Joseph High
9E1FDC

Problem A:

(a.) Use the Kantorovich inequality to derive an upper bound to the magnitude of the bias in the regenerative estimate $\hat{\ell}$ in terms of m and M.

Sol'n: From the definition of the bias of $\hat{\ell}$,

$$\text{Bias}(\hat{\ell}) = E[\hat{\ell}] - \ell$$

and $\mathbb{E}[\hat{\ell}] = E\left[\frac{\hat{R}}{\hat{\tau}}\right] - \frac{E[R]}{E[\tau]}$

Now, since τ_i and R_i are independent for all i then

$$E[\hat{\ell}] = \mathbb{E}\left[\frac{\hat{R}}{\hat{\tau}}\right] = \mathbb{E}\left[\frac{\frac{1}{N}\sum_{i=1}^{N} R_i}{\frac{1}{K}\sum_{i=1}^{K} \tau_i}\right] =$$

$$= \mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N} R_i\right] \mathbb{E}\left[\frac{1}{\frac{1}{K}\sum_{i=1}^{K} \tau_i}\right]$$ (since τ_i and R_i independently sampled)

($K \neq N$ (in general) since distinct sample paths)

$$= E[R]\ \mathbb{E}\left[\frac{1}{\hat{\tau}}\right]$$

Then,

$$|\text{Bias}(\hat{\ell})| = |E[\hat{\ell}] - \ell| = \left| E[R]\, E\left[\frac{1}{\hat{\tau}}\right] - \frac{E[R]}{E[\tau]} \right|$$

$$\leq |E[R]| \left| E\left[\frac{1}{\hat{\tau}}\right] - \frac{1}{E[\tau]} \right| \leq \ldots \longrightarrow$$

continued..

$$\leq \left|\frac{E[R]}{E[\tau]}\right| \left| E[\tau] E\left[\frac{1}{\hat{\tau}}\right] - 1 \right|$$

$$= |\ell| \left| E[\hat{\tau}] E\left[\frac{1}{\hat{\tau}}\right] - 1 \right| \leq$$

$$\leq |\ell| \left| \frac{(m+M)^2}{4mM} - 1 \right| = |\ell| \left| \frac{m^2 + 2mM + M^2 - 4mM}{4mM} \right|$$

$$= |\ell| \left| \frac{(m-M)^2}{4mM} \right|$$

$$\Rightarrow |\mathrm{Bias}(\hat{\ell})| \leq |\ell| \left| \frac{(m-M)^2}{4mM} \right|$$

since $m, M > 0$ $\qquad = |\ell| \frac{(m-M)^2}{4mM}$ //

(b) <u>Numerical Values of the upperbound:</u>

<u>For $M/m = 1.1$</u> : $M = (1.1)m$

$$\Rightarrow |\mathrm{Bias}(\hat{\ell})| \leq |\ell| \frac{(m - 1.1m)^2}{4m(1.1m)}$$

$$\Rightarrow \left| \frac{\mathrm{Bias}(\hat{\ell})}{\ell} \right| \leq \frac{0.01m^2}{4.4m^2} \approx 0.00227$$

$\Longrightarrow$

For $M/m = 1.5$: $\left|\frac{\text{Bias}(\hat{\ell})}{\ell}\right| \le \frac{(m - 1.5m)^2}{4m(1.5m)} =$

$$= \frac{\frac{1}{4}m^2}{6m^2} = 0.04167$$

For $M/m = 2.0$: $\left|\frac{\text{Bias}(\hat{\ell})}{\ell}\right| \le \frac{(m - 2m)^2}{4m\,2m} = \frac{m^2}{8m^2}$

$$= \frac{1}{8}$$

$$= 0.125$$

Thus, for

	Bound on $\left\lvert\frac{\text{Bias}(\hat{\ell})}{\ell}\right\rvert$
$M/m = 1.1$	0.00227
$= 1.5$	0.04167
$= 2.0$	0.125

Potential practical application(s):

In the service/queuing problem in Ch. 1 (and represented in problem 4.8) this might be a practical approach to estimating ℓ.
For instance, if one can find bounds on the interarrival times, T_i, then one can determine the bound on $\left|\frac{\text{Bias}(\hat{\ell})}{\ell}\right|$ and subsequently determine how far the mean of the estimate $\hat{\ell}$ is from the true value ℓ.

Joseph High
9E1FDC

4.8 part a

Code:

```
%Exercise 4.8
%Part a

M=10000;
K=100;
N=30;
T=(M-K)/N;
mu=2;
lambda=1;
rho=lambda/mu;
n1=0;
nn1=n1;
ev_list=inf*ones(2,2);
t=0;
tot=0;
tt=0;
ev_list(1,:) = [-log(rand)/lambda, 1];
N_ev=1;
Tot = zeros(1, N);
while t < M
t = ev_list(1,1);
tt=[tt,t];
ev_type = ev_list(1,2);
switch ev_type
case 1
    interarrival
    case 2
        service
end
N_ev = N_ev - 1;
ev_list(1,:) = [inf,inf];
ev_list = sortrows(ev_list, 1);
nn1 = [nn1, n1];
tot = tot + nn1(end-1)*(tt(end) - tt(end-1));
end
batches = zeros(1,N);
D = zeros(1,N);
for i = 1:N
    batches(i)=mean(nn1(101 + (i-1)*T:100 + T*i));
end
batchmean = mean(batches);
re = std(batches)/hope/sqrt(N)
res= tot/t;
fprintf('batches %g ; 0.95 CI (%g, %g) /n', batchmean, batchmean*(1-1.96*re),
batchmean*(1+1.96*re))
```

Joseph High
9E1FDC

Definitions:

```
%interarrival.m
N_ev = N_ev + 1;
ev_list(N_ev, :) = [t - log(rand)/lambda, 1];
if n1 == 0
    N_ev = N_ev + 1;
    ev_list(N_ev, :) = [t - log(rand)/mu, 2];
end
n1 = n1+1;


% service.m
n1=n1-1;
if n1 ~= 0
N_ev = N_ev + 1;
ev_list(N_ev,:)=[t - log(rand)/mu, 2] ;
end
```

Output:

```
>> hw8prob8crap

re =

    0.0362

batches 1.01037 ; 0.95 CI (0.93872, 1.08202) /n>>
```

4.8 part b

Code

```
%Part B
M=10000;
K=100;
N=30;
T=(M-K)/N;
mu=2;
lambda=1;
rho=lambda/mu;
n1=0;
nn1=n1;
ev_list=inf*ones(2,2);
t=0;
tot=0;
tt=0;
ev_list(1,:) = [-log(rand)/lambda, 1];
N_ev=1;
R = zeros(1,N);
tau = zeros(1,N);
Rsum = 0;
regcount = 0;
```

```
lastregtime = 1;
for i = 1:numel(nn1)
while t < M
t = ev_list(1,1);
tt=[tt,t];
ev_type = ev_list(1,2);
switch ev_type
case 1
    interarrival
    case 2
        service
end
Rsum = Rsum + nn1(i);
N_ev = N_ev - 1;
ev_list(1,:) = [inf,inf];
ev_list = sortrows(ev_list, 1);
nn1 = [nn1, n1];
end
if nn1(i) == 0
    regcount = regcount + 1;
    R(regcount) = Rsum;
    tau(regcount) = i - lastregtime;
    Rsum = 0;
    lastregtime = i;
end
end
regmean = mean(R)/mean(tau)
Covariance = cov(R, tau);
S = sqrt(Covariance(1,1)-2*regmean*Covariance(1,2) + regmean^2*Covariance(2,2))
RelError = S/mean(tau)/sqrt(M)
fprintf('regmean %g; 0.95 CI (%g, %g) \n', regmean, regmean*(1-1.96*RelError),
regmean*(1+1.96*RelError))
```

Output:

```
>> hw8prob8b

RelError =

    0.0589

regmean 1.06699; 0.95 CI (0.943813, 1.19016)
```

Part C

Relative Width = 1.96 x 2 x Relative Error

For part a and part b, the relative width around the confidence intervals are 0.1176 for part a and 0.2309 for part b, this is for simulation time of M = 10,000. If we increase M to 70,000 we see that the relative error shrinks down to approximately 0.0145 for part a and if we increase to 170,000 for part b, we get an approximately relative error of 0.0145 as well. This gives a relative width of approximately 0.05 for each.

Joseph High
9E1FDC

The output for each is below. Note: the code is not attached for this because it is exactly the same as the above just with M = 70,000 for part a and M = 170,000 for part b. (Note, I know the solution manual says 60,000 for both, but that did not work for my code).

Output for part a:
re =

0.0145

batches 0.992564 ; 0.95 CI (0.964332, 1.0208) /n>> hw8prob8b

Output for part b:
RelError =

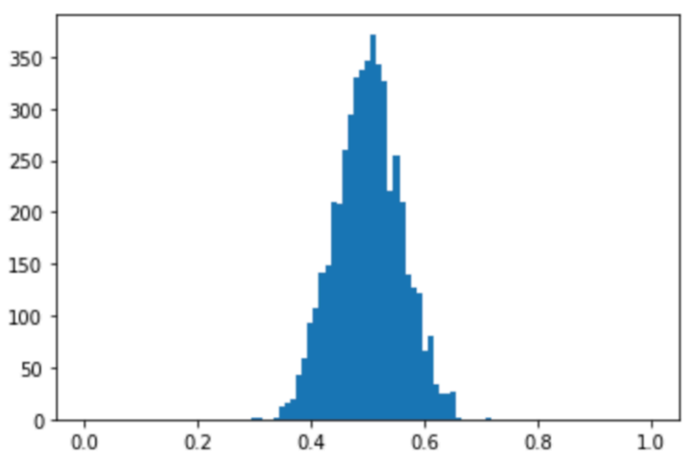
0.0145

regmean 0.989388; 0.95 CI (0.961363, 1.01741)

Exercise B

a
```
np.random.seed(123)
sample.size = 5000
numpoints = 25
d = np.random.uniform(0,1,(numpoints, sample.size))
dbar = np.sort(np.mean(d,0))
ugen = np.random.uniform(0,1,sample.size)
unorm = sample.size*ugen
unorm = unorm.astype(int)
dbarbar = np.sort(dbar[unorm])
plt.hist(dbarbar, bins = np.linspace(0,1,100))
plt.show()
```

Output:



Joseph High
9E1FDC

b

```
updatedbar = np.mean(dbar)
L = int(0.025*sample.size) – 1
U = int(0.975*sample.size) – 1
Lb = dbar[L] – updatedbar
Ub = dbar[U] – updatedbar
print("CI "str(Lb)" "+str(Ub)+')

Output:
(-0.137899983311, 0.140012210078)
```