Measuring Performance

Measure collection and analysis

- The scope of a simulation is to evaluate the system performance
- The performance indexes we are interested in vary according to the situation, but they are typically represented by numeric quantities measured inside the program
- The key activity of our simulation is to **collect samples** of those numeric quantities to later proceed to the evaluation of our indexes

How to measure averages?

- We need to measure different types of averages, depending on the involved quantities
  - Point averages
  - Time averages
  - Statistical frequencies

What do we want to measure?

- **Averaged quantities (instances of a r.v.)**
  - Point averages: Average of variables that are taken at *discrete time* instants (average delay in the queue per customer, average service time,…)
  - Time averages: Average of *continuous-time* variables (average no. of customers, average utilization of the servers, …)
- Besides averages, we are often interested on a **complete statistical characterization**
  - e.g. variance, standard deviation, quantiles, …
- **Statistical frequencies**
  - How often a given condition is observed/occurs

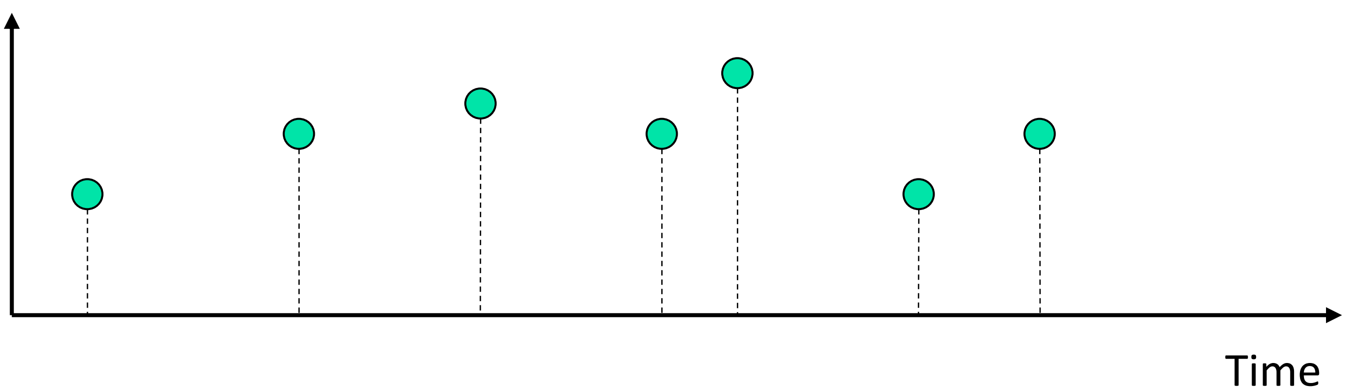
Measuring averages

- Measuring averages requires a lot of attention
- In our simulator we have two possibilities:
  - collect all the samples of the quantities we are interested in, saving them outside the simulator (in a file) and leaving the statistical analysis of our quantities to an external dedicated program
  - perform inside the program the needed mathematical operations to produce the desired outputs (typically the average quantities)
- Due to the length and complexity of our simulations, the latter possibility is the one usually adopted

Point averages

- They are used for quantities that are sampled at discrete time instants; i.e., for which we have a finite number of samples

$$\hat{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$



Point averages

- In the code
  - We accumulate the samples x_i and the no. of samples

```
total += sample
nr_samples++
```

  - At the end of the simulation we compute

```
mean = total/nr_samples
```

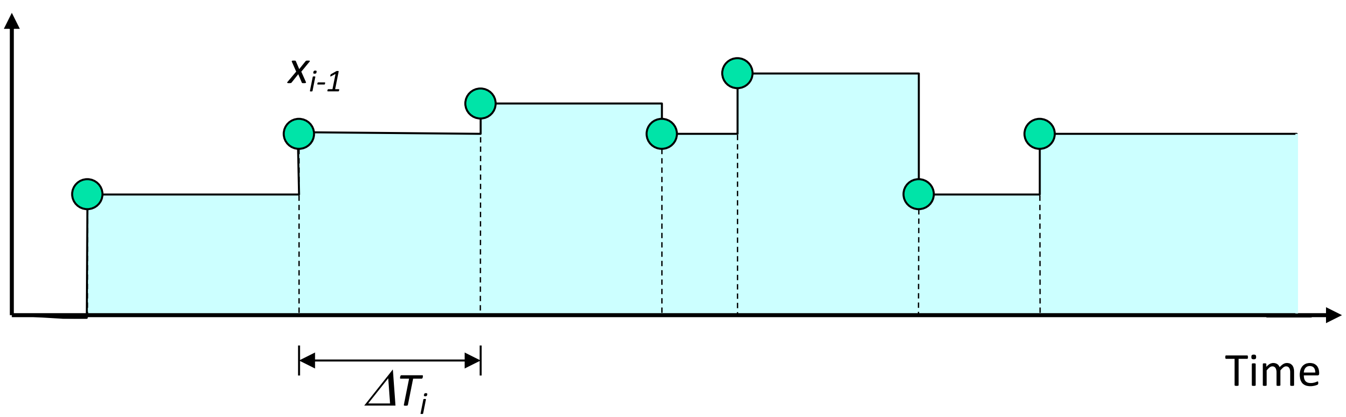
- In a similar way we can compute the variance using the estimator

$$\hat{\sigma}^2 = \frac{1}{N-1}\left(\sum_{i=1}^{N} x_i^2 - N \cdot \hat{x}^2\right)$$

Time averages

- They are used for quantities that take a value in any (continuous) time instant
  - Both the value and the duration of time for which that value holds have to be taken into account

$$\hat{\mu}_x = \lim_{T \to \infty} \frac{1}{T} \int_0^T x(t)dt$$



Time averages

- In the code
  - We accumulate the areas $\Delta T_i \cdot x_{i-1}$

```
delta_time = current_time – last_time % time between events
total_area += old_value*delta_time % update the integral
old_value = current_value % needed at next event
last_time = current_time
```

  - At the end of the simulation we compute

```
mean = total_area / final_time
```

- The evaluation is slightly more complex since computations are done at the next sample

Time averages

- Time averages can be obtained also exploiting properties of Poisson Point Processes:
  - Sequence $\{T_n\}_n$ forms a PPP iff $X_n = T_{n+1} - T_n$ are i.i.d and exponentially distributed
- It can be proved that:

$$\hat{\mu}_{\mathrm{x}} = \lim_{T\to\infty} \frac{1}{T}\int_0^T x(t)dt = \lim_{T\to\infty} \frac{\sum_1^n x(T_i)}{n}$$

  - provided that $\{T_n\}_n$ is independent from system dynamics

Statistical frequences

- We use them to evaluate probabilities of some specific situation to occur
- Again we distinguish between discrete- and continuous-time variables
  - Discrete-time variables: we relate the number of occurrences of the situation of interest with the total number of potential occurrences
  - Continuous-time variables: we relate the duration of time for which a specific situation occurs over the total time the situation could have occurred

Statistical frequences

- Discrete-time variables:

$$\hat{p}_i = \frac{N_i}{N}$$

N_i ← Tot. no. of times that a situation occurred (e.g., a packet was lost)

N ← Tot. no. of times that a situation could have occurred (e.g., tot no. packet)

- In the code
  - We count the favorable samples

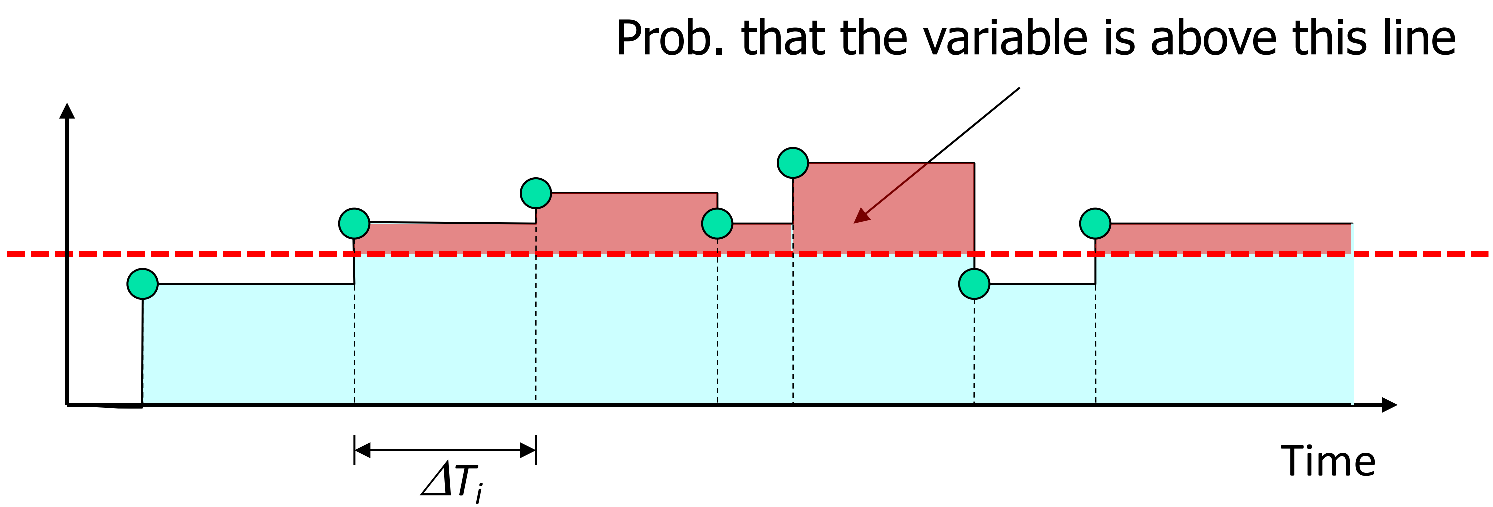
```
if (condition):
        nr_good +=1
nr_samples +=1
```

  - At the end of the simulation we compute

```
prob = nr_good / nr_samples
```

Statistical frequences

- Continuous-time variables:
  - Sum all the periods of time in which the condition is true and divide it by the total time



Statistical frequences

- In the code
  - We count the "favorable" periods; i.e., we accumulate the areas ΔT_i

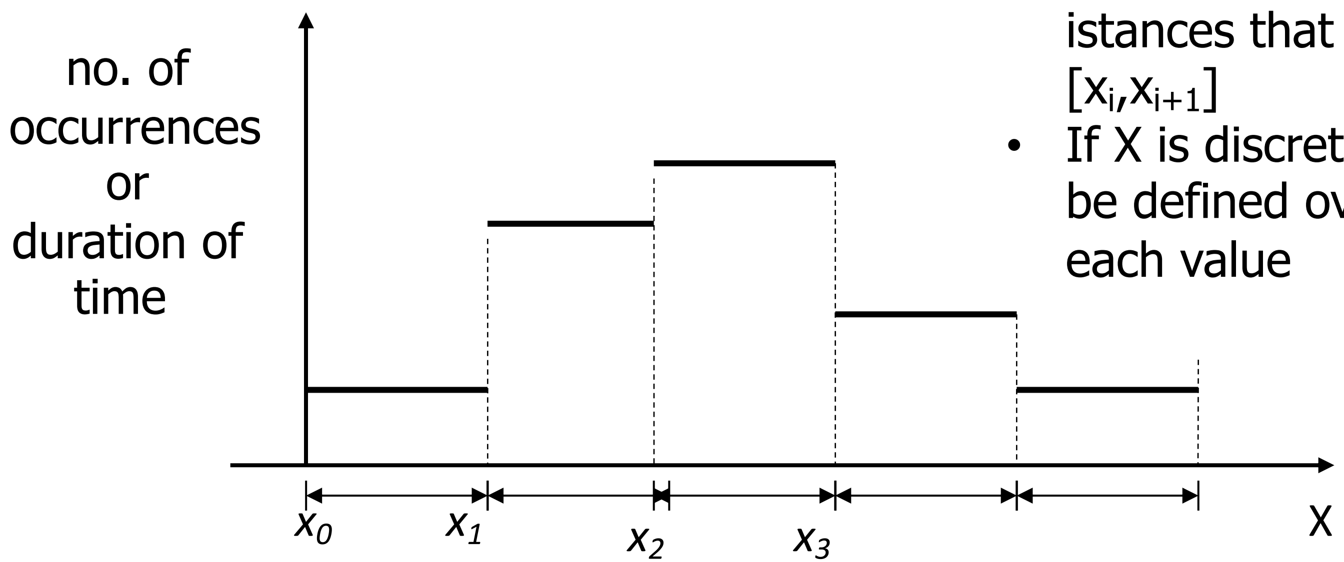
```
if (condition):
        delta_time = current_time – last_time # time between events
        total_time += delta_time # update the
        last_time = current_time
```

  - At the end of the simulation we compute

```
mean = total_time / final_time
```

Histograms and quantiles

- Histograms and quantiles are derived similarly to probabilities discussed above by defining
  - Number and size of the bins of possible values of the variable to be estimated



- If X is continuous count no. of istances that fall in the intervals $[x_i,x_{i+1}]$
- If X is discrete, the histograms can be defined over intervals or over each value

Caveat

- Our computations are, at the moment, we get only an unreliable estimates of the quantities we want to measure
- We are not considering
  - the effect of the initial conditions
  - the effect of the transient
  - the fact we are measuring a single specific realization of the stochastic process representing the simulated system
- We will learn later the correct procedures to measure quantities with suitable confidence on their accuracy

Initialization

- Before starting the simulation (entering the Event Loop) we must
  - initialize all the variables used to store our measures
  - initialize the data structures needed for the simulation
  - assign a value to all the simulation parameters, possibly through user inputs
  - bootstrap the simulation, scheduling the first few events

Termination

- At the end of the Event Loop, before exiting the program, we must inform the users of the simulation results
- If the simulation ended regularly, we print (on the screen and/or on a file) all the measures collected and any information it is important to report
- If the simulation ended anomalously, we will explicitly report this fact, to avoid the possible usage of meaningless data

Example: queue

```
# ****************************************
# To take the measurements
# ****************************************
class Measure:
    def __init__(self,Narr,Ndep,NAveraegUser,OldTimeEvent,AverageDelay):
        self.arr = Narr # count the no. of clients that have arrived
        self.dep = Ndep  # count the no. of clients that have left the queue
        self.ut = NAveraegUser # count the time average of no. of clients
        self.oldT = OldTimeEvent
        self.delay = AverageDelay
# ****************************************
# Initialization
# ****************************************
data = Measure(0,0,0,0,0) # variables for the measures
time = 0 # the simulation time
users = 0 # state variable
FES = PriorityQueue() # the list of events in the form: (time, type)
FES.put((0, "arrival")) # schedule the first arrival
```

Example: queue

```
def arrival(time, FES, queue):
    global users

    # cumulate statistics
    data.arr += 1
    data.ut += users*(time-data.oldT)
    data.oldT = time

    # sample the time until the next event
    inter_arrival = random.expovariate(1.0/ARRIVAL)

    # schedule the next arrival
    FES.put((time + inter_arrival, "arrival"))
    users += 1

    # create a record for the client
    client = Client(TYPE1,time)

 --- CONTINUE
```

Time average

Example: queue

```
CONTINUE ---
    ...

    # insert the record in the queue
    queue.append(client)

    # if the server is idle start the service
    if users==1:

        # sample the service time
        service_time = random.expovariate(1.0/SERVICE)

        # schedule when the client will finish the server
        FES.put((time + service_time, "departure"))
```

Example: queue

```
def departure(time, FES, queue):
    global users

    # cumulate statistics
    data.dep += 1
    data.ut += users*(time-data.oldT)
    data.oldT = time

    # get the first element from the queue
    client = queue.pop()

    data.delay += (time-client.arrival_time)
    users -= 1
```

Point average

--- CONTINUE

Example: queue

```
# see whether there are more clients to in the line
if users >0:
    # sample the service time
    service_time = random.expovariate(1.0/SERVICE)

    # schedule when the client will finish the server
    FES.put((time + service_time, "departure"))
```

Example: queue

```
# At the end of the simulation

# Compute the average time in the queue
average_delay = data.delay/data.dep

# Compute the average no. of customers in the queue
average_no_cust = data.ut/time
```

Simulation!

- A run of the program corresponds to a single realization of the stochastic process representing the simulated system, i.e., to a single point in the space of the possible results
- This is not enough: to characterize and study the system we want to explore a large portion of the results' space
- We plan a simulation campaign!

Simulation!

- To execute a simulation campaign we should:
  - distinguish the input parameters between:
    - **fixed parameters**, whose influence is not a subject of interest in the specific campaign
    - **varying parameters**, whose effect on the system performance indexes is the subject of simulation
  - identify the **output parameters** whose variation we are interested in
  - execute simulation runs for each meaningful combination of the varying input parameters
  - aggregate and represent the output data in a way that explicates their dependence on the varying input parameters

Simulation!

- Varying input parameters
  - For each parameter we must define a variation range and the number of values in such a range
  - Simulation runs for each combination of values of the parameters
    - For each combination, several independent runs to reduce the effect of starting conditions and the pseudo-random sequences
    - Complexity increases exponentially with the number of parameters: don’t exaggerate!
- Representation of the output results
  - We need to select the best way to present our results -> families of parametric plots
  - Possibly use a specific tool for the graphs

Wrap-up

- Results of a simulation are statistical measures of the performance of the system, obtained through the stochastic processes represented by the simulation model
- Measuring performance requires to collect statistics about the values taken by state variables
- The way we collect statistics depends on the kind of variables
  - Discrete-time variables: sampled at discrete time instants
  - Continuous-time variables: that take a values in any instant
  - Frequencies of particular conditions/situations