Desert Crossing

Callum Teape, Jet Hughes, Jake Norton, Cody Airey

Due on

Question 1.

Using the vehicle without refuelling, how far into the desert can you safely go?

Without refuelling, the vehicle has a capacity of 60L + 4x20L, which is 140L. The vehicle can travel 12km per 1L of fuel. This means without refuelling the vehicle can travel 12km x 140L = 1680km.

Question 2.

Describe a procedure whereby you could cross the desert in the vehicle.

To cross the desert, you could make several trips out into the desert, to drop fuel at "checkpoints" along the route. With enough fuel ready at points spaced close enough, you could make a trip across the desert by refuelling at the checkpoints along the way.

Question 3.

Describe a procedure whereby you could cross the desert and return in the vehicle.

To cross the desert and return, you could virtually the same procedure as in question 2. In order to facilitate a return trip, you would need to drop enough extra fuel at the checkpoints to be able to make a return trip while refuelling along the way.

Question 4.

Describe a procedure whereby you could cross the desert in the vehicle using the minimum amount of fuel.

Key assumptions (fuel closest to start) -> writing program to test different distributions of fuel drops -> finding optimal with exponential -> propose a solution for general distance.

Firstly, we note that the further into the desert you drop fuel, the amount of fuel you use getting there and back increases. So there is clearly some sort of cost associated with dropping fuel further out into the desert. With this in mind, we seek to maximise the distance between the fuel depot locations and the end of the desert. This leads us to a few critical assumptions:

1. We assume that the optimal solution will have the final fuel depot one full tank away from the end, as this is the maximum possible distance away from the end of the desert.
2. We assume that by the time we're done, there should be no fuel left in the desert. If there is then that means we've wasted fuel, and so our solution is not minimal.
3. We assume that every time we leave a depot driving out into the desert, we have full tank. This seems logical, as if we are going to be dropping fuel at the next depot, then we have to bear the cost of driving there, and we want to maximise the amount of fuel we drop per unit of cost to get the most out of every mile we drive. It doesn't make any sense to have only half a tank and then only be able to drop 20 litres, when we could have had a full tank and dropped 80 litres for the same driving cost.

4. We drop fuel at only one depot per trip from base. This assumption is more or less a consequence of 3), as it doesn't make sense to drop fuel at a location and then fuel up to a full tank to drive to another. This is because there is more driving cost associated with this move then simply leaving more fuel there in a previous trip (where we didn't drive as far from base).

5. As a consequence of 4), we also assume that we drop fuel at each depot only once. At a distance of this range, this assumption is ok. If the desert was much longer it wouldn't be possible to do this. This assumption seems logical, because we only drop fuel at one depot per trip, so two drop fuel at a depot twice means taking two whole separate trips to do it, which by 3) involves two refuels at base.

Using these assumptions, we try to find a solution.
Let $(a_n) = (a_1, a_2, ...a_n)$ be the locations of the fuel depots in litres from the start of the desert.

By 1), we fix a_n at $\frac{d-mc}{m}$

Now we are left with two questions: How many fuel depot's should we use between the start of the desert and the final drop? And how should these fuel depot's be distributed?

4) essentially gives us that we want the minimum possible fuel drops such that 5) holds.

3) give us that we need to have $d(a_n, a_{n-1})$ at the last depot. If we only drop fuel here once, and we are trying to maximise the distance between a_{n-1} and the end of the desert, then this give us that we have $3 * d(a_n, a_{n-1}) = c$. so $d(a_n, a_{n-1}) = \frac{c}{3}$.

For $d = 2413, c = 140, m = 12$, this gives us $(a_n) = (0, 14.417, 61.083)$.

Question 5.

Describe a procedure whereby you could cross the desert and return in the vehicle using the minimum amount of fuel.

Let the total length of the desert be d, let the mileage of the vehicle be m in km per litre, let the total fuel capacity of the vehicle be c.

We start by seeking some sort of standard procedure to cross the desert. With this we can find some sort of general formula for expenditure, and therefore can evaluate solutions programmatically rather than by hand. So we start with a few assumptions, which we came by logic and trial and error:

1. As before, we assume that the best solution has it's final fuel depot the maximum possible distance from the end of the desert, which in this case is half a tank. Which gives us that we need to have a full tank when we depart this depot for the end of the desert.

2. 2 from Q4)

3. 3 from Q4)

4. 4 from Q4)

5. On the final return journey, we assume that the optimum solution will have us run out of fuel exactly as we come to each fuel depot, which in turn gives us that to make it back, we need to leave at each fuel depot the exact amount of fuel that it takes to drive to the next. This seems logical, as it means we arrive back at base empty, implying we used the minimal fuel.

With these assumptions, we can work backwards from the finish line to construct a general procedure, and therefore a general expenditure. To simplify the explanation, let $(a_n) = (a_1, a_2, ...a_n)$ be the locations of the fuel depots in litres from the start of the desert, let c, m, d be as defined before.

By 1), we need a full tank when we leave the final depot for the end of the desert. By 4), we know that we have a full tank every time we leave the depot before the final depot so to have a full tank when we leave for the end, we will need to have $d(a_n, a_{n-1})$ litres there to refuel what we lost by driving there. By 2), we know that we will need to leave another $d(a_n, a_{n-1})$ litres at the final depot for the return journey. So we need $2 * d(a_n, a_{n-1})$ at the final depot.

By 4), we have a full tank every time we leave depot a_{n-1}, so we can drop $min(80, c - 2 * d(a_n, a_{n-1})$, litres at the final depot with every trip we make from the depot before. Therefore we need to make $\frac{2*d(a_n,a_{n-1})}{min(80,c-2*d(a_n,a_{n-1})}$ trips from depot a_{n-1} in order to drop enough fuel at the last depot.

By 4) we have a full tank every time we leave depot a_{n-2}, so to ensure that 4) holds for a_{n-1}, we need to leave enough there to refuel the amount lost by driving from the depot before however many times we will pass it. We know that we have to make $\frac{2*d(a_n,a_{n-1})}{min(80,c-2*d(a_n,a_{n-1})}$ trips beyond depot a_{n-1} to ensure there is enough fuel at the last depot for the final journey, and we also know that will pass depot a_{n-1} twice more (there and back) for that final journey. So the amount of fuel needed at depot a_{n-1} is $(2*\frac{2*d(a_n,a_{n-1})}{min(80,c-2*d(a_n,a_{n-1})}+2)*d(a_{n-1}, a_{n-2})$.

Now, by the same logic, we will need to make

$$\frac{(2*\frac{2*d(a_n,a_{n-1})}{min(80,c-2*d(a_n,a_{n-1})}+2)*d(a_{n-1},a_{n-2})}{min(80,c-2*d(a_{n-1},a_{n-2}))}$$

trips from depot a_{n-2} in order to drop of enough fuel, but we also will pass it $2 * \frac{2*d(a_n,a_{n-1})}{min(80,c-2*d(a_n,a_{n-1})}$ more when we make journeys from depot a_{n-1} to drop fuel at the final depot, and we also need to leave $2 * d(a_{n-2}, a_{n-3})$ for the final journey. So we require

$$(2 * \frac{(2*\frac{2*(d(a_n,a_{n-1})}{min(80,c-2*d(a_n,a_{n-1})}+2)*d(a_{n-1},a_{n-2})}{min(80,c-2*d(a_{n-1},a_{n-2}))} + 2 * \frac{2*d(a_n,a_{n-1})}{min(80,c-2*d(a_n,a_{n-1})} + 2) * d(a_{n-2}, a_{n-3})$$

litres of fuel at depot a_{n-2}.

We can express the logic above more concisely by iteratively defining a sequences (r_n), for the required fuel at each depot, and (t_n) for the number of trips from each depot. Then with initial conditions $r_0 = 2*d(a_n, a_{n-1})$ and $t_0 = \frac{2*d(a_n,a_{n-1})}{min(80,c-2*d(a_n,a_{n-1})}$.

$$r_i = d(a_{n-(i+1)}, a_{n-i})(2(\textstyle\sum_{k=0}^{i-1} t_k) + 2)$$

$$t_i = \frac{r_i}{min(80,c-2*d(a_{n-i},a_{n-(i+1)})}$$

Then we obtain a general formula for expenditure:

$$E = \textstyle\sum_{i=0}^{n} c * t_i$$

Finally, we can codify this in the following method:

```
def calcExpenditure5(a):
    """
    calculating the expenditure given an array of fuel drop locations for round trip.

    :params a: integer array
        the array of fuel drop positions in litres from the start of the desert
    """
    numTrips = 0

    tripcost = [] #array which holds the cost of each leg of the journey

    n = len(a)

    for k in range(1,n):
        j = n-k
        distance = a[j] - a[j-1]

        #if the distance between the points is greater than half of the fuel tank, we can't make
            it, so return FAIL
        if(distance*2 > 140):
            return "FAIL"

        if j == n-1: #first case for the last drop
            required = 2*distance
        else:
            required = (numTrips*2+2)*distance #this will be the numTrips from the prior iteration.


        thisTrip = math.ceil(required/min(80, (140-2*distance))) #how many trips does it take to
            drop the required amount of fuel at the spot, we can drop max of 80L, hence the min
            function?
        numTrips += thisTrip #we need enough fuel not only to drop the fuel at the next spot, but
            also for any trips beyond that from drop sites further up the chain, so need numtrips
            to be accumulative

        #building array to show cost at each step of the journey.
        tripcost.append(thisTrip*2*distance + required)


    return sum(tripcost)
```

Now that we have this method, we don't have to evaluate solutions by hand anymore.

Now to find the optimal solution, we again start by forcing assumption 1). Now, from the general formula for expenditure it is clear that the optimal solution minimises the amount of trips you make. Because more trips at the end of the desert compound throughout the chain of depots, leading to many more trips near the beginning, we again try to approach the problem working backwards from the end. So we try to minimise the number of trips between the last depot, and the second to last. Forcing assumptions 2) and 4), mean that we need to space the second to last depot a_{n-1} one quarter of the capacity away, $\frac{c}{4}$ from the last depot a_n. With this spacing, we can only have to make one trip to the last depot to drop half a tank, $\frac{c}{2}$. We apply the same philosophy to the spacing between a_{n-1} and a_{n-2} and make this distance also $\frac{c}{4}$. We do this all the way until the very first depot, which will simply be the distance left over from repeating the

process.

This leads to a general solution $(a_n) = (a_0, a_1, a_2, ...a_n)$, where

$$a_n = d - \frac{mc}{2}, \quad a_k = a_{k+1} - \frac{c}{4} \text{ for all } k \in [1, n-1] \subset \mathbb{N}, \quad a_0 = 0$$

Which has been codifed in the function below:

```
def evenDist(d, c, m):
last = (d - (1/2)*m*c)/m
current = last - c/4
a = [last]
while(current > 0):
    a.append(current)
    current = current - c/4

a.append(0)
a = list(reversed(a))
return a
```

Running this program with the input parameters in the question ($d = 2413, c = 140, m = 12$), we obtain the points $a = [0, 26.083, 61.083, 96.083, 131.083]$. Running the cost calculating algorithm specified earlier, we find that the expenditure of this set of points (given the standard procedure outlined earlier) is 1710.333 litres.

Alternate Solutions:

To give confidence in our theorised optimal solution, and also for the fun of it. We will try to beat it using some algorithmic generation of fuel depot locations.

We seek a way to distribute points according to some sort of ratio between them.

Let $(a_n) = (a_0, a_1, a_2, ...a_n)$ be the locations of the fuel depots in litres from the start of the desert as in the above.

Using the knowledge that the cost of dropping fuel further out increases, we decide that having the distance between fuel depots grow by some exponential function might yield a good solution. So we have $d(a_i - a_{i-1}) = g_i(e)d(a_{i-1} - a_{i-2})$. For the sake of simplicity, we assume that g_i only acts on e as an exponent, so we say $g_i e = e^{f(i)}$. We also stick with our first assumption that the last fuel depot should be as far away from the end of the desert as possible, giving $a_n = \frac{d - \frac{mc}{2}}{m}$. Putting all of these together gives:

$$a_0 = 0,$$

$$a_1 = \frac{d - cm}{m(1 + \sum_{k=1}^{n} e^{\sum_{j=1}^{k} f(j)})},$$

$$a_i = a_1(1 + \sum_{k=1}^{i} e^{\sum_{j=1}^{k} f(j)})) \text{ for } i \in [1, n] \subset \mathbb{N}$$

So now we essentially have an iterative function which distributes n fuel depots based on some exponential ratio for an arbitrary distance, capacity, and mileage.

The next question is how to choose n. Obviously we have to choose it such that the maximum distance $|a_n - a_{n-1}| \leq \frac{cm}{2}$, because otherwise we would run out of fuel. From there on out it becomes a question of how much fuel is it optimal to drop on each run. For instance, it seems very wasteful to drive 65 litres out and only be able to drop 10 litres in order to make it back. From the above we suspect that this number is a fraction of our fuel tank, $\frac{c}{4}$. In practice, we will just compute the expenditure $n = 1$ through 10 and take the minimum.

This distribution is automatically calculated using the functions below, with f(j) being the given exponential function.

```
(in main...)
    for n in (range(1,10)):
        lastDrop = (d-(1/2)*m*c)/m #half of full tank for the return journey, a full tank for the
            one way

        a1 = lastDrop/(eTerm(n))
        a = [0, a1]
        for i in range(2, n+2):
            a.append(a[i-1] + a1*eNum(i-1, fj))


def eTerm(n):
    result = 1
    for k in range(1,n+1):
        result += eNum(k)
```

```
    return result


def eNum(k):
    exponent = 0
    for j in range(1,k+1):
        exponent += f(j)

    return math.pow(math.e, exponent) #return ln(math.e) to get rid of the e and just return the
        exponent (for testing out non exponential trends)

def f(j):
    #some function
```

Now, we calculate expenditure using the same code as before, and we observe the data for many different functions f(j).

GRAPH of all general solutions (expenditure on y against distance on x)!