Using the Monte Carlo Process to Optimize Gasoline Inventory

Maggie Barker, Amy Lan
MAT 301 001

November 15, 2024

Introduction

We are using mathematical modeling techniques to determine how much gasoline should be delivered to a chain of gasoline stations, and how often deliveries should take place, in order to minimize cost and satisfy demand. In this model, we assume that the only sources of cost are delivery and storage, that the storage cost is proportional to the amount of gasoline stored, and that the delivery cost is constant per delivery.

We use a probabilistic submodel to approximate the daily demand. Using historical data of a particular gas station, we create a frequency table of the number of gallons demanded. From this table, we find the probability of each demand level, and build a cumulative probability table. We graph the cumulative probability histogram and approximate the cumulative probabilities with points, and then create a demand curve using cubic spline interpolation. Then we use the Monte Carlo process to find the randomized daily demand of a gas station.

We use an algorithm to simulate the daily inventory. This algorithm accounts for periodic deliveries of gasoline, with the same time period and quantity for each delivery. We tally the cost of delivery and storage to calculate the average daily cost. To prevent running out of gas (unfilled demands), we keep track of the number of days that our inventory runs out, and the number of gallons of unfilled demands.

We apply a range of delivery quantities and periods to simulate the average daily cost for each combination of delivery parameters. As we have two independent variables (delivery quantity and time between deliveries) and one dependent variable (average daily cost), we create a multivariable function, and we plot a 3D surface to investigate the effect of delivery parameters on cost. Our goal is to find the minimum value of this function.

Since we assume the only sources of cost are from delivery and storage, we can reason that a low cost would be achieved by buying a small amount of gas and waiting a long time between deliveries. A minimum cost ($0) could be achieved by never having any gas delivered, which would obviously not be conducive to a successful business operation. In fact, we would prefer to never run out of gas and be able to fulfill all demands. On the other hand, we don't want to buy too much gasoline, or buy it too frequently, which would drive up either storage or delivery costs, or both. We would prefer a "happy medium" where we buy just enough gas to fulfill demands. So in our model, we thus modify our cost surface to cut away those combinations of quantity and frequency that result in unfilled demands. Then we find the minimum point on this new cut surface. This minimum point gives us what we are looking for: The quantity of gas to have delivered, and the frequency of delivery, that will result in minimum cost and ensure that all of our demands are fulfilled.

As there is some randomness involved in the Monte Carlo process used in our simulation, our results are slightly different each time we run the simulation. We run the simulation many times and find the mean and standard deviation of our results. We then plot a histogram of our minimum costs to investigate the statistical distribution of our outcome.

Methods

We use the table of cumulative probabilities in Table 5.8 on page 206 of the textbook (Giordano, 2014) to start building our probabilistic demand model. We perform a cubic spline interpolation from selected cumulative probability points to create a demand curve that we can use later in our Monte Carlo process. We created a MATLAB script, `cubicsplines.m`, to perform the interpolation. (Our scripts are included at the end of the document.) We get the following cubic spline polynomials:

```
S1: 750826.2255x^3 + 0.0000x^2 + 4924.9174x + 1000.0000, [0.00, 0.01)
S2: -1501652.4510x^3 + 67574.3603x^2 + 4249.1738x + 1002.2525, [0.01, 0.03)
S3: 451817.6961x^3 + -108237.9529x^2 + 9523.5432x + 949.5088, [0.03, 0.08)
S4: -4918.9185x^3 + 1378.8346x^2 + 754.2002x + 1183.3579, [0.08, 0.20)
S5: 2475.9884x^3 + -3058.1096x^2 + 1641.5890x + 1124.1987, [0.20, 0.40)
S6: 140.7951x^3 + -255.8776x^2 + 520.6962x + 1273.6510, [0.40, 0.67)
S7: 5655.7074x^3 + -11340.8515x^2 + 7947.6287x + -385.0305, [0.67, 0.85)
S8: 11965.1108x^3 + -27429.8300x^2 + 21623.2604x + -4259.7929, [0.85, 0.93)
S9: 382648.2595x^3 + -1061635.8148x^2 + 983434.8263x + -302421.3783, [0.93, 0.97)
S10: -576340.2250x^3 + 1729020.6751x^2 + -1723501.9689x + 572821.5188, [0.97, 1.00)
```

In Figure 1 on page 3, we plot the cumulative probability points with the cubic spline polynomials to visually confirm that the splines are a good fit. Then we use another MATLAB script, `demand.m`, to evaluate our piecewise polynomial function for each random number from 0 to 1. We hardcoded the coefficients that resulted from `cubicsplines.m` to avoid recalculating the cubic splines every time we run the simulation. Our `demand` function has the following signature:

```
function q = demand(x)
% Calculates demand based on a random variable x
% x must be a scalar in the interval [0, 1]
```

In another MATLAB script, `inventory.m`, which functions as the core of our model, we perform the actual simulation. Our inventory function has the following signature:

```
function [c, L, D] = inventory(Q, T, d, s, N)
% Simulates inventory by Monte Carlo method
% Q: Delivery quantity of gasoline in gallons
% T: Time between deliveries in days
% d: Delivery cost in dollars per delivery
% s: Storage cost per gallon per day
% N: Number of days to run the simulation
% Returns [average daily cost, number of days with unfilled demands, amount of unfilled
↪  demand in gallons]
```

We create variables as follows:
`K`: Number of days remaining in the simulation
`I`: Current inventory in gallons
`C`: Total running cost
`L`: Number of days with unfilled demands
`D`: Amount of unfilled demands in gallons
`flag`: An indicator used to terminate the algorithm

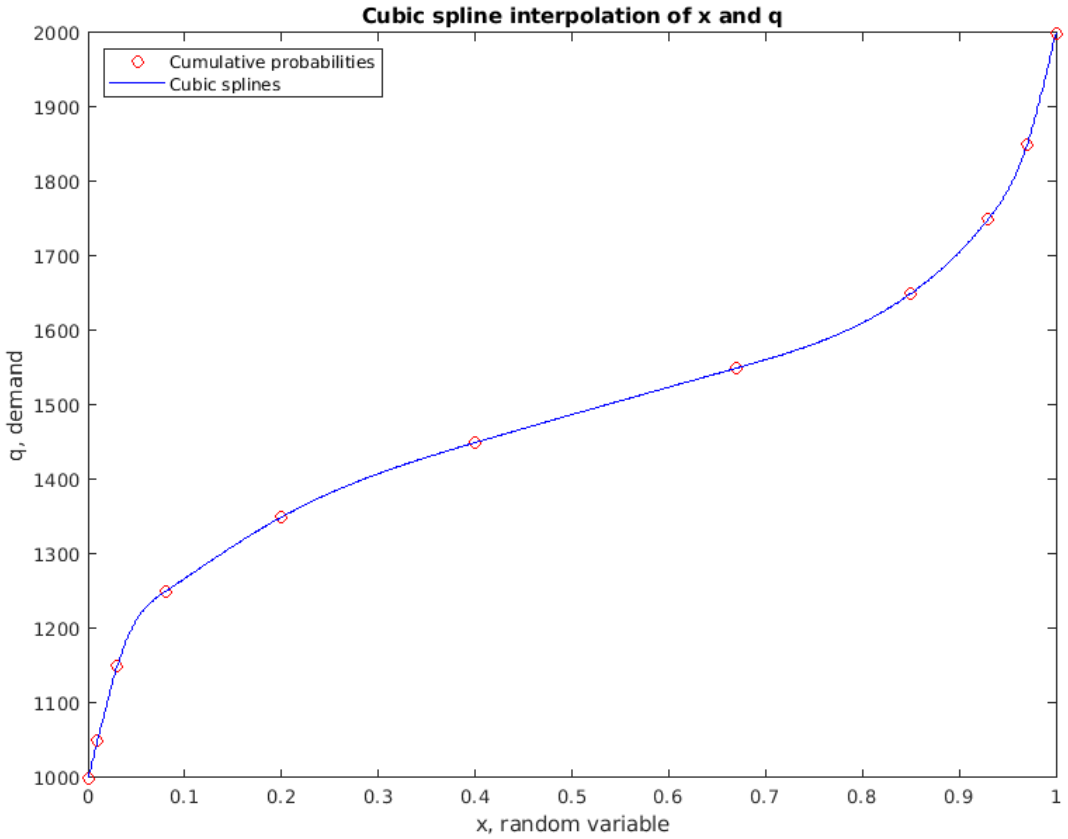

Figure 1: Cumulative probability plot, with cubic spline interpolating polynomials superimposed.

We initially set `K` to equal `N`, the number of days to run the simulation, and we set `I`, `C`, `L`, `D`, and `flag` to equal `0` initially. Then we use a `while` loop to run the simulation until we are ready to terminate it. We begin the cycle with a delivery, so we increase the inventory `I` by the quantity `Q` that was delivered, and we increase the total cost `C` by the delivery cost `d`. We check to see if the simulation will stop before our next delivery, and if so, we set the termination `flag` to `1` and we reduce the inventory cycle time `T` to be the number of days left in the simulation.

Then we use a `for` loop to iterate over the `T` days, which will either be the number of days between deliveries, or the number of days until the simulation ends, whichever is least. We generate a random number between `0` and `1`, and we call our `demand` function in `demand.m` to find the demand from our random variable using the piecewise spline polynomial. If our demand is more than our remaining inventory, we increment the counter `L` to keep track of the number of days that we have unfilled demands. We also increase `D` by the deficit to keep track of the number of gallons of unfilled demands. We will use this information later. Then we deduct the demand from the inventory. If we have completely depleted our inventory, we set our inventory to zero because we cannot have negative inventory. If we have any inventory remaining, we need to store it, incurring storage costs. So we increase our total cost `C` by the storage cost `I*s`, which is the amount of inventory in gallons multiplied by the storage cost per gallon. Then we reduce the number of days remaining, and the `for` loop repeats for the next day in the inventory cycle. Once the inventory cycle (or simulation) is over, the `while` loop checks if the termination `flag` has been set to `1`, indicating that the simulation should end. If not, we get another delivery and repeat the process until termination.

Once we terminate the simulation, we calculate the average daily cost by dividing the total running cost by the number of days simulated, and we return this value, along with the number of days of unfilled demands and the number of gallons unfilled.

Now we use another MATLAB script, `optimize.m`, to run our inventory script for a range of different delivery quantities and frequencies. Our `optimize` function has the following signature:

```
function [min_cost, min_Q, min_T] = optimize(d, s, N, plots)
% Finds optimum delivery quantity in gallons and time between deliveries in days to
↪ minimize average daily cost
% d: Delivery cost in dollars per delivery
% s: Storage cost per gallon per day
% N: Number of days to run the simulation
% plots: Whether to show surface plots or not
% Returns [minimum cost, quantity, time]
```

We need to make some assumptions to limit our ranges of `Q` and `T` (delivery quantity and period, respectively). Let us assume that the most frequently we should expect deliveries is every three days, and the least frequently is every thirty days. We only consider whether a delivery takes place during a given day or not, and fractional days are irrelevant, so the number of days between deliveries is an integer between 3 and 30. Considering that the daily demand ranges from 1,000 to 2,000 gallons, we want to find the extreme quantities that we should consider for delivery. At a minimum, if we have gas delivered every three days, and we expect the lowest demand each day, we would need 3,000 gallons delivered each time. At a maximum, if we have gas delivered every thirty days, and we expect the highest demand each day, we would need 60,000 gallons delivered each time. To make our cost surface smoother, we should have a small step between delivery quantities, but to make our code faster, we should have a large step. After trial and error, we found that the compromise step value of 500 works well. Thus, we have the following ranges of values to test:

```
Q = 3000:500:60000;
T = 3:1:30;
```

We create a large matrix (115 x 28) to hold the average daily cost for each value of `Q` and `T`. We also create a matrix of the same size for the number of days with unfilled demands, and a matrix for the number of gallons unfilled. Finally, we create a fourth matrix that will contain the average daily cost, but with those combinations of `Q` and `T` that result in unfilled demands removed. We use the following double `for` loop to run the simulation over all combinations of `Q` and `T` and populate our matrices:

```
    for i = 1:length(Q)
        for j = 1:length(T)
            [cost(i, j) unfilled_days(i, j) unfilled_gallons(i, j)] = inventory(Q(i), T(j), d,
↪   s, N);
            if unfilled_days(i, j) > 0
                cost_remove_unfilled(i, j) = NaN;
            else
                cost_remove_unfilled(i, j) = cost(i, j);
            end
        end
    end
```

Then we create four surface plots, all with quantity and period of deliveries as the independent variables, with the following dependent variables:

- Minimum average daily cost,
- Days with unfilled demands,
- Gallons of unfilled demands, and
- Minimum average daily cost (unfilled demands removed)

Next, we use the following code to find our minimum daily cost (with no unfilled demands) and with which combination of `Q` and `T` it occurs:

```
    min_cost = min(min(cost_remove_unfilled));
    [min_i min_j] = find(cost_remove_unfilled == min(min(cost_remove_unfilled)));
    min_Q = Q(min_i);
    min_T = T(min_j);
```

We return the coordinates `min_cost`, `min_Q`, and `min_T`, which represent the minimum value of the cost function and the point where it occurs.

We use a simple script to perform a single simulation, `singlesimulation.m`. However, there are still two questions we have not answered. How much does each delivery cost? What is our storage cost per gallon? We have assumed these to be constant, but we have not specified what values these constants should have. We used the information given in Project 2 on page 213 (Giordano, 2014) to find reasonable values of $500 per delivery and $0.001 per gallon stored. We also decided to run the simulation for 365 days at a time. Finally, we use the MATLAB stopwatch functions `tic` and `toc` to time the run of our algorithm.

```
    tic
    d = 500; % Delivery cost
    s = 0.001; % Storage cost
    N = 365; % Number of days
    clf
    [min_cost min_Q min_T] = optimize(500, 0.001, 365, true);
    fprintf("Delivery cost: %d\nStorage cost: %.3f\nDays simulated: %d\nMinimum average daily
↪   cost: $%.2f\nDelivery quantity: %d\nTime between deliveries: %d\n", d, s, N, min_cost,
↪   min_Q, min_T);
    toc
```

As addressed earlier, there is some variation in our results every time we run the algorithm, so we want to find an average of multiple runs. We do this in our MATLAB script `multiplesimulations.m`. We want to avoid making surface plots for every single run, so we have an argument in our optimize function to allow us to suppress the plots. We implemented a stopwatch and a progress indicator while the simulations are running, since it can take quite a while with many iterations. Then we output our results in a matrix, perform a quick statistical summary with the MATLAB `summary` command, and draw a histogram of our minimum average daily costs to visualize the distribution of our costs, which we might expect to be normal.

Finally, our average values of our minimum average daily cost, and the average values of `Q` and `T`, give us a best estimate of how much gasoline we should have delivered and how often to schedule deliveries. The standard deviations of these values give us an idea of the precision of our simulation.

Results

We are running the `singlesimulation.m` script to perform a single simulation and optimization. Our output is as follows:

```
Delivery cost: 500
Storage cost: 0.001
Days simulated: 365
Minimum average daily cost: $40.52
Delivery quantity: 38000
Time between deliveries: 25
Elapsed time is 1.573916 seconds.
```

We will discuss this output after we take a look at our surface plots.

Our surface plots are as follows:

- Figure 2 (page 7): Minimum average daily cost,
- Figures 3 (page 8) and 4 (page 9): Days with unfilled demands,
- Figure 5 (page 10): Gallons of unfilled demands, and
- Figure 6 (page 11): Minimum average daily cost (unfilled demands removed)

In Figure 2, we can see that the highest average daily cost (approximately $3,600) is incurred when deliveries take place every three days, and when 60,000 gallons are delivered each time. The dark blue flat region of the surface indicates a low average daily cost. This surface disregards the situation in which the station does not have enough gasoline to satisfy demand.

In Figure 3, we can see that for lower delivery quantities and longer delivery periods, there will be days that the station does not have enough gasoline to satisfy demand. In this particular instance, when 3,000 gallons of gasoline are delivered every 30 days, the station will have unfilled demands 342 out of 365 days (94%), which is clearly unacceptable. Interestingly, there is a straight line in the $Q - T$ plane, on one side of which, all demands are fulfilled, and on the other side, there are unfilled demands. We now rotate the plot so that the viewing angle is "top down", to get a better look at the $Q - T$ plane in Figure 4.

In Figure 4, we see more clearly the dividing line between fulfilled demands (the dark blue region) and unfilled demands (the multicolored region). This dividing line approximately follows the linear equation $Q = 1500T$. This makes sense, since the daily demand is simulated to be between 1,000 and 2,000 gallons per day, with an average demand near 1,500 gallons per day, as seen in Figure 1 on page 3.

Figure 5 looks similar to Figure 3, but it is showing the unfilled demand in *gallons* rather than the number of days with unfilled demands. Figure 3 is a graphical indicator of the variable `L` in the algorithm, while Figure 5 is an indicator of `D`. As may be expected, these graphs have a similar shape but a different scale.

In Figure 6, those combinations of delivery quantity and period that result in unfilled demands (the multi-colored region of Figure 4) have been cut away, leaving only those parts of the surface where all demands are fulfilled. Our goal is to find the minimum average daily cost with no unfilled demands, and that is represented by the lowest point on the surface in Figure 6.

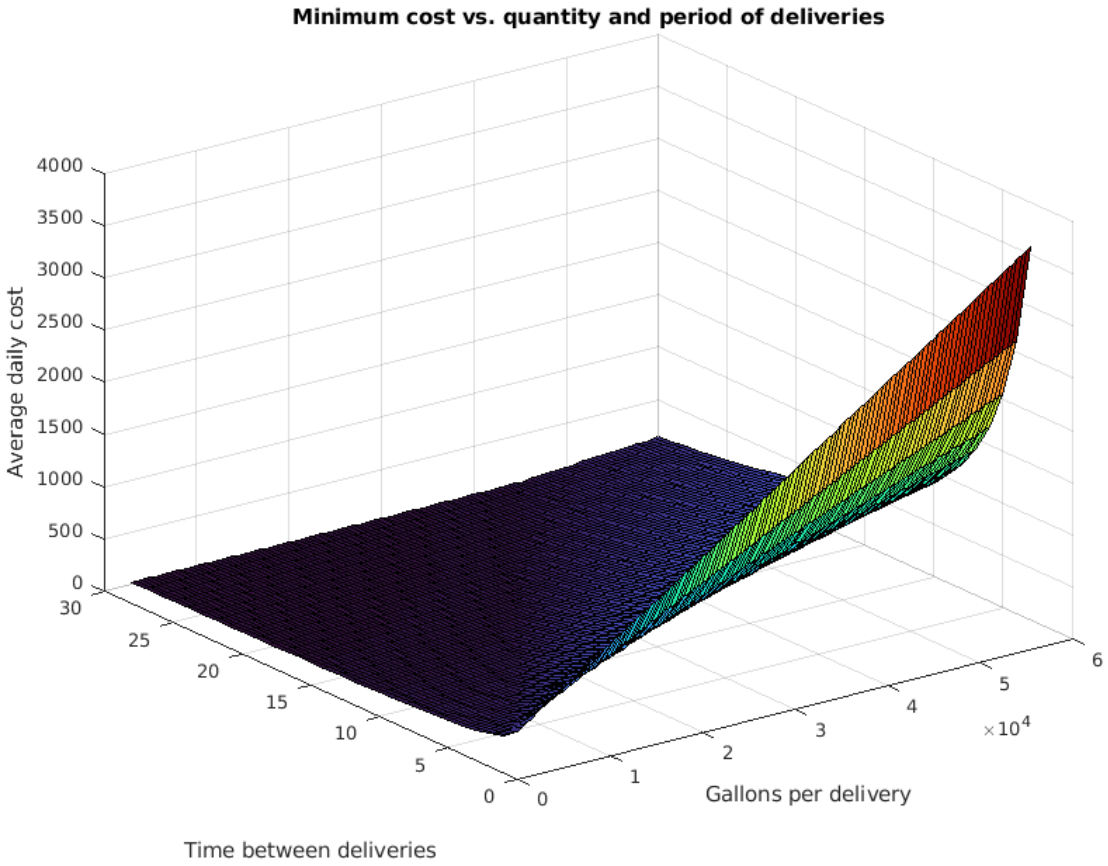

Figure 2: Minimum average daily cost vs. delivery quantity and period.

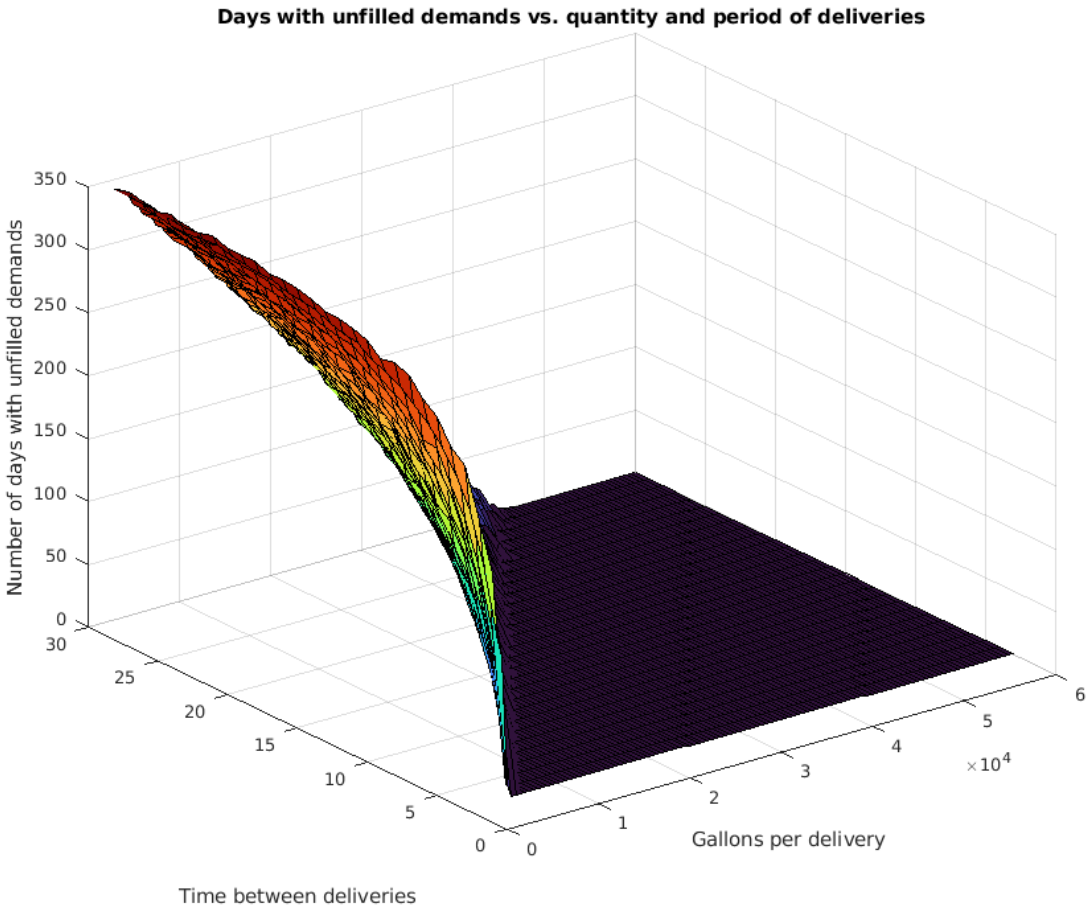

Figure 3: Number of days with unfilled demands vs. delivery quantity and period.

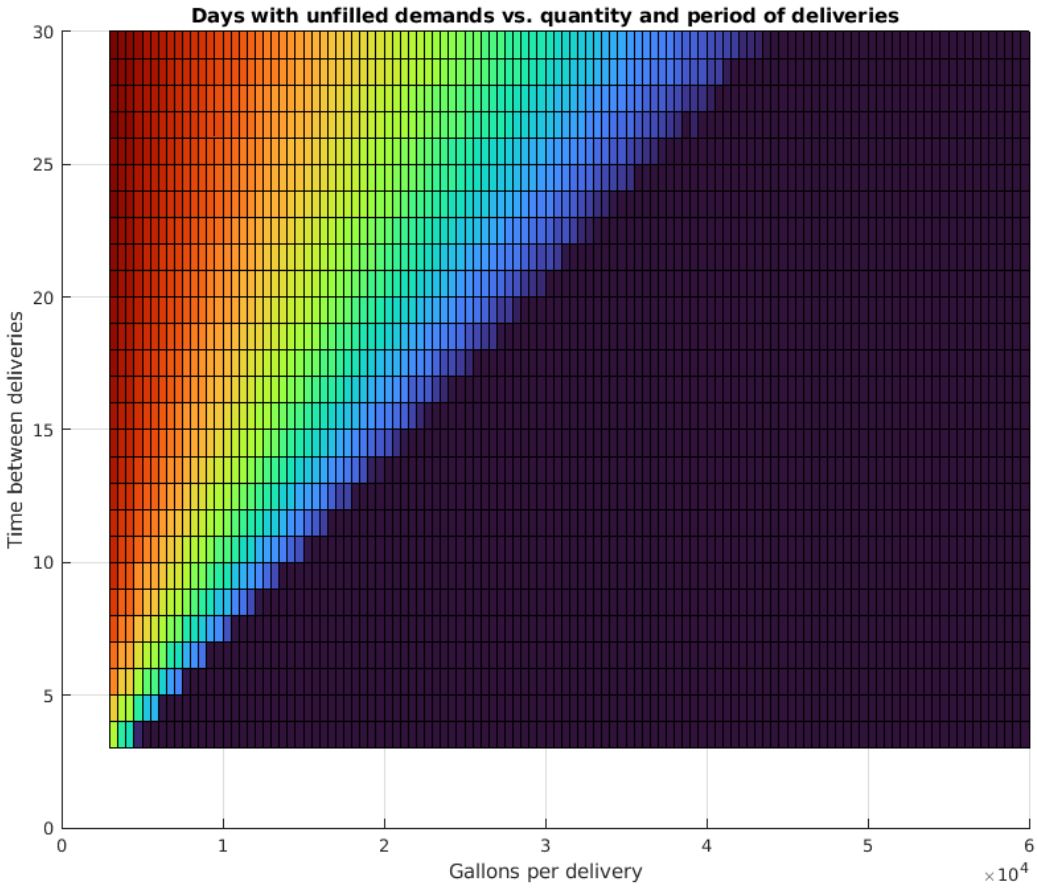

Figure 4: Number of days with unfilled demands vs. delivery quantity and period. This is the same surface as in Figure 3, with a different viewing aspect.

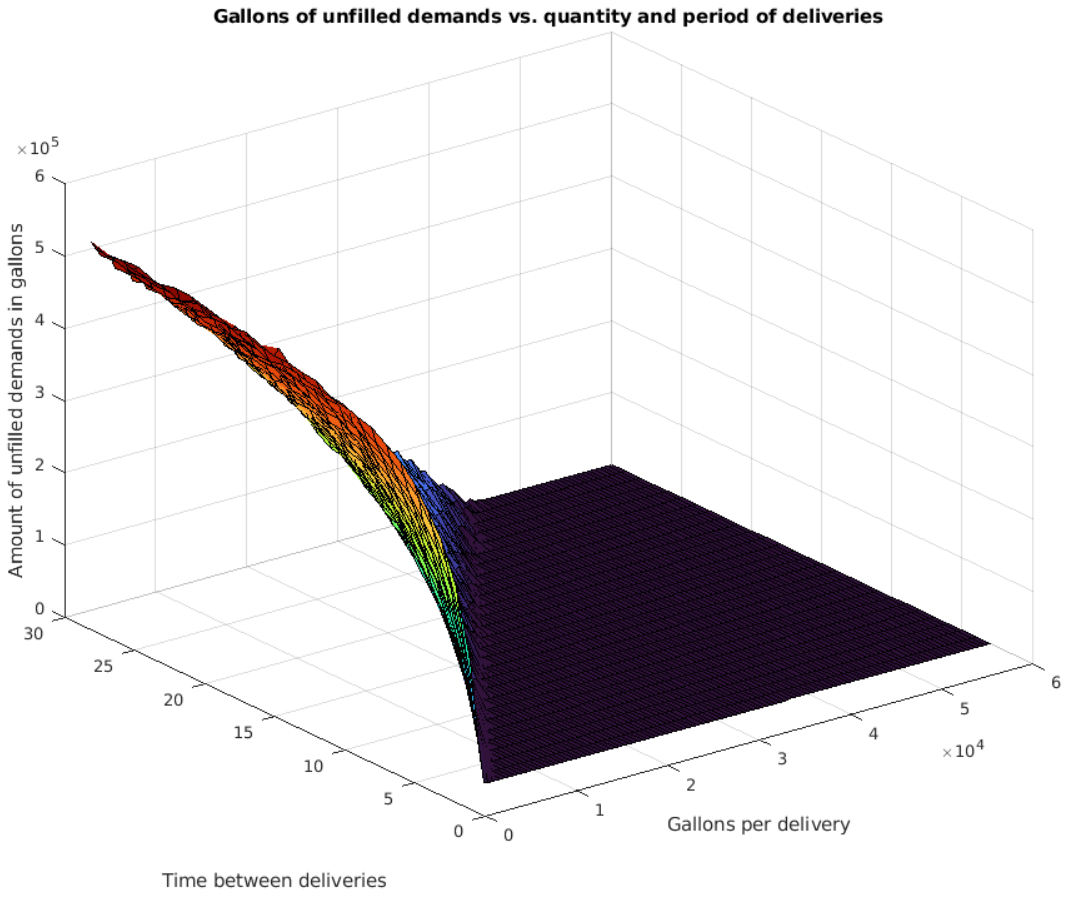

Figure 5: Gallons of unfilled demands vs. delivery quantity and period.

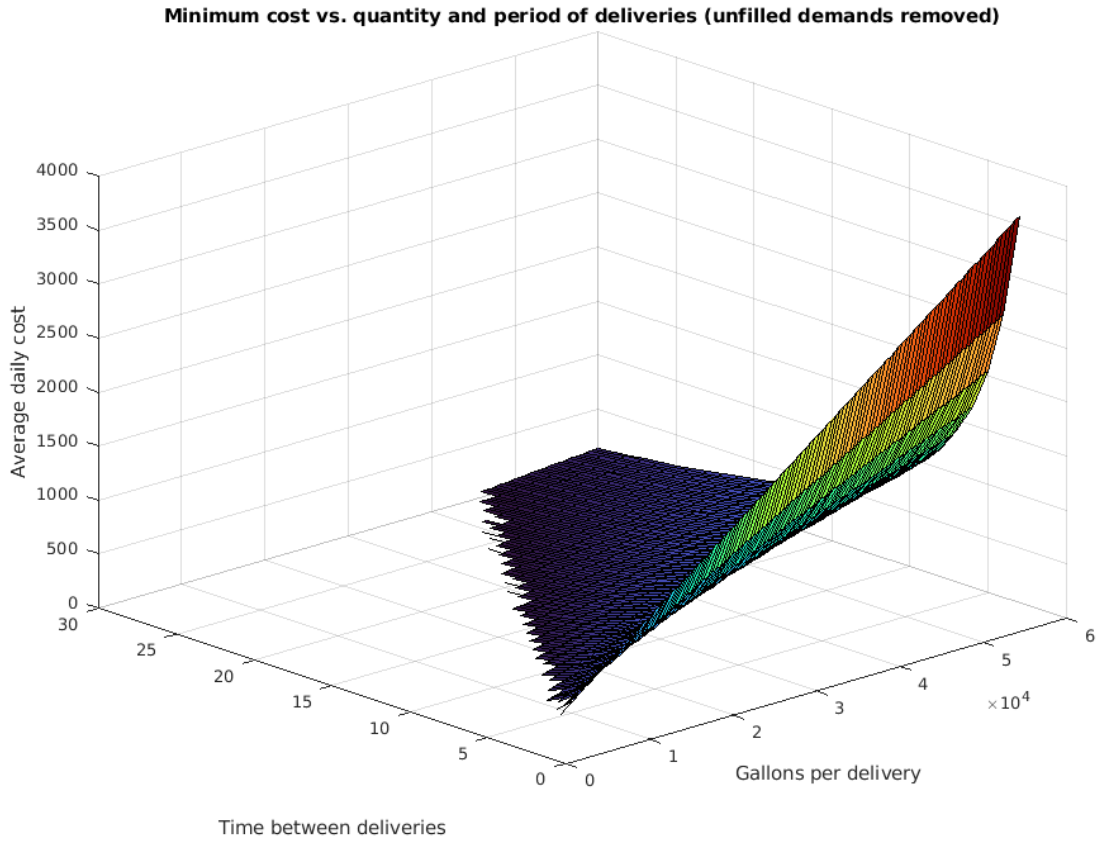

Figure 6: Minimum average daily cost vs. delivery quantity and period. Combinations of delivery quantity and period that result in unfilled demands have been cut away.

We return to the output from our script to analyze it further.

```
Delivery cost: 500
Storage cost: 0.001
Days simulated: 365
Minimum average daily cost: $40.52
Delivery quantity: 38000
Time between deliveries: 25
Elapsed time is 1.573916 seconds.
```

The delivery cost, storage cost, and days simulated are constants that we specified. The elapsed time shows that our simulation takes approximately 1.6 seconds to run, which is an important consideration when we want to run multiple simulations. The key insight given here is in lines 4 through 6. This shows that the minimum average daily cost with no unfilled demands is $40.52 (`min_cost`), and this occurs when deliveries of 38,000 gallons (‘`min_Q`‘) take place every 25 days (`min_T`). That is excellent information, but we still have one major concern. Every time we run the simulation and optimization process, we get a slightly different result. This variation stems from the randomness in the Monte Carlo demand simulation. The plots look roughly the same, so we will only consider the changes in numerical output. Here are the outputs from eight additional runs:

```
Minimum average daily cost: $40.66
Delivery quantity: 39000
Time between deliveries: 26

Minimum average daily cost: $41.16
Delivery quantity: 44000
Time between deliveries: 29

Minimum average daily cost: $40.72
Delivery quantity: 34500
Time between deliveries: 23

Minimum average daily cost: $41.07
Delivery quantity: 43000
Time between deliveries: 29

Minimum average daily cost: $40.63
Delivery quantity: 39000
Time between deliveries: 26

Minimum average daily cost: $40.89
Delivery quantity: 44500
Time between deliveries: 30

Minimum average daily cost: $40.50
Delivery quantity: 41500
Time between deliveries: 28

Minimum average daily cost: $41.37
Delivery quantity: 44500
Time between deliveries: 30
```

As can be seen, the variation is not very large, since the minimum average daily cost remains roughly around $41. The delivery quantities are approximately 35,000-45,000 gallons, and the delivery periods are around 23-30 days.

To get a better idea of the variation at play, we now run our `multiplesimulations.m` script to easily perform 1,000 runs and collect the data. Considering that one simulation and optimization takes around 1.6 seconds to perform, we figure this will take quite some time. To have a visual indication that the script is still running, we output a message after every simulation, along with a progress indicator.

```
Performing simulation 1 of 1000 (0.10%)...
Performing simulation 2 of 1000 (0.20%)...
Performing simulation 3 of 1000 (0.30%)...
Performing simulation 4 of 1000 (0.40%)...
Performing simulation 5 of 1000 (0.50%)...
Performing simulation 6 of 1000 (0.60%)...
Performing simulation 7 of 1000 (0.70%)...
Performing simulation 8 of 1000 (0.80%)...
Performing simulation 9 of 1000 (0.90%)...
Performing simulation 10 of 1000 (1.00%)...
```

After about 24 minutes, the final output is as follows, where the first column is minimum average daily cost (`min_cost`), the second column is delivery quantity (`min_Q`), and the third column is delivery period (`min_T`):

```
41.58      34000.00         23.00
40.46      38500.00         26.00
40.66      38500.00         26.00
40.62      33000.00         22.00
42.56      43500.00         29.00
41.95      36000.00         24.00
41.70      37500.00         25.00
40.96      34500.00         23.00
40.30      34500.00         23.00
41.40      41500.00         28.00
...
40.73      33000.00         22.00
39.89      38500.00         26.00
40.34      39000.00         26.00
40.29      36000.00         24.00
40.46      40000.00         27.00


results: 1000×3 double

NumMissing          0              0           0
Min             39.12       28000.00       19.00
Median          41.14       38000.00       25.00
Max             43.96       45500.00       30.00
Mean            41.19       38310.50       25.63
Std              0.81        3914.23        2.63

Elapsed time is 1453.711439 seconds.
```

Here we see the mean of the minimum average daily costs (N = 1,000) is $41.19, where the mean gallons per delivery is 38,310.50 gallons, and the mean days between deliveries is 25.63. Thus, it appears that the cost can be minimized by ordering around 38,000-39,000 gallons every 25-26 days, which is in close agreement with our initial run above. We note that the standard deviation of the minimum average daily cost is $0.81, the standard deviation of gallons per delivery is 3,914.23 gallons, and the standard deviation of time between deliveries is 2.63 days. We also note that the mean ($41.19) and median ($41.14) of the minimum average daily cost are approximately equal, indicating that the distribution is not significantly skewed. Finally, we look at a histogram of minimum average daily cost (Figure 7) to see that it is approximately normally distributed.

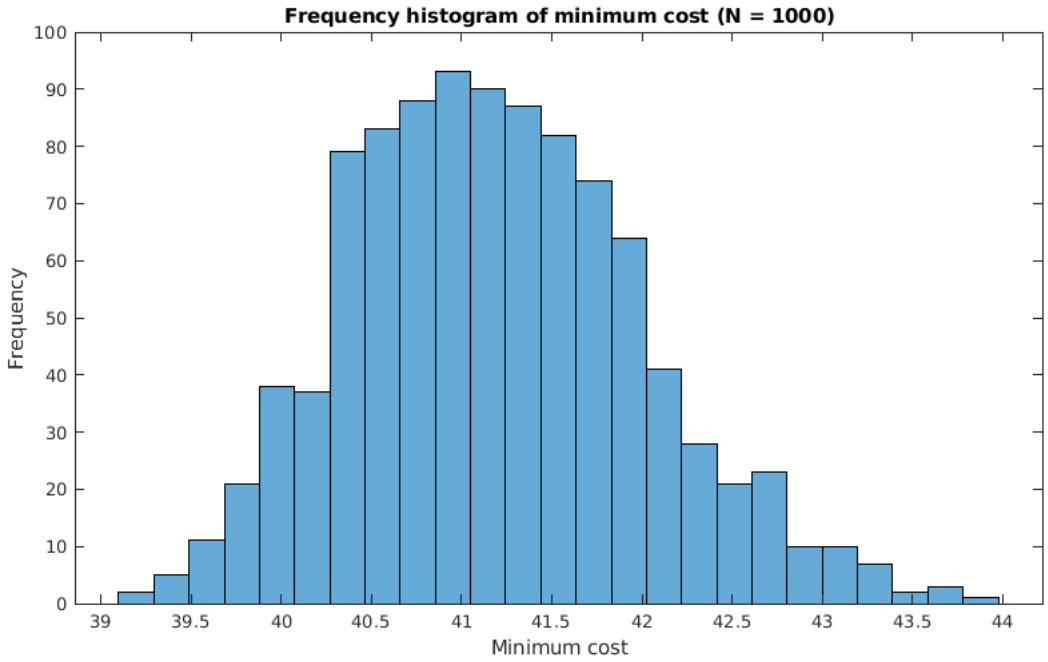

Figure 7: Frequency histogram of minimum average daily cost (N = 1,000). This reveals a symmetric distribution of minimum average daily cost that is approximately normal, with the mode in the bin from $40.855 to $41.05.

Conclusion

Scripts

Our MATLAB code is listed here, and it is downloadable from the following URL:
https://github.com/marautbar/mat-301-inventory-project

cubicsplines.m

```
x = [0, 0.01 0.03 0.08 0.2 0.4 0.67 0.85 0.93 0.97 1];
q = [1000, 1050:100:1850 2000];
A = zeros(40, 40);

% Set adjacent functions equal to the q values at the endpoints

A(1, 1:4) = [x(1).^3 x(1).^2 x(1) 1];
A(2, 1:4) = [x(2).^3 x(2).^2 x(2) 1];
A(3, 5:8) = [x(2).^3 x(2).^2 x(2) 1];
A(4, 5:8) = [x(3).^3 x(3).^2 x(3) 1];
A(5, 9:12) = [x(3).^3 x(3).^2 x(3) 1];
A(6, 9:12) = [x(4).^3 x(4).^2 x(4) 1];
A(7, 13:16) = [x(4).^3 x(4).^2 x(4) 1];
A(8, 13:16) = [x(5).^3 x(5).^2 x(5) 1];
A(9, 17:20) = [x(5).^3 x(5).^2 x(5) 1];
A(10, 17:20) = [x(6).^3 x(6).^2 x(6) 1];
A(11, 21:24) = [x(6).^3 x(6).^2 x(6) 1];
A(12, 21:24) = [x(7).^3 x(7).^2 x(7) 1];
A(13, 25:28) = [x(7).^3 x(7).^2 x(7) 1];
A(14, 25:28) = [x(8).^3 x(8).^2 x(8) 1];
A(15, 29:32) = [x(8).^3 x(8).^2 x(8) 1];
A(16, 29:32) = [x(9).^3 x(9).^2 x(9) 1];
A(17, 33:36) = [x(9).^3 x(9).^2 x(9) 1];
A(18, 33:36) = [x(10).^3 x(10).^2 x(10) 1];
A(19, 37:40) = [x(10).^3 x(10).^2 x(10) 1];
A(20, 37:40) = [x(11).^3 x(11).^2 x(11) 1];

% Set adjacent first derivatives equal to each other at the endpoints

A(21, 1:8) = [3.*x(2).^2 2.*x(2) 1 0 -3.*x(2).^2 -2.*x(2) -1 0];
A(22, 5:12) = [3.*x(3).^2 2.*x(3) 1 0 -3.*x(3).^2 -2.*x(3) -1 0];
A(23, 9:16) = [3.*x(4).^2 2.*x(4) 1 0 -3.*x(4).^2 -2.*x(4) -1 0];
A(24, 13:20) = [3.*x(5).^2 2.*x(5) 1 0 -3.*x(5).^2 -2.*x(5) -1 0];
A(25, 17:24) = [3.*x(6).^2 2.*x(6) 1 0 -3.*x(6).^2 -2.*x(6) -1 0];
A(26, 21:28) = [3.*x(7).^2 2.*x(7) 1 0 -3.*x(7).^2 -2.*x(7) -1 0];
A(27, 25:32) = [3.*x(8).^2 2.*x(8) 1 0 -3.*x(8).^2 -2.*x(8) -1 0];
A(28, 29:36) = [3.*x(9).^2 2.*x(9) 1 0 -3.*x(9).^2 -2.*x(9) -1 0];
A(29, 33:40) = [3.*x(10).^2 2.*x(10) 1 0 -3.*x(10).^2 -2.*x(10) -1 0];

% Set adjacent second derivatives equal to each other at the endpoints

A(30, 1:8) = [6.*x(2) 2 0 0 -6.*x(2) -2 0 0];
A(31, 5:12) = [6.*x(3) 2 0 0 -6.*x(3) -2 0 0];
A(32, 9:16) = [6.*x(4) 2 0 0 -6.*x(4) -2 0 0];
```

```
A(33, 13:20) = [6.*x(5) 2 0 0 -6.*x(5) -2 0 0];
A(34, 17:24) = [6.*x(6) 2 0 0 -6.*x(6) -2 0 0];
A(35, 21:28) = [6.*x(7) 2 0 0 -6.*x(7) -2 0 0];
A(36, 25:32) = [6.*x(8) 2 0 0 -6.*x(8) -2 0 0];
A(37, 29:36) = [6.*x(9) 2 0 0 -6.*x(9) -2 0 0];
A(38, 33:40) = [6.*x(10) 2 0 0 -6.*x(10) -2 0 0];

% Natural splines

A(39, 2) = 2;
A(40, 37:38) = [6.*x(11) 2];

% Solve the matrix-vector equation to get the coefficient vector c

b = [q(1) q(2) q(2) q(3) q(3) q(4) q(4) q(5) q(5) q(6) q(6) q(7) q(7) q(8) q(8) q(9) q(9) q(10)
↪  q(10) q(11) zeros(1, 20)]';
c = A\b;

% Plot original data points

plot(x, q, 'ro')

% Plot the splines

hold on
X = x(1):0.001:x(2);
plot(X, polyval(c(1:4), X), 'b-')
X = x(2):0.001:x(3);
plot(X, polyval(c(5:8), X), 'b-')
X = x(3):0.001:x(4);
plot(X, polyval(c(9:12), X), 'b-')
X = x(4):0.001:x(5);
plot(X, polyval(c(13:16), X), 'b-')
X = x(5):0.001:x(6);
plot(X, polyval(c(17:20), X), 'b-')
X = x(6):0.001:x(7);
plot(X, polyval(c(21:24), X), 'b-')
X = x(7):0.001:x(8);
plot(X, polyval(c(25:28), X), 'b-')
X = x(8):0.001:x(9);
plot(X, polyval(c(29:32), X), 'b-')
X = x(9):0.001:x(10);
plot(X, polyval(c(33:36), X), 'b-')
X = x(10):0.001:x(11);
plot(X, polyval(c(37:40), X), 'b-')
hold off
xlabel("x, random variable")
ylabel("q, demand")
legend("Cumulative probabilities", "Cubic splines")
legend("location", "northwest")
title("Cubic spline interpolation of x and q")
```

```
% Display the spline polynomials

fprintf("S1: %.4fx^3 + %.4fx^2 + %.4fx + %.4f, for [%.2f, %.2f)\n\n", c(1:4), x(1:2))
fprintf("S2: %.4fx^3 + %.4fx^2 + %.4fx + %.4f, for [%.2f, %.2f)\n\n", c(5:8), x(2:3))
fprintf("S3: %.4fx^3 + %.4fx^2 + %.4fx + %.4f, for [%.2f, %.2f)\n\n", c(9:12), x(3:4))
fprintf("S4: %.4fx^3 + %.4fx^2 + %.4fx + %.4f, for [%.2f, %.2f)\n\n", c(13:16), x(4:5))
fprintf("S5: %.4fx^3 + %.4fx^2 + %.4fx + %.4f, for [%.2f, %.2f)\n\n", c(17:20), x(5:6))
fprintf("S6: %.4fx^3 + %.4fx^2 + %.4fx + %.4f, for [%.2f, %.2f)\n\n", c(21:24), x(6:7))
fprintf("S7: %.4fx^3 + %.4fx^2 + %.4fx + %.4f, for [%.2f, %.2f)\n\n", c(25:28), x(7:8))
fprintf("S8: %.4fx^3 + %.4fx^2 + %.4fx + %.4f, for [%.2f, %.2f)\n\n", c(29:32), x(8:9))
fprintf("S9: %.4fx^3 + %.4fx^2 + %.4fx + %.4f, for [%.2f, %.2f)\n\n", c(33:36), x(9:10))
fprintf("S10: %.4fx^3 + %.4fx^2 + %.4fx + %.4f, for [%.2f, %.2f)", c(37:40), x(10:11))
```

demand.m

```
function q = demand(x)
    % Calculates demand based on a random variable x
    % x must be a scalar in the interval [0, 1]

    if length(x) > 1
        error('x must be a scalar')
        q = 0;
        return;
    end

    if x < 0 | x > 1
        error('x must be in the interval [0, 1]')
        q = 0;
        return;
    end

    % Cubic spline coefficients computed by cubicsplines.m
    splines = [
       % x^3 coefficient        x^2 coefficient        x coefficient          constant term
         7.508262254911285e+05  0                      4.924917377450891e+03  1000
↪   % S1  [0.00, 0.01)
        -1.501652450982267e+06  6.757436029420185e+04  4.249173774508872e+03
↪   1.002252478676473e+03 % S2  [0.01, 0.03)
         4.518176960804987e+05 -1.082379529414470e+05  9.523543171578338e+03
↪   9.495087847057786e+02 % S3  [0.03, 0.08)
        -4.918918507206453e+03  1.378834559602201e+03  7.542001714944003e+02
↪   1.183357931374684e+03 % S4  [0.08, 0.20)
         2.475988361845712e+03 -3.058109561829099e+03  1.641588995780660e+03
↪   1.124198676422266e+03 % S5  [0.20, 0.40)
         1.407951004512076e+02 -2.558776481556927e+02  5.206962303112980e+02
↪   1.273651045151514e+03 % S6  [0.40, 0.67)
         5.655707448149366e+03 -1.134085146702899e+04  7.947628688956410e+03
↪   -3.850305372792314e+02 % S7  [0.67, 0.85)
         1.196511079204448e+04 -2.742982999396154e+04  2.162326043684908e+04
↪   -4.259792865848703e+03 % S8  [0.85, 0.93)
```

```
         3.826482594740238e+05 -1.061635814816684e+06  9.834348263219807e+05
↪ -3.024213782902393e+05 % S9  [0.93, 0.97)
        -5.763402250302850e+05  1.729020675090855e+06 -1.723501968888332e+06
↪ 5.728215188277620e+05 % S10 [0.97, 1.00]
    ];

    if x < 0.01
        q = polyval(splines(1, :), x);
    elseif x < 0.03
        q = polyval(splines(2, :), x);
    elseif x < 0.08
        q = polyval(splines(3, :), x);
    elseif x < 0.20
        q = polyval(splines(4, :), x);
    elseif x < 0.40
        q = polyval(splines(5, :), x);
    elseif x < 0.67
        q = polyval(splines(6, :), x);
    elseif x < 0.85
        q = polyval(splines(7, :), x);
    elseif x < 0.93
        q = polyval(splines(8, :), x);
    elseif x < 0.97
        q = polyval(splines(9, :), x);
    else
        q = polyval(splines(10, :), x);
    end
end
```

inventory.m

```
function [c, L, D] = inventory(Q, T, d, s, N)
    % Simulates inventory by Monte Carlo method
    % Q: Delivery quantity of gasoline in gallons
    % T: Time between deliveries in days
    % d: Delivery cost in dollars per delivery
    % s: Storage cost per gallon per day
    % N: Number of days to run the simulation
    % Returns [average daily cost, number of days with unfilled demands, amount of unfilled
↪ demand in gallons]

    K = N; % K: Days remaining in the simulation
    I = 0; % I: Current inventory in gallons
    C = 0; % C: Total running cost
    L = 0; % L: Number of days with unfilled demands
    D = 0; % D: Amount of unfilled demand in gallons
    flag = 0; % flag: An indicator used to terminate the algorithm
    while flag == 0
        % Begin the next inventory cycle with a delivery
        I = I + Q; % Add delivery quantity to current inventory
        C = C + d; % Add delivery cost to running cost
```

```
        if T >= K % Determine if the simulation will terminate during this cycle
            T = K; % If we will stop the simulation before the next delivery, we reduce
                   % the cycle time to be the number of days left in the simulation
            flag = 1; % Stop the loop after this iteration
        end
        for i = 1:T % Simulate each day in the inventory cycle (or portion remaining)
            x_i = rand(); % A random number in the interval [0, 1]
            q_i = demand(x_i); % A daily demand (from cubic splines)
            if q_i > I % If there is not enough inventory to meet demand
                L = L + 1; % Increment the number of days with unfilled demands
                D = D + (q_i - I); % Increase amount of unfilled demands by deficit
            end
            I = I - q_i; % Deduct the demand from the current inventory
            if I <= 0 % Determine if the inventory has been depleted
                I = 0; % We cannot have negative inventory
            else
               C = C + I * s; % Any gasoline remaining in inventory needs to be stored, so add
    ↪  storage cost to running cost
            end
            K = K - 1; % Decrement the number of days remaining in the simulation
        end
    end
    c = C/N; % Return average daily cost
end
```

optimize.m

```
function [min_cost, min_Q, min_T] = optimize(d, s, N, plots)
    % Finds optimum delivery quantity in gallons and time between deliveries in days to
    ↪  minimize average daily cost
    % d: Delivery cost in dollars per delivery
    % s: Storage cost per gallon per day
    % N: Number of days to run the simulation
    % plots: Whether to show surface plots or not
    % Returns [minimum cost, quantity, time]

    % Q: Delivery quantity of gasoline in gallons
    Q = 3000:500:60000;

    % T: Time between deliveries in days
    T = 3:1:30;

    % Build the cost matrix from the inventory function
    % We need to use a nested for loop instead of a vectorized function to avoid a MATLAB
    ↪  warning

    cost = zeros(length(Q), length(T));
    unfilled_days = zeros(length(Q), length(T));
    unfilled_gallons = zeros(length(Q), length(T));
    cost_remove_unfilled = zeros(length(Q), length(T));

```

```
    for i = 1:length(Q)
        for j = 1:length(T)
            [cost(i, j) unfilled_days(i, j) unfilled_gallons(i, j)] = inventory(Q(i), T(j), d,
↪   s, N);
            if unfilled_days(i, j) > 0
                cost_remove_unfilled(i, j) = NaN;
            else
                cost_remove_unfilled(i, j) = cost(i, j);
            end
        end
    end

    if plots
        % Plot cost surface as a function of Q and T

        [TT, QQ] = meshgrid(T, Q);

        figure(1)
        surf(QQ, TT, cost)
        colormap turbo
        xlabel("Gallons per delivery")
        ylabel("Time between deliveries")
        zlabel("Average daily cost")
        title("Minimum cost vs. quantity and period of deliveries")

        figure(2)
        surf(QQ, TT, unfilled_days)
        colormap turbo
        xlabel("Gallons per delivery")
        ylabel("Time between deliveries")
        zlabel("Number of days with unfilled demands")
        title("Days with unfilled demands vs. quantity and period of deliveries")

        figure(3)
        surf(QQ, TT, unfilled_gallons)
        colormap turbo
        xlabel("Gallons per delivery")
        ylabel("Time between deliveries")
        zlabel("Amount of unfilled demands in gallons")
        title("Gallons of unfilled demands vs. quantity and period of deliveries")

        figure(4)
        surf(QQ, TT, cost_remove_unfilled)
        colormap turbo
        xlabel("Gallons per delivery")
        ylabel("Time between deliveries")
        zlabel("Average daily cost")
        title("Minimum cost vs. quantity and period of deliveries (unfilled demands removed)")
    end

    % Find minimum cost among those with no unfilled demands
```

```
    min_cost = min(min(cost_remove_unfilled));
    [min_i min_j] = find(cost_remove_unfilled == min_cost);
    min_Q = Q(min_i);
    min_T = T(min_j);
end
```

singlesimulation.m

```
tic
d = 500; % Delivery cost
s = 0.001; % Storage cost
N = 365; % Number of days
clf
[min_cost min_Q min_T] = optimize(d, s, N, true);
fprintf("Delivery cost: %d\nStorage cost: %.3f\nDays simulated: %d\nMinimum average daily
↪  cost: $%.2f\nDelivery quantity: %d\nTime between deliveries: %d\n", d, s, N, min_cost,
↪  min_Q, min_T);
toc
```

multiplesimulations.m

```
tic
d = 500; % Delivery cost
s = 0.001; % Storage cost
N = 365; % Number of days
trials = 1000;
results = zeros(trials, 3);
for i = 1:trials
    fprintf("Performing simulation %d of %d (%.2f%%)...\n", i, trials, (i / trials) * 100);
    [min_cost min_Q min_T] = optimize(500, 0.001, 365, false);
    results(i, :) = [min_cost, min_Q, min_T];
end
format bank
results
summary(results)
clf
figure(1)
histogram(results(:, 1), 25)
xlabel("Minimum cost")
ylabel("Frequency")
title("Frequency histogram of minimum cost (N = 1000)")
toc
```

References

Giordano, F., Fox, W., Horton, S. *A First Course in Mathematical Modeling, Fifth Edition.* Cengage Learning. 2014.