Simulation

Where real stuff starts
ToC

1. What, transience, stationarity
2. How, discrete event, recurrence
3. Accuracy of output
4. Monte Carlo
5. Random Number Generators
6. How to sample
1 What is a simulation?

- An experiment in computer
- Important differences
  - Simulated vs real time
  - Serialization of events
A simulation can be terminating or not

Example 6.2: Joe’s Computer Shop. We are interested in evaluating the time it takes to serve \( n \) customers who request a file together at time 0. We run a simulation program that terminates at time \( T_1 \) when all users have their request satisfied. This is a terminating simulation; its output is the time \( T_1 \).

For a terminating simulation you should make sure it is stationary

Example 6.3: Information Server. An information server is modelled as a queue. The simulation program starts with an empty queue. Assume the arrival rate of requests is smaller than the server can handle. Due to the fluctuations in the arrival process, we expect some requests to be held in the queue, from time to time. After some simulated time, the queue starts to oscillate between busy periods and idle periods. At the beginning of the simulation, the behaviour is not typical of the stationary regime, but after a short time it becomes so (Figure 6.1 (a)).
Information server, scenario 1

(a) Utilization = 0.96
Information server, scenario 2

M/M/1 Queue, $\mu_F = 0.0101, \mu_G = 0.01$

(b) Utilization = 1.01
Stationarity

- In scenario 1:
  - Transient phase, followed by “typical” (=stationary) phase
  - You want to measure things only in the stationary phase
    - Otherwise: non reproduceable, non typical

- In scenario 2:
  - There is no stationary regime
    “walk to infinity”

you should not do a non terminating simulation with this scenario
Definition of Stationarity

- A property of a stochastic model \( X_t \)
- Let \( X_t \) be a stochastic model that represents the state of the simulator at time \( t \). It is **stationary** iff

  for any \( s > 0 \)

  \( (X_{t1}, X_{t2}, ..., X_{tk}) \) has same distribution as \( (X_{t1+s}, X_{t2+s}, ..., X_{tk+s}) \)

- i.e. simulation does not get “old”
Classical Cases

Markov models
- State $X_t$ is sufficient to draw the future of the simulation
- Common case for all simulations

For a Markov model, over a discrete state space
- If you run the simulation long enough it will either walk to infinity (unstable) or converge to stationary
  - Ex: queue with $\rho > 1$: unstable
  - queue with $\rho < 1$: becomes stationary after transient
- If the state space is strongly connected (any state can be reached from any state) then there is 0 or 1 stationary regime
  - Ex: queue
- Else, there may be several distinct stationary regimes
  - Ex: system with failure modes
Stationarity and Transience

Knowing whether a model is stationary is sometimes a hard problem
  ► We will see important models where this is solved
  ► Ex: Solved for single queues, not for networks
  ► Ex: time series models

Reasoning about your system may give you indications
  ► Do you expect growth?
  ► Do you expect seasonality?

Once you believe your model is stationary, you should handle transients
  ► Remove (how? Look at your output and guess)
  ► Sometimes it is possible to avoid transients at all (perfect simulation)
Non-Stationary (Time Dependent Inputs)
Typical Reasons For Non Stationarity

- **Obvious dependency on time**
  - Seasonality, growth
  - Can be ignored at small time scale (minute)

- **Non Stability: Explosion**
  - Queue with utilization factor >1

- **Non Stability: Freezing Simulation**
  - System becomes slower with time (aging)
    - Typically because there are rare events of large impact (« Kings »)
    - The longer the simulation, the larger the largest king
  - Ex: time between regeneration points has infinite mean
  - We’ll come back to this in the chapter « Importance of the View Point »
2 Simulation Techniques

- Discrete event simulations
- Recurrences
  - Stochastic recurrences
Discrete event simulation
Uses an Event Scheduler

Example:
- Information system modelled as a single server queue
- Three event classes
  - arrival
  - service
  - Departure
- One event scheduler (global system clock)
Scheduler: Timeline

State of scheduler Event being executed

<table>
<thead>
<tr>
<th>Step 1</th>
<th>Step 2</th>
<th>Step 3</th>
<th>Step 4</th>
<th>Step 5</th>
<th>Step 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrival t=0</td>
<td>service t=0</td>
<td>arrival t=t_1</td>
<td>departure t=t_2</td>
<td>service t=t_2</td>
<td>departure t=t_3</td>
</tr>
<tr>
<td>arrival t=t_1</td>
<td>departure t=t_2</td>
<td>arrival t=t_4</td>
<td>arrival t=t_4</td>
<td>arrival t=t_4</td>
<td>arrival t=t_4</td>
</tr>
</tbody>
</table>

queue length

initialization

<table>
<thead>
<tr>
<th>t</th>
<th>0</th>
<th>t_1</th>
<th>t_2</th>
<th>t_3</th>
<th>t_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue length</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Statistical Counters

- Assume we want to output: mean queue length and mean response time. How do we do this?

**Statistics Counters:**
- `queueLengthCtr` is $\int_0^t q(s) ds$ where $q(s)$ is the value of `buffer.length` at time $s$ and $t$ is the current time. At the end of the simulation, the mean queue length is $\frac{queueLengthCtr}{T}$ where $T$ is the simulation finish time.

- The counter `responseTimeCtr` holds $\sum_{m=1}^{m} R_m$ where $R_m$ is the response time for the $m$th request and $n$ is the value of `nbRequests` at the current time. At the end of the simulation, the mean response time is $\frac{responseTimeCtr}{N}$ where $N$ is the value of `nbRequests`.

- Note the difference between
  - Event based statistic
  - Time based statistic
A Classical Organization of Simulation Code

- Events contain specific code
- A main loop advances the state of the scheduler
- Example: in the code of a *departure* event

**Departure:** *Update Event Based Counters.* Let $c$ be the request at the head of buffer. Increment $\text{responseTimeCtr}$ by $d - a$, where $d$ is this event’s $\text{firingTime}$ and $a$ is the arrival time of the request $c$. Increment $\text{nbRequests}$ by 1.

**Execute Event’s Actions.** Remove the request $c$ from buffer and delete it.

**Schedule Follow-Up Events.** If buffer is not empty after the removal, create a new event of class *Service*, with $\text{firingTime}$ equal to this event’s $\text{firingTime}$, and insert it into $\text{eventScheduler}$. 
Main program

- **Bootstrapping.** Create a new event of class `Arrival` with `firingTime` equal to 0 and insert it into `eventScheduler`.

- **Execute Events.** While the simulation stopping condition is not fulfilled, do the following.
  
  - **Increment Time Based Counters.** Let `e` be the first event in `eventScheduler`. Increment `queueLengthCtr` by \( q(t_{\text{new}} - t_{\text{old}}) \) where \( q = \text{buffer.length}, t_{\text{new}} = e.firingTime \) and \( t_{\text{old}} = \text{currentTime} \).

  - Execute `e`.

  - Set `currentTime` to `e.firingTime`

  - Delete `e`

- **Termination.** Compute the final statistics:
  
  \[
  \text{meanQueueLength} = \frac{\text{queueLengthCtr}}{\text{currentTime}}
  \]

  \[
  \text{meanResponseTime} = \frac{\text{responseTimeCtr}}{\text{nbRequests}}
  \]
**Stochastic Recurrence**

- An alternative to discrete event simulation
  - faster but requires more work on the model
  - not always applicable
- Defined by iteration:

\[
\begin{align*}
X_0 &= x_0 \\
X_{n+1} &= f(X_n, Z_n)
\end{align*}
\]

where \(X_n\) is the state of the system at the \(n\)th transition (For any realization, \(X_n\) is in some possibly complicated state space \(\mathcal{X}\)), \(x_0\) is a fixed, given state in \(\mathcal{X}\), \(Z_n\) is some stochastic process that can be simulated (for example a sequence of iid random variables, or a Markov chain), and \(f\) is a deterministic mapping.

The simulated time \(T_n\) at which the \(n\)th transition occurs is assumed to be included in the state variable \(X_n\).
Example: random waypoint mobility model

**Example 6.5: Random Waypoint.**

The *random waypoint* is a model for a mobile point, and can be used to simulate the mobility pattern in Example 6.1. It is defined as follows. The state variable is $X_n = (M_n, T_n)$ where $M_n$ is the position of the mobile at the $n$th transition (the $n$th “waypoint”) and $T_n$ is the time at which this destination is reached. The point $M_n$ is chosen at random, uniformly in a given convex area $\mathcal{A}$. The speed at which the mobile travels to the next waypoint is also chosen at random uniformly in $[v_{\text{min}}, v_{\text{max}}]$.

The random waypoint model can be cast as a stochastic recurrence by letting $Z_n = (M_{n+1}, V_{n+1})$, where $M_{n+1}, V_{n+1}$ are independent i.i.d. sequences, such that $M_{n+1}$ is uniformly distributed in $\mathcal{A}$ and $V_{n+1}$ in $[v_{\text{min}}, v_{\text{max}}]$. We have then the stochastic recurrence

$$X_{n+1} := (M_{n+1}, T_{n+1}) = (M_{n+1}, T_n + \frac{||M_{n+1} - M_n||}{V_n})$$
Course of one user

Waypoints of one user

(a) 1 mobile
Course of all users

Waypoints of all users

(b) 10 mobiles
Queuing System implemented as Stochastic Recurrence

Let $X_n$ represent the state of the simulator just after an arrival or a departure, as follows:

$$X_n = (t_n, b_n, q_n, a_n, d_n)$$

with $t_n =$ the simulated time at which this transition occurs, $b_n =$buffer.length, $q_n =$queueLengthCtr (both just after the transition), $a_n =$ the time interval from this transition to the next arrival and $d_n =$ the time interval from this transition to the next departure.

Let $Z_n$ be a couple of two random numbers, drawn independently of anything else, with distribution uniform in $(0, 1)$.

The recurrence is defined by $f((t, b, q, a, d), (z_1, z_2)) = (t', b', q', a', d')$ with
if $a < d$ // this transition is an arrival

$$
\Delta = a \\
 t' = t + a \\
 b' = b + 1 \\
 q' = q + b\Delta \\
 a' = F^{-1}(z_1)
$$

if $b == 0$ then $d' = G^{-1}(z_2)$ else $d' = d - \Delta$

else // this transition is a departure

$$
\Delta = d \\
 t' = t + d \\
 b' = b - 1 \\
 q' = q + b\Delta \\
 a' = a - \Delta
$$

if $b' > 0$ then $d' = G^{-1}(z_1)$ else $d' = \infty$
3 Accuracy of Simulation Output

- A stochastic simulation produces a random output, we need confidence intervals
- Method of choice: independent replications
- Remove transients
  - For non terminating simulations
- Be careful to have truly random seeds
  - Ex: use computer time as seed
Results of 30 Independent Replications
Do They Look Normal?
Bootstrap Replicates
Confidence Intervals

Confidence Intervals for Mean and Median

Number of customers

Mean, normal approx
Mean, bootstrap
Median
5 Random Number Generator

- A stochastic simulation does not use truly random numbers but pseudo-random numbers
  - Produces a random number \( \sim U(0,1) \)
  - Example (obsolete but commonly used, e.g., in ns2:)

---

**Example 6.8: Linear Congruence.** A widespread generator (for example the default in ns2) has \( a = 16^{1607} \) and \( m = 2^{31} - 1 \). The sequence is \( x_n = \frac{sa^n \mod m}{m} \)

where \( s \) is the seed. \( m \) is a prime number, and the smallest exponent \( h \) such that \( a^h = 1 \mod m \) is \( m - 1 \). It follows that for any value of the seed \( s \), the period of \( x_n \) is exactly \( m - 1 \). Figure 6.5 shows that the sequence \( x_n \) indeed looks random.

- Output appears to be random (see next slides)
The Linear Congruential Generator of ns2

Uniform QQ Plot

(a)

autocorrelation

uniform qq plot
Lag Diagram, 1000 points
Period of RNG

- RNG is in fact periodic
  - Period should be much larger than maximum number of uses

The period of a random number generator should be much smaller than the number of times it is called in a simulation. The generator in Example 3.8 on page 72 has a period of ca. $2 \times 10^9$, which may be too small for very large simulations. There are other generators with much longer periods, for example the “Mersenne Twister” [Matsumoto-98] with a period of $2^{19937} - 1$. They use other chaotic sequences and combinations of them.
Two Parallel Streams with too simple a RNG

(a)
Impact of RNG

(a) Linear Congruence with $a = 16'807$ and (b) L’Ecuyer’s generator\cite{Lecuyer2001} $m = 2^{31} - 1$
Take Home Message

- Be careful to have a RNG that has a period orders of magnitude larger than what you will ever use in the simulation
- Serialize the use of the RNG rather than parallel streams
6 Sampling From A Distribution

- Pb is:
  - Given a distribution $F()$, and a RNG, produce some sample $X$ that is drawn from this distribution
- A common task in simulation
- Matlab does it for us most of the time, but not always
- Two generic methods
  - CDF inversion
  - Rejection sampling
CDF Inversion

- Applies to real or integer valued RV

**Theorem 6.6.1.** Let $F$ be the CDF of a random variable $X$ with values in $\mathbb{R}$. Define the pseudo-inverse, $F^{-1}$ of $F$ by

$$F^{-1}(p) = \sup \{ x : F(x) \leq p \}$$

Let $U$ be a sample of a random variable with uniform distribution on $(0, 1)$; $F^{-1}(U)$ is a sample of $X$. 
Example 7.10: Exponential Random Variable. The CDF of the exponential distribution with parameter $\lambda$ is $F(x) = 1 - e^{-\lambda x}$. The pseudo-inverse is obtained by solving the equation

$$1 - e^{-\lambda x} = p$$

where $x$ is the unknown. The solution is $x = -\frac{\ln(1-p)}{\lambda}$. Thus a sample $X$ of the exponential distribution is obtained by letting $X = -\frac{\ln(1-U)}{\lambda}$, or, since $U$ and $1-U$ have the same distribution:

$$X = -\frac{\ln(U)}{\lambda}$$

(7.9)

where $U$ is the output of the random number generator.
Example: integer valued RV

Thus, an integer valued random variable $N$ can be sampled by: $N = \text{the index } n \text{ such that}$

![Diagram showing the pseudo-inverse of the CDF for an integer-valued random variable.](image)

Figure 3.8: Pseudo-Inverse of cdf $F()$ of an integer-valued random variable $F(n - 1) \leq U < F(n)$, where $U$ is the output of the random generator.
Example 7.11: Geometric Random Variable. Here $X$ takes integer values $0, 1, 2, \ldots$. The geometric distribution with parameter $\theta$ satisfies $\mathbb{P}(X = k) = \theta(1 - \theta)^k$, thus for $n \in \mathbb{N}$:

$$F(n) = \sum_{k=0}^{n} \theta(1 - \theta)^k = 1 - (1 - \theta)^{n+1}$$

by application of Eq.(7.10):

$$F^{-1}(p) = n \Leftrightarrow n \leq \frac{\ln(1 - p)}{\ln(1 - \theta)} < n + 1$$

hence

$$F^{-1}(p) = \left\lfloor \frac{\ln(1 - p)}{\ln(1 - \theta)} \right\rfloor$$

and, since $U$ and $1 - U$ have the same distribution, a sample $X$ of the geometric distribution is

$$X = \left\lfloor \frac{\ln(U)}{\ln(1 - \theta)} \right\rfloor$$  \hspace{1cm} (7.11)
Rejection Sampling

- Applies more generally, also to joint n-dimensional distributions
- Example 1: conditional distribution on this area
- Step 1:
  - Can you sample a point uniformly in the bounding rectangle?
- Step 2:
  - How can you go from there to a uniform sample inside the non convex area?

---

10 The coordinates are independent and uniform: generate two independent samples $U, V \sim \text{Unif}(0, 1)$; the sample is $((1-U)x_{\min} + UX_{\max}, (1-V)y_{\min} + VY_{\max})$. 
Rejection Sampling for Conditional Distribution

This is the main idea

THEOREM 3.6.2 (Rejection Sampling for a Conditional Distribution). Let $X$ be a random variable in some space $S$ such that the distribution of $X$ is the conditional distribution of $\tilde{X}$ given that $\tilde{Y} \in A$, where $(\tilde{X}, \tilde{Y})$ is a random variable in $S \times S'$ and $A$ is a measurable subset of $S$. A sample of $X$ is obtained by the following algorithm:

\[
do
\begin{align*}
&\text{draw a sample of } (\tilde{X}, \tilde{Y}) \\
&\text{until } \tilde{Y} \in A \\
&\text{return}(\tilde{X})
\end{align*}
\]

The expected number of iterations of the algorithm is $\frac{1}{P(\tilde{Y} \in A)}$.

How can you apply this to the example in the previous slide?
Rejection Sampling for General Distributions

THEOREM 3.6.3 (Rejection Sampling for Distribution with Density). Consider two random variables $X, Y$ with values in the same space, that both have densities. Assume that:

- we know a method to draw a sample of $X$
- the density of $Y$ is known up to a normalization constant $K$: $f_Y(y) = K f_Y^n(y)$, where $f_Y^n$ is a known function
- there exist some $c > 0$ such that

$$\frac{f_Y^n(x)}{f_X(x)} \leq c$$

A sample of $Y$ is obtained by the following algorithm:

```
    do
        draw independent samples of $X$ and $U$, where $U \sim \text{Unif}(0, c)$
        until $U \leq \frac{f_Y^n(X)}{f_X(X)}$
    return $(X)$
```

The expected number of iterations of the algorithm is $\frac{c}{K}$. 

Example 3.14: Arbitrary Distribution with Density. Assume that: \( Y \) takes values in the bounded interval \([a, b]\), has a density \( f_Y = K f^n_Y(y) \) that can easily be computed but for the multiplicative constant \( K \), and that we know an upper bound \( M \) on \( f^n_Y \). We take \( X \) uniformly distributed over \([a, b]\) and obtain the sampling method:

\[
\begin{align*}
do & \\
& \text{draw } X \sim \text{Unif}(a, b) \text{ and } U \sim \text{Unif}(0, M) \\
& \text{until } U \leq f^n_Y(X) \\
& \text{return}(X)
\end{align*}
\]

Note that we do not need to know the multiplicative constant \( K \). For example, consider the distribution with density

\[
f_Y(y) = K \frac{\sin^2(y)}{y^2} 1_{\{ -a \leq y \leq a \}} \quad (3.13)
\]

\( K \) is hard to compute, but a bound \( M \) on \( f^n_Y \) is easy to find \((M = 1)\).
A Sample from a Weird Distribution

Figure 3.8: (a) Empirical histogram (bin size = 10) of 2000 samples of the distribution with density $f_X(x)$ proportional to $\frac{\sin^2(x)}{x^2}1\{-a \leq y \leq a\}$ with $a = 10$. (b) 2000 independent samples of the distribution on the rectangle with density $f_{X_1, X_2}(x_1, x_2)$ proportional to $|x_1 - x_2|$. 
Example 6.14: A Stochastic Geometry Example. We want to sample the random vector \((X_1, X_2)\) that takes values in the rectangle \([0, 1] \times [0, 1]\) and whose distribution has a density proportional to \(|X_1 - X_2|\). We take \(f_X = \text{the uniform density over } [0, 1] \times [0, 1]\) and \(f_Y^n(x_1, x_2) = |x_1 - x_2|\). An upper bound on the ratio \(\frac{f_Y^n(x_1, x_2)}{f_X(x_1, x_2)}\) is 1. The sampling algorithm is thus:

```
do
    draw \(X_1, X_2\) and \(U \sim \text{Unif}(0, 1)\)
until \(U \leq |X_1 - X_2|\)
return \((X_1, X_2)\)
```

Figure 6.10 shows an example. Note that there is no need to know the normalizing constant to apply the sampling algorithm.
Another Sample from a Weird Distribution

Figure 3.8: (a) Empirical histogram (bin size = 10) of 2000 samples of the distribution with density $f_X(x)$ proportional to $\frac{\sin^2(x)}{x^2} \mathbf{1}_{-a \leq y \leq a}$ with $a = 10$. (b) 2000 independent samples of the distribution on the rectangle with density $f_{X_1, X_2}(x_1, x_2)$ proportional to $|x_1 - x_2|$. 
6.3 Ad-Hoc Methods

- Optimized methods exist for some common distributions
  - Optimization = reduce computing time
- If implemented in your tool, use them!
- Example: simulating a normal distribution
  - Inversion method is not simple (no closed form for $F^{-1}$)
  - Rejection method is possible
  - But a more efficient method exists, for drawing jointly 2 independent normal RV
- There are also ad-hoc methods for n-dimensional normal distributions (gaussian vectors)
Figure 3.11: 1000 independent samples of the normal vector $X_1, X_2$ with 0 mean and covariance $\Omega_{1,1} = \sigma_1^2 = 5$, $\Omega_{1,2} = \Omega_{2,1} = 5$, $\Omega_{2,2} = \sigma_2^2 = 10$. It is obtained by the transformation $X = LY$ with $Y \sim N_{0,1}$ and $L = \sqrt{5}(1, 0; 1/2, \sqrt{3}/2)$. 
4 Monte Carlo Simulation

- A simple method to compute integrals of all kinds
- Idea: interpret the integral as $\beta = \mathbb{E}(\varphi(\bar{X}))$.
- Assume you can simulate as many independent samples of $X$ as you want

Monte-Carlo simulation consists in generating $R$ iid replicates $\bar{X}^r$, $r = 1, \ldots, R$. The Monte-Carlo estimate of $\beta$ is

$$\hat{\beta} = \frac{1}{R} \sum_{r=1}^{R} \varphi(\bar{X}^r)$$

A confidence interval for $\beta$ can then be computed using the methods in Chapter 2 for a confidence interval for the mean. By adjusting $R$, the number of replications, we can control the accuracy of the method, i.e. the width of the confidence interval.
Example 3.7: \textit{p-value of a test.} Let $X_1, \ldots, X_n$ be a sequence of iid random variables that take values in the discrete set \{1, 2, \ldots, I\}. Let $q_i = \mathbb{P}(X_k = i)$. Let $N_i = \sum_{k=1}^{n} 1_{\{X_k=i\}}$ (number of observation that are equal to $i$). Assume we want to compute

$$p = \mathbb{P}\left( \sum_{i=1}^{k} N_i \ln \frac{N_i}{nq_i} > a \right) \quad (3.3)$$

where $a > 0$ is given. This computation arises in the theory of goodness of fit tests, when we want to test whether $X_i$ does indeed come from the model defined above. For large values of the sample size $n$ we can approximate $\beta$ by a $\chi^2$ distribution (see Section 7.5), but for small values there is no analytic result.
We use Monte-Carlo simulation to compute $p$. We generate $R$ iid replicates $X_1^r, \ldots, X_n^r$ of the sequence ($r = 1, \ldots, R$). This can be done by using the inversion method described in this chapter. For each replicate $r$, let

$$N_i^r = \sum_{k=1}^{n} 1\{X_k^r = i\}$$  \hspace{1cm} (3.4)

The Monte Carlo estimate of $p$ is

$$\hat{p} = \frac{1}{R} \sum_{r=1}^{R} 1\{\sum_{i=1}^{k} N_i \ln \frac{N_i}{nq_i} > a\}$$  \hspace{1cm} (3.5)
We compute a confidence interval by using a normal approximation, as explained in Example 2.14 on page 39. The sample variance is estimated by

$$\hat{\sigma} = \sqrt{\frac{\hat{p}(1 - \hat{p})}{R}} \quad (3.6)$$

and a confidence interval at level 0.95 is $\hat{p} \pm 1.96\hat{\sigma}$. Assume we want a relative accuracy at least equal to some fixed value $\epsilon$ (for example $\epsilon = 0.05$). This is achieved if

$$\frac{1.96\hat{\sigma}}{\hat{p}} \leq \epsilon \quad (3.7)$$

which is equivalent to

$$R \geq \frac{3.92}{\epsilon^2} \left( \frac{1}{\hat{p}} - 1 \right) \quad (3.8)$$
ulation when this happens. Table 3.1 shows some results; we see that \( p \) is equal to
0.19 with an accuracy of 5%; the number of Monte Carlo replicates is proportional to
the relative accuracy to the power \(-2\).

<table>
<thead>
<tr>
<th>( R )</th>
<th>( \hat{p} )</th>
<th>margin</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>0.2667</td>
<td>0.1582</td>
</tr>
<tr>
<td>60</td>
<td>0.2500</td>
<td>0.1096</td>
</tr>
<tr>
<td>120</td>
<td>0.2333</td>
<td>0.0757</td>
</tr>
<tr>
<td>240</td>
<td>0.1917</td>
<td>0.0498</td>
</tr>
<tr>
<td>480</td>
<td>0.1979</td>
<td>0.0356</td>
</tr>
<tr>
<td>960</td>
<td>0.2010</td>
<td>0.0254</td>
</tr>
<tr>
<td>1920</td>
<td>0.1865</td>
<td>0.0174</td>
</tr>
<tr>
<td>3840</td>
<td>0.1893</td>
<td>0.0124</td>
</tr>
<tr>
<td>7680</td>
<td>0.1931</td>
<td>0.0088</td>
</tr>
</tbody>
</table>

Table 3.1: Computation of \( p \) in Example 3.7 on page 72 by Monte Carlo simulation. The parameters of the
model are \( I = 4, q_1 = 9/16, q_2 = q_3 = 3/16, q_4 = 1/16, n = 100 \) and \( \alpha = 2.4 \). The table shows the estimate \( \hat{p} \) of \( p \) with its 95% confidence margin versus the number of Monte-Carlo replicates \( R \). With 7680 replicates
the relative accuracy (margin/\( \hat{p} \)) is below 5%. 
Take Home Message

- Most hard problems relative to computing a probability or an integral can be solved with Monte Carlo
  - Braindumb but why not
  - Run time may be large -> importance sampling techniques (see later in this lecture)
Conclusion

Simulating well requires knowing the concepts of
- Transience
- Confidence intervals
- Sampling methods