Solutions to Problem Sheet 4

1. Use Monte Carlo simulation to estimate the value of the following integral, giving a confidence interval for your estimate.

$$I = \int_{3/2}^{5/2} x^3 + 1 \, dx$$

Solution 1 *First let's create a function for estimating proportions:*

```
>>> def estimate_proportion(integrand, xlower, xupper, ymax, N):
...     n = 0
...     for trial in range(N):
...         x = ((xupper - xlower) * random.random()) + xlower
...         y = random.random() * ymax
...         if y < integrand(x):
...             n += 1
...     return n / N
```

Now a function for calculating confidence intervals

```
>>> def confidence_interval(p, N, area):
...     lower = p - 1.96 * ((p * (1 - p) / N) ** 0.5)
...     upper = p + 1.96 * ((p * (1 - p) / N) ** 0.5)
...     return (area * lower, area * upper)
```

Now define the integrand:

```
>>> def integrand_1(x):
...     return (x ** 3) + 1
```

This is an increasing function, so the maximum value that the integrand can take is when $x = 5/2$. So let's sample in the rectangle with height from 0 to this maximum, and width from $3/2$ to $5/2$.

Solution 1 (continuing from p. 1) *Then:*

```
>>> random.seed(0)
>>> N = 1000
>>> xlower, xupper, ymax = 3/2, 5/2, integrand_1(5/2)
>>> area = (xupper - xlower) * ymax
>>> p = estimate_proportion(integrand_1, xlower, xupper, ymax, N)
>>> I = p * area
>>> I
9.559375
>>> confidence_interval(p, N, area)
(9.049990046124861, 10.068759953875137)
```

So we can be 95% confident that $9.05 < I < 10.07$.

2. Use Monte Carlo simulation to estimate the value of the following integral, giving a confidence interval for your estimate.

$$I = \int_{-1}^{1} \sqrt{1 - x^2}\, dx$$

Solution 2 *Define the integrand:*

```
>>> def integrand_2(x):
...     return (1 - (x ** 2)) ** 0.5
```

This is the upper half circle radius 1, so the maximum value of the integrand is 1. Using the same functions as in the previous question, sample points in the rectangle height [0, 1], and width [-1, 1]:

```
>>> random.seed(0)
>>> N = 1000
>>> xlower, xupper, ymax = -1, 1, 1
>>> area = (xupper - xlower) * ymax
>>> p = estimate_proportion(integrand_2, xlower, xupper, ymax, N)
>>> I = p * area
>>> I
1.542
>>> confidence_interval(p, N, area)
(1.4899128017877714, 1.5940871982122287)
```

So we can be 95% confident that $1.49 < I < 1.59$.

3. Use Monte Carlo simulation to find the probability p, that when four dice are thrown simultaneously, they can be split into two pairs with equal sums. For example, the roll $(3, 1, 3, 5)$ can be split into two pairs with equal sums: $3 + 3 = 1 + 5$, however $(2, 2, 2, 4)$ cannot. Give a confidence interval for your estimate.

Solution 3 *Two functions will be useful, first, a function to roll a die:*

```
>>> def roll():
...     return random.randint(1, 6)
```

then a function to check if four random numbers can be split into two pairs with equal sums:

```
>>> def check(a, b, c, d):
...     comb1 = (a + b) == (c + d)
...     comb2 = (a + c) == (b + d)
...     comb3 = (a + d) == (b + c)
...     if comb1 or comb2 or comb3:
...         return True
```

Now run the simulation with $N = 1000$ trials:

```
>>> random.seed()
>>> N = 1000
>>> n = 0
>>> for trial in range(N):
...     a, b, c, d = roll(), roll(), roll(), roll()
...     if check(a, b, c, d):
...         n += 1
>>> p = n / N
>>> p
0.286
```

and finally the confidence interval:

```
>>> lower = p - 1.96 * ((p * (1 - p) / N) ** 0.5)
>>> upper = p + 1.96 * ((p * (1 - p) / N) ** 0.5)
>>> (lower, upper)
(0.2579916068579434, 0.31400839314205653)
```

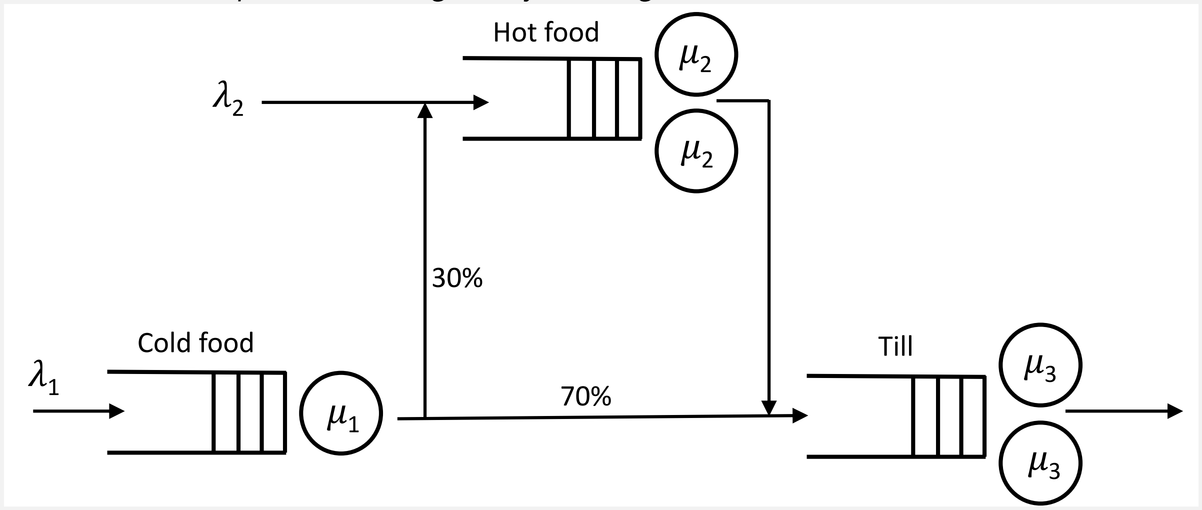
So we can be 95% confident that $0.258 < p < 0.314$.

4. The system to order food at a cafe in the centre of Cardiff behaves as a network of queues. There are three counters: the cold food counter, the hot food counter, and the till where customers pay for their food. If customers want cold food only, they arrive and join the cold food till; if they want hot food only, they arrive and join the hot food till; if they want both cold and hot food they must first queue at the cold food counter, and then at the hot food counter. After picking up their food, they must then queue at the till to pay.

   - Customers arrive at the cold food counter at a rate of 18 per hour,
   - Customers arrive at the hot food counter at a rate of 12 per hour,
   - 30% of customers who queue at the cold food counter also want hot food,
   - On average it takes 1 minute to be served cold food, 2 and a half minutes to be served hot food, and 2 minutes to pay,
   - There is 1 server at the cold food counter, 2 servers at the hot food counter, and 2 servers at the till,
   - The cafe is open for 3 hours during the lunch hour.

   The cafe would like to know how many customers they expect to be served each lunchtime on average. Perform a discrete-event simulation in Ciw, and show the conceptual model.

Solution 4 *The conceptual model is given by the diagram below:*



We will assume arrivals are Markovian and service times are Exponentially distributed. Choosing our time units as minutes, we have $\lambda_1 = {}^{18}\!/_{60} = 0.3$, $\lambda_2 = {}^{12}\!/_{60} = 0.2$, $\mu_1 = {}^{1}\!/_{1} = 1$, $\mu_2 = {}^{1}\!/_{2.5} = 0.4$, *and* $\mu_3 = {}^{1}\!/_{2} = 0.5$.

Solution 4 (continuing from p. 4) *Therefore the Ciw network would be:*

```
>>> import ciw
>>> N = ciw.create_network(
...     arrival_distributions=[ciw.dists.Exponential(rate=0.3),
...                            ciw.dists.Exponential(rate=0.2),
...                            None],
...     service_distributions=[ciw.dists.Exponential(rate=1.0),
...                            ciw.dists.Exponential(rate=0.4),
...                            ciw.dists.Exponential(rate=0.5)],
...     routing=[[0.0, 0.3, 0.7],
...              [0.0, 0.0, 1.0],
...              [0.0, 0.0, 0.0]],
...     number_of_servers=[1, 2, 2]
... )
```

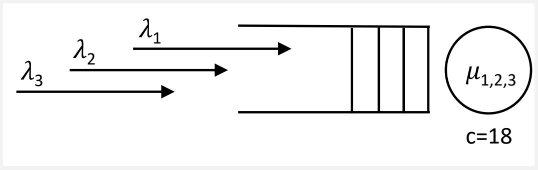
This is a terminal problem, as the system begins from empty each lunchtime. Therefore we should run trials. Choosing say 10 trials:

```
>>> completed_custs = []
>>> for trial in range(10):
...     ciw.seed(trial)
...     Q = ciw.Simulation(N)
...     Q.simulate_until_max_time(180)
...     recs = Q.get_all_records()
...     num_completed = len([r for r in recs if r.node==3])
...     completed_custs.append(num_completed)

>>> N = sum(completed_custs) / len(completed_custs)
80.4
```

5. In an A&E department that is open for 24 hours, patients arrive according to a Poisson process at rate 35 per hour. Of these, $^1/_7$th are categorised as Triage category 1; $^2/_7$th are categorised as Triage category 2; and $^4/_7$the are categorised at Triage category 3. Patients wait to be seen by one of the 18 doctors on duty at the A&E, and they are seen by a doctor according to priorities: Triage 1 patients have priority over Triage 2 patients, who in turn have priority over Triage 3 patients. The time it takes to see each patient depends on their triage category: Triage 3 patients are given a 15 minute appointment; Triage 2 patients' time with a doctor is Uniformly distributed between 15 and 25 minutes; and Triage 1 patients' time with a doctor is Uniformly distributed between 20 and 90 minutes. What is the average waiting time for patients of each triage category?

Solution 5 *The conceptual model is given by the diagram below, where staggered arrival arrows represent priorities:*



The different triage categories can be dealt with by considering different customer classes. Using minutes as the time unit, we have that $\lambda_1 = {}^{5}/_{60}$ *due to Poisson thinning, and similarly* $\lambda_2 = {}^{10}/_{60}$ *and* $\lambda_3 = {}^{20}/_{60}$.
Therefore the Ciw network would be:

```
>>> import ciw
>>> N = ciw.create_network(
...     arrival_distributions={
...         'Triage 1': [ciw.dists.Exponential(rate=5/60)],
...         'Triage 2': [ciw.dists.Exponential(rate=10/60)],
...         'Triage 3': [ciw.dists.Exponential(rate=20/60)]
...     },
...     service_distributions={
...         'Triage 1': [ciw.dists.Uniform(20, 90)],
...         'Triage 2': [ciw.dists.Uniform(15, 25)],
...         'Triage 3': [ciw.dists.Deterministic(value=15)]
...     },
...     number_of_servers=[18],
...     priority_classes={'Triage 1': 0, 'Triage 2': 1, 'Triage 3': 2}
... )
```

using the `priority_classes` keyword to indicate priorities.
This is a steady-state problem, so we will choose a long run time of say a month, and a sufficient warmup time of say a day:

```
>>> ciw.seed(0)
>>> Q = ciw.Simulation(N)
>>> Q.simulate_until_max_time(31 * 24 * 60)
>>> all_recs = Q.get_all_records()
>>> recs = [r for r in all_recs if r.arrival_date > (24 * 60)]
```

Solution 5 (continuing from p. 6) *and we can find the average waiting time for each triage category:*

```
>>> wait_t1 = [r.waiting_time for r in recs if r.customer_class=='Triage 1']
>>> wait_t2 = [r.waiting_time for r in recs if r.customer_class=='Triage 2']
>>> wait_t3 = [r.waiting_time for r in recs if r.customer_class=='Triage 3']
>>> sum(wait_t1) / len(wait_t1)
0.17204197587155948
>>> sum(wait_t2) / len(wait_t2)
0.2501438257279699
>>> sum(wait_t3) / len(wait_t3)
0.6713875498450778
```