.1 | 0.104187 |
| 1.2 | 0.170434 |
| 1.3 | 0.232587 |
| 1.4 | 0.286941 |
| 1.5 | 0.334181 |
| 1.6 | 0.375505 |
| 1.7 | 0.411759 |
| 1.8 | 0.443563 |
| 1.9 | 0.473009 |
| 2 | 0.500341 |
| 2.2 | 0.545071 |
| 2.4 | 0.58286 |
| 2.6 | 0.615754 |
| 2.8 | 0.642584 |
| 3 | 0.666158 |
| 3.2 | 0.687521 |
| 3.4 | 0.705811 |
| 3.6 | 0.72211 |
| 3.8 | 0.736771 |
| 4 | 0.749985 |
| 4.2 | 0.761831 |
| 4.4 | 0.772713 |
| 4.6 | 0.782435 |
| 4.8 | 0.791789 |
| 5 | 0.799806 |
| 5.4 | 0.814893 |
| 5.8 | 0.827207 |
| 6.2 | 0.838806 |
| 6.6 | 0.848585 |
| 7 | 0.857074 |
| 7.4 | 0.864882 |
| 7.8 | 0.871806 |
| 8.2 | 0.877947 |
| 8.6 | 0.884021 |
| 9 | 0.888871 |
| 9.4 | 0.893482 |
| 9.8 | 0.897915 |
|  |  |
| Buffer of 40 |  |
| Rho Value | Average # of Packets |
| 0.4 | 0 |
| 0.5 | 0 |
| 0.6 | 0 |
| 0.7 | 0 |
| 0.8 | 1.35E-05 |
| 0.9 | 0.00136967 |
| 1 | 0.0230063 |
| 1.1 | 0.0919283 |
| 1.2 | 0.166971 |
| 1.3 | 0.230473 |
| 1.4 | 0.285437 |
| 1.5 | 0.333663 |
| 1.6 | 0.374852 |
| 1.7 | 0.412715 |
| 1.8 | 0.443458 |
| 1.9 | 0.473452 |
| 2 | 0.500905 |
| 2.2 | 0.545798 |
| 2.4 | 0.584402 |
| 2.6 | 0.6147 |
| 2.8 | 0.642708 |
| 3 | 0.666941 |
| 3.2 | 0.687487 |
| 3.4 | 0.706212 |
| 3.6 | 0.722358 |
| 3.8 | 0.737232 |
| 4 | 0.750246 |
| 4.2 | 0.761808 |
| 4.4 | 0.772747 |
| 4.6 | 0.78252 |
| 4.8 | 0.791157 |
| 5 | 0.799958 |
| 5.4 | 0.814673 |
| 5.8 | 0.82808 |
| 6.2 | 0.838487 |
| 6.6 | 0.848497 |
| 7 | 0.857318 |
| 7.4 | 0.864934 |
| 7.8 | 0.871999 |
| 8.2 | 0.878538 |
| 8.6 | 0.883683 |
| 9 | 0.888817 |
| 9.4 | 0.89358 |
| 9.8 | 0.897941 |

Ploss is obtained by dividing Nloss by Ngen. Those are obtained in SortDES, when departure event is created. The system with bigger buffer is less likely to lose packets when rho is small. However, when rho is greater than one, more and more packets will be lost, until the ratio approaches to 1.