Simulations

Homework 4

M.S. in Stochastics and Data Science

Andrea Crusi, Lorenzo Sala

December 2, 2024

Exercise 1

1. To obtain the explicit expression for the distribution we need to substitute $m = 2$ in the probability distribution formula:

$$p(n) = \begin{cases} p(0)\left(\dfrac{\lambda}{\mu}\right)^n \dfrac{1}{n!} & \text{for } 0 \leqslant n \leqslant 2 \\ p(0)\left(\dfrac{\lambda}{\mu}\right)^n \dfrac{1}{2!2^{n-2}} = p(0)\left(\dfrac{\lambda}{\mu}\right)^n \dfrac{1}{2^{n-1}} & \text{for } n > 2. \end{cases}$$

2. We can use the fact that

$$\sum_{n=0}^{N} p(n) = 1$$

to find an analytical solution for $p(0)$. A simple way to do this is to check how $p(n)$ behaves for $N \to \infty$.

$$\sum_{n=0}^{2} p(0)\left(\frac{\lambda}{\mu}\right)^n \frac{1}{n!} + \sum_{n=3}^{\infty} p(0)\left(\frac{\lambda}{\mu}\right)^n \frac{1}{2!2^{n-2}} = 1.$$

By explicitly computing the first sum and rearranging the second one so that it starts from

$n = 0$ we can get

$$p(0)\left[1+\frac{\lambda}{\mu}+\frac{\lambda^2}{2\mu^2}+\sum_{n=3}^{\infty}\left(\frac{\lambda}{\mu}\right)^n\frac{1}{2^{n-1}}\right]=1$$

$$\begin{aligned}
\implies p(0) &= \frac{1}{1+\frac{\lambda}{\mu}+\frac{\lambda^2}{2\mu^2}+\sum_{n=3}^{\infty}\left(\frac{\lambda}{\mu}\right)^n\frac{1}{2^{n-1}}} = \\
&= \frac{1}{1+\frac{\lambda}{\mu}+\frac{1}{2}\left(\frac{\lambda}{\mu}\right)^2+\sum_{n=3}^{\infty}\left(\frac{\lambda}{\mu}\right)^n\frac{1}{2^{n-1}}} = \\
&= \frac{1}{1+\frac{\lambda}{\mu}\left[1+\frac{\frac{\lambda}{\mu}}{2}+\sum_{n=3}^{\infty}\left(\frac{\frac{\lambda}{\mu}}{2}\right)^{n-1}\right]} = \\
&= \frac{1}{1+\frac{\lambda}{\mu}\left[\underbrace{\sum_{n=0}^{\infty}\left(\frac{\frac{\lambda}{\mu}}{2}\right)^{n-1}}_{\text{geometric series}}\right]} = \\
&= \frac{1}{1+\frac{\lambda}{\mu}\left[\frac{1}{1-\frac{\frac{\lambda}{\mu}}{2}}\right]} = \\
&= \frac{1}{1+\frac{\lambda}{\mu}\left[\frac{2}{2-\frac{\lambda}{\mu}}\right]} = \\
&= \frac{2-\frac{\lambda}{\mu}}{2+\frac{\lambda}{\mu}}.
\end{aligned}$$

So our new distribution becomes

$$\begin{aligned}
p(n) &= \begin{cases} \frac{2-\frac{\lambda}{\mu}}{2+\frac{\lambda}{\mu}}\left(\frac{\lambda}{\mu}\right)^n\frac{1}{n!} & \text{for } 0 \leqslant n \leqslant 2 \\ \frac{2-\frac{\lambda}{\mu}}{2+\frac{\lambda}{\mu}}\left(\frac{\lambda}{\mu}\right)^n\frac{1}{2^{n-1}} & \text{for } n > 2 \end{cases} \\
&= \begin{cases} \frac{2-\frac{\lambda}{\mu}}{2+\frac{\lambda}{\mu}}\left(\frac{\lambda}{\mu}\right)^n\frac{1}{n!} & \text{for } 0 \leqslant n \leqslant 2 \\ \frac{2-\frac{\lambda}{\mu}}{2+\frac{\lambda}{\mu}}\left(\frac{\lambda}{\mu}\right)^n\frac{1}{2^{n-1}} & \text{for } n > 2 \end{cases}
\end{aligned}$$

which can also be expressed, by denoting $\frac{\lambda}{\mu} = \rho$ as the load of the server, as

$$p(n) = \begin{cases} \frac{2-\rho}{2+\rho} \left(\rho\right)^n \dfrac{1}{n!} & \text{for } 0 \leqslant n \leqslant 2 \\ \frac{2-\rho}{2+\rho} \left(\rho\right)^n \dfrac{1}{2^{n-1}} & \text{for } n > 2. \end{cases}$$

Exercise 2

We know that when a network comprises load independent stations the recursive equation for utilization in a station i is

$$\overline{n_i}(n) = U_i[1 + \overline{n_i}(n-1)].$$

We are interested in what happens when n goes to ∞ so

$$\overline{n_i} = U_i[1 + \overline{n_i}] \implies \overline{n_i} - U_i\overline{n_i} = U_i \implies \overline{n_i} = \frac{U_i}{1 - U_i}.$$

In bottleneck stations b, where the utilization U_b tends to 1 due to the saturation of the system, the queue length will tend to infinity. Moreover, we know that, by consistency laws,

$$\frac{U_b}{U_i} = \frac{V_b S_b}{V_i S_i} = \frac{D_b}{D_b} \implies U_i = \frac{D_i U_b}{D_b}$$

and therefore

$$\overline{n_i} = \frac{\frac{D_i U_b}{D_b}}{\frac{D_b - D_i U_b}{D_b}} = \frac{D_i U_b}{D_b - D_i U_b}$$

but since as load increases $U_b \to 1$ we get

$$\overline{n_i} = \frac{D_i}{D_b - D_i}$$

for non-bottleneck stations.
Let's turn to average waiting time. Little's Law tells us that

$$\overline{n_i} = \overline{w_i} X_i \implies \overline{w_i} = \frac{\overline{n_i}}{X_i}$$

but we know that as the load increases $\overline{n_i} \to U_i[1 + \overline{n_i}]$ so

$$\overline{w_i} = \frac{U_i[1 + \overline{n_i}]}{X_i}.$$

In bottleneck stations, as the queue goes to ∞, so does the average waiting time. Remember that for load independent stations we have $U_i = X_i(n) \cdot S_i$, so

$$\overline{w_i} = \frac{\cancel{X_i(n)} S_i[1 + \overline{n_i}]}{\cancel{X_i(n)}} = S_i[1 + n_i].$$

Now let's take into account the throughput. As we said before, by Little's Formula we know that

$$\overline{n_i} = \overline{w_i} X_i \implies X_i(n) = \frac{\overline{n_i}}{\overline{w_i}}$$

and by substituting we get

$$X_i(n) = \frac{\overline{n_i}}{S_i[1 + n_i]}.$$

In bottleneck stations, as $\overline{n_i}$ becomes increasingly larger, $[1 + \overline{n_i}]$ becomes closer and closer to $\overline{n_i}$ so that the throughput for bottleneck stations will tend to

$$\frac{1}{S_i}.$$

Exercise 3

The normalization constant is defined in two ways:

$$\underbrace{g(n,m) = \sum_{\mathbf{n} \in S(n,m)} \prod_{i=1}^{m} f_i(n_i)}_{\text{definition}} \qquad \underbrace{\sum_{k=0}^{n} f_m(k) g(n-k, m-1).}_{\text{convolution method}}$$

We know that the distribution of $p_i(k)$, which is the fraction of time the station i spends with k customers inside, can be computed as

$$\begin{aligned}
p_i(k) &= \sum_{\substack{\mathbf{n} \in S(n,i) \\ n_i = k}} P(\mathbf{n}) \\
&= \sum_{\substack{\mathbf{n} \in S(n,i) \\ n_i = k}} \frac{1}{g(n,i)} \prod_{j=1}^{M} f_j(n_j) \\
&= \frac{1}{g(n,i)} \sum_{\substack{\mathbf{n} \in S(n,i) \\ n_i = k}} \prod_{j=1}^{M} f_j(n_j) \\
&= \frac{f_i(k)}{g(n,i)} \underbrace{\sum_{\mathbf{n} \in S(n-k,i-1)} \prod_{j=1}^{i-1} f_j(n_j)}_{g(n-k,i-1)} \qquad \text{since } n_i = k \text{ for all states we can take it out of the sum} \\
&= \frac{f_i(k) g(n-k, i-1)}{g(n,i)}
\end{aligned}$$

To get $p_i(k, N)$ we must sum the probabilities for station i to have k customers over all possible distribution in the other station with a maximum of N customers. This happens over the reduced state space

$$S^{[-i]}(N, M)$$

(which is basically the whole state space of N maximum clients and M stations but with the number of customers in i-th station fixed to 0) and with the reduced normalization constant with the i-th station missing over the reduced state space

$$g^{[-i]}(N, M) = \sum_{\mathbf{n} \in S^{[-i]}(N,M)} \prod_{j \neq i} f_j(n_j).$$

So, in a similar manner as before, we get

$$\begin{aligned} p_i(k, N) &= \sum_{\mathbf{n} \in S^{[-i]}(N,i)} P(\mathbf{n}) \\ &= \frac{f_i(k)}{g(N,i)} \underbrace{\sum_{\mathbf{n} \in S^{[-i]}(N-k,i-1)} \prod_{j=1}^{i-1} f_j(n_j)}_{g^{[-i]}(N-k,i-1)} \\ &= \frac{f_i(k) g^{[-i]}(N-k, i-1)}{g(N,i)}. \end{aligned}$$

The service function $f_i(k)$ is defined as

$$f_i(k) = \begin{cases} 1 & k = 0 \\ V_i S_i f_i(k-1) & k > 0. \end{cases}$$

This allows us to write

$$\begin{aligned} p_i(k, N) &= \frac{f_i(k) g^{[-i]}(N-k, i-1)}{g(N,i)} \\ &= V_i S_i f_i(k-1) \cdot \frac{g^{[-i]}(N-k, i-1)}{g(N,i)} \\ &= V_i S_i f_i(k-1) \cdot \frac{g^{[-i]}(N-k, i-1)}{g(N,i)} \cdot \frac{g(N-1,i)}{g(N-1,i)} \\ &= \underbrace{V_i \frac{g(N-1,i)}{g(N,i)}}_{X_i(N)} \cdot S_i \cdot \underbrace{\frac{f_i(k-1) g^{[-i]}(\overbrace{(N-1)-(k-1)}^{N-k}, i-1)}{g(N-1,i)}}_{p_i(k-1,N-1)} \\ &= X_i(N) S_i p_i(k-1, N-1). \end{aligned}$$

Exercise 4

This code computes both the MVA computations and the bottleneck analysis computations.

```c
/* ---------------------------------------------------------------------------
Università di Torion
M.S. in STOCHASTICS AND DATA SCIENCE
Course in Simulation
Homework 4

By Andrea Crusi and Lorenzo Sala
* ---------------------------------------------------------------------------
*/
#include <stdio.h>

// Constants
#define M 4     // Number of stations
#define N 80  // Maximum number of customers

// Input parameters
double Z = 10.0;  // Delay time of the station
double S[M] = {0, 0.04, 0.06, 0.04};  // Service times
char ST[M] = {'D','L', 'L', 'L',};  // Station types ('L' for load independent, 'D' for delay.
↪  Our reference station is the delay)
double Q[M][M] = {
        {0, 1, 0, 0},
        {0.1,0,0.55,0.35},
        {0,1,0,0},
        {0,1,0,0}
};

double V[M];

// We need bi-dimensional arrays for results
double X[M][N+1];
double U[M][N+1];
double n[M][N+1];
double w[M][N+1];
double R[M][N+1];

int main() {
        // Initialization
        for (int i = 0; i < M; i++) {
                n[i][0] = 0.0;  // put 0 as the value for all stations
        }

        // Compute visit count
        V[0]=1.0;
        V[1]=V[0]/Q[1][0];
        V[2]=V[1]*Q[1][2];
        V[3]=V[1]*Q[1][3];

        // Compute performance measures for each population size k (from 1 to N)
        for (int k = 1; k <= N; k++) {

                // Compute the waiting time w_i[k] for each station i
                for (int i = 0; i < M; i++) {
                        if (ST[i] == 'D') {
                                w[i][k] = Z;  // Delay station
                        } else {
                                w[i][k] = S[i] * (1 + n[i][k - 1]);  // Queue station
                        }
                }
                double sum = 0.0; // initialize sum


                // Compute the sum of Vi * wi[k] across all stations
                for (int i = 0; i < M; i++) {
```

```
                sum += V[i] * w[i][k];
            }

            // Compute throughput for reference job. Remember that the reference station is the station 0 (delay station)
            double Xref = k / sum;

            // Compute performance metrics for each station i using MVA equations seen in class
            for (int i = 0; i < M; i++) {
                X[i][k] = V[i] * Xref;  // Throughput for station i

                if (ST[i] == 'D') {
                    // Delay station
                    n[i][k] = Z * X[i][k];
                    U[i][k] = n[i][k] / k;
                } else {
                    // Computational station
                    U[i][k] = S[i] * X[i][k];  // Utilization
                    n[i][k] = U[i][k] * (1 + n[i][k - 1]);  // Average queue length
                }
            }
            // Compute the response time
            double Y[M]={0.0,0.0,0.0,0.0};
            for (int i=0; i < M; i++) {

                for (int i = 0; i < M; i++) {
                    Y[i] += V[i] * w[i][k];
                }
                if (ST[i] == 'D') {
                    R[i][k] = (k/X[i][k])-Z; //formula for delay station
                } else {
                    R[i][k] = w[i][N]+S[i];
                }
            }
        }
        int n_customers[2] = {1,N};
        // Print results for N = 1 and N = 80;
        for (int j=0; j<=1; j++){
            printf("-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*\n\n");
            printf("Simulation with %d customers\n\n", n_customers[j]);
            for (int i = 0; i < M; i++) {
                printf("Station %d results:\n", i);
                printf("Throughput (X[%d])\t\t= %f\n", n_customers[j], X[i][n_customers[j]]);
                printf("Utilization (U[%d])\t\t= %f\n", n_customers[j], U[i][n_customers[j]]);
                printf("Mean queue length (n[%d])\t= %f\n", n_customers[j], n[i][n_customers[j]]);
                printf("Mean waiting time (w[%d])\t= %f\n", n_customers[j], w[i][n_customers[j]]);
                //printf("Mean response time (R_0)\t\t= %f\n", (n_customers[j]/X[i][n_customers[j]])-Z);
                printf("Mean response time (R_0)\t\t= %f\n", (R[i][n_customers
                [j]]));
                printf("\n");
            }
        }

        double X_0_1=X[0][1]; // We will need this later to compute the saturation point

        // We now make a pair of csv files that we will plot with pgfplots in the homework discussion
        FILE *file = fopen("x0.csv", "w");
```

```
        if (file == NULL) {
                fprintf(stderr, "Error opening file for writing.\n");
                return 1;
        }

        for (int k = 1; k <= N; k++) {
                fprintf(file, "%d %f\n", k, X[0][k]);
        }

        fclose(file);
        printf("Data written to x0.csv successfully.\n");

        // We now save the results of the response time for station 0
        FILE *file2 = fopen("R0.csv", "w");
        if (file2 == NULL) {
                fprintf(stderr, "Error opening file for writing.\n");
                return 1;
        }

        for (int k = 1; k <= N; k++) {
                fprintf(file2, "%d %f\n", k, R[0][k]);
        }

        fclose(file2);
        printf("Data written to R0.csv successfully.\n");

        // We now save the results of average queue length
        FILE *file3 = fopen("queue.csv", "w");
        if (file3 == NULL) {
                fprintf(stderr, "Error opening file for writing.\n");
                return 1;
        }

        fprintf(file3, "n station0 station1 station2 station3\n");

        for (int k = 1; k <= N; k++) {
                fprintf(file3, "%d %f %f %f %f\n", k, n[0][k],n[1][k],n[2][k],n[3][k]);
        }

        fclose(file3);
        printf("Data written to queue.csv successfully.\n");

        // We now save the results of utilization
        FILE *file4 = fopen("utilization.csv", "w");
        if (file4 == NULL) {
                fprintf(stderr, "Error opening file for writing.\n");
                return 1;
        }

        fprintf(file4, "n station0 station1 station2 station3\n");

        for (int k = 1; k <= N; k++) {
                fprintf(file4, "%d %f %f %f %f\n", k, U[0][k],U[1][k],U[2][k],U[3][k]);
        }

        fclose(file4);
        printf("Data written to utilization.csv successfully.\n");

        // Now we perform the bottleneck analysis using the equations derived in the previous
        ↪  exercise.
        printf("\n");

        double D[M];             // Service demands for each station
        double Xb;             // Variable to store the max throughput of the bottleneck station
```

```c
        double X[M];            // Throughput
        double U[M];            // Utilization for each station
        double n[M];            // Mean queue length for each station
        double w[M];            // Mean waiting time for each station

        double Db = 0.0;
        int bottleneck_station = 0;

        // Calculate service demand for each station and identify the bottleneck as the station with higher service demand
        for (int i = 0; i < M; i++) {
                D[i] = V[i] * S[i];
                if (D[i] > Db) {
                        Db = D[i];
                        bottleneck_station = i;
                }
        }

        // Maximum throughput as N -> infinity: we can use this formula because we are in a closed system
        Xb = 1.0 / Db;

        // Calculate utilization, mean queue length, and mean waiting time for each station
        for (int i = 0; i < M; i++) {
                X[i] = V[i]*Xb; //our reference station is the bottleneck station
                U[i] = X[i] * S[i];

                if (ST[i] == 'D') {
                        // For the delay station
                        n[i] = Z * Xb;
                        w[i] = Z;
                } else {
                        // For load independent station stations
                        n[i] = U[i] / (1 - U[i]);    // Mean queue length
                        w[i] = n[i] / Xb;         // Mean waiting time
                }
        }

        // Print the bottleneck analysis results
        printf("=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+\n");
        printf("Bottleneck Analysis (using formulas of the previous exercise)\n");
        printf("=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+\n\n");
        printf("The bottleneck station is %d, with the highest service demand that is %f\n", bottleneck_station, Db);
        printf("Maximum Throughput (Xb) as N -> infinity: %f\n\n", Xb);

        // Print results for each station
        for (int i = 0; i < M; i++) {
                printf("Station %d:\n", i);
                printf("  Service Demand (D_%d(N))       = %f\n", i, D[i]);
                printf("  Throughput (X_%d(N))           = %f\n", i, X[i]);
                printf("  Utilization (U_%d(N))          = %f\n", i, U[i]);
                printf("  Mean Queue Length (n_%d(N))    = %f\n", i, n[i]);
                printf("  Mean Waiting Time (w_%d(N))    = %f\n\n", i, w[i]);
        }
        printf("_._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._.\n");
        printf("For plotting purposes:\n");
        printf("Vb*Sb=Db=%f\n",Db);
        printf("Point of saturation N_star: %f clients",(1/(X_0_1*Db)));
        return 0;
}
```

This prints the following results for the simulation with $N = 1$:

	Station 0	Station 1	Station 2	Station 3
Throughput ($X_i(1)$)	0.091996	0.919963	0.505980	0.321987
Utilization ($U_i(1)$)	0.919963	0.036799	0.030359	0.012879
Mean queue legnth ($\overline{n_i}(1)$)	0.919963	0.036799	0.030359	0.012879
Mean waiting time ($\overline{w_i}(1)$)	10.000000	0.040000	0.060000	0.040000
Mean response time (R_0)	0.870000	0.870000	0.870000	0.870000

For $N = 80$ we have:

	Station 0	Station 1	Station 2	Station 3
Throughput ($X_i(80)$)	2.499979	24.999794	13.749887	8.749928
Utilization ($U_i(80)$)	0.312497	0.999992	0.824993	0.349997
Mean queue legnth ($\overline{n_i}(80)$)	24.999794	49.749403	4.712349	0.538454
Mean waiting time ($\overline{w_i}(80)$)	10.000000	1.989992	0.342719	0.061538
Mean response time (R_0)	22.000263	22.000263	22.000263	22.000263

Finally, for the bottleneck analysis (where we have $N \to \infty$) we obtain:

	Station 0	Station 1	Station 2	Station 3
Service demand ($D_i(N)$)	0	0.4	0.33	0.14
Throughput ($X_i(N)$)	2.5	25	13.75	8.75
Utilization ($U_i(N)$)	0	1	0.825	0.35
Mean queue legnth ($\overline{n_i}(N)$)	25	∞	4.714286	0.538462
Mean waiting time ($\overline{w_i}(N)$)	10	∞	1.885714	0.215385

We know that the bottleneck of the system is station 1, since it has the highest service demand. We can see how results for $N = 80$ for the MVA computations are consistent with the bottleneck analysis: the mean queue length $\overline{n_1}$ tends to ∞ as N grows while the other values of performance tend to their respective value in the bottleneck analysis.
We now plot the data gained from the simulation along with the expected asymptotes. The value $N^\star = 27.175000$ is obtained from the simulation, being simply the point of intersection of the two asymptotes in either case. In other words. this is what we would expect if our calculations are correct.

standalone [x11names,table]xcolor tikz pgfplots

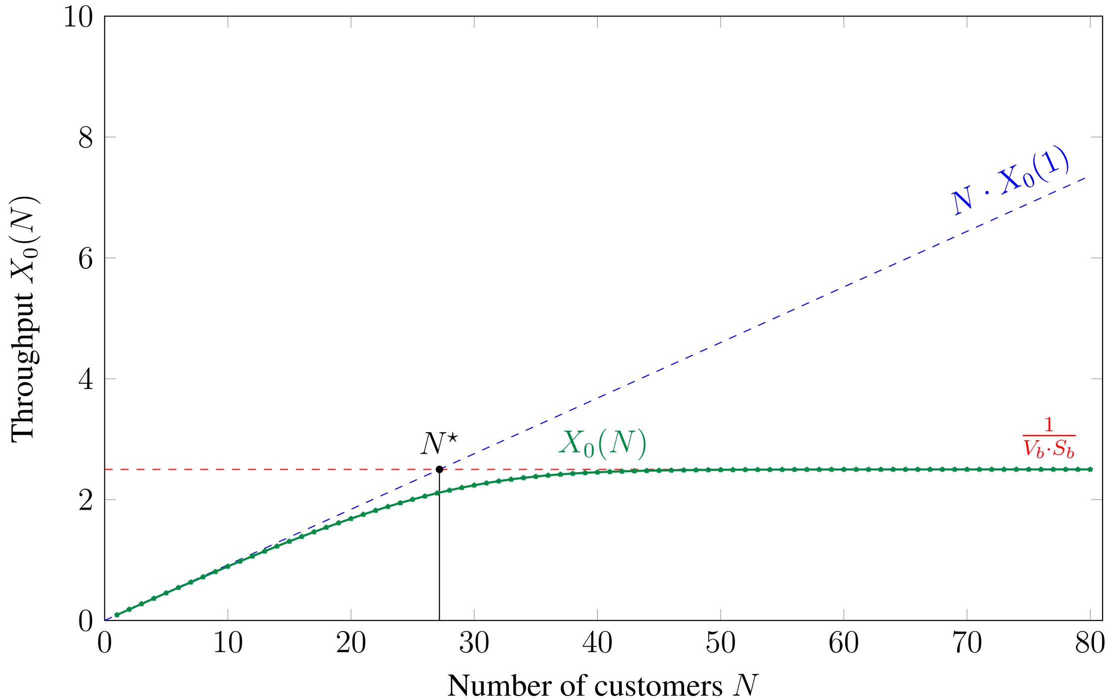

Figure 1: Evolution of throughput $X_0(N)$ as N increases. The point of saturation is $N^\star = 27.175000$. We can see how the throughput reaches a plateau after the saturation point.

standalone [x11names,table]xcolor tikz pgfplots

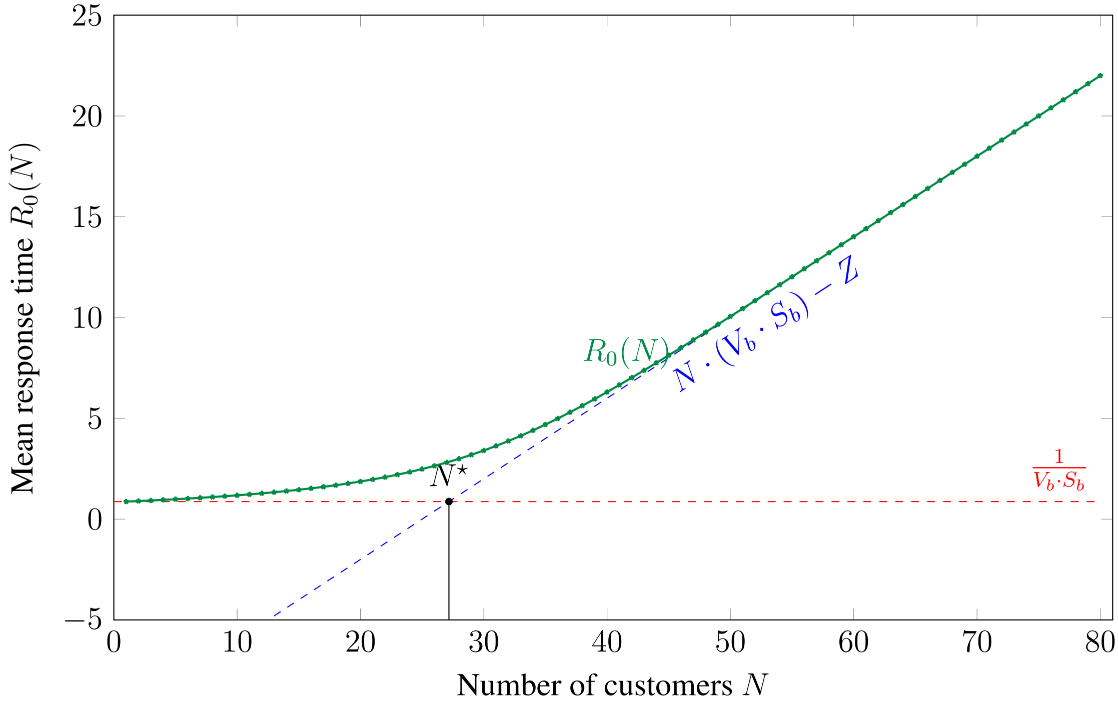

Figure 2: Evolution of mean response time $R_0(N)$ as N increases. The point of saturation is $N^\star = 27.175000$. On the contrary here the response time explodes after the saturation point

standalone [x11names,table]xcolor tikz pgfplots

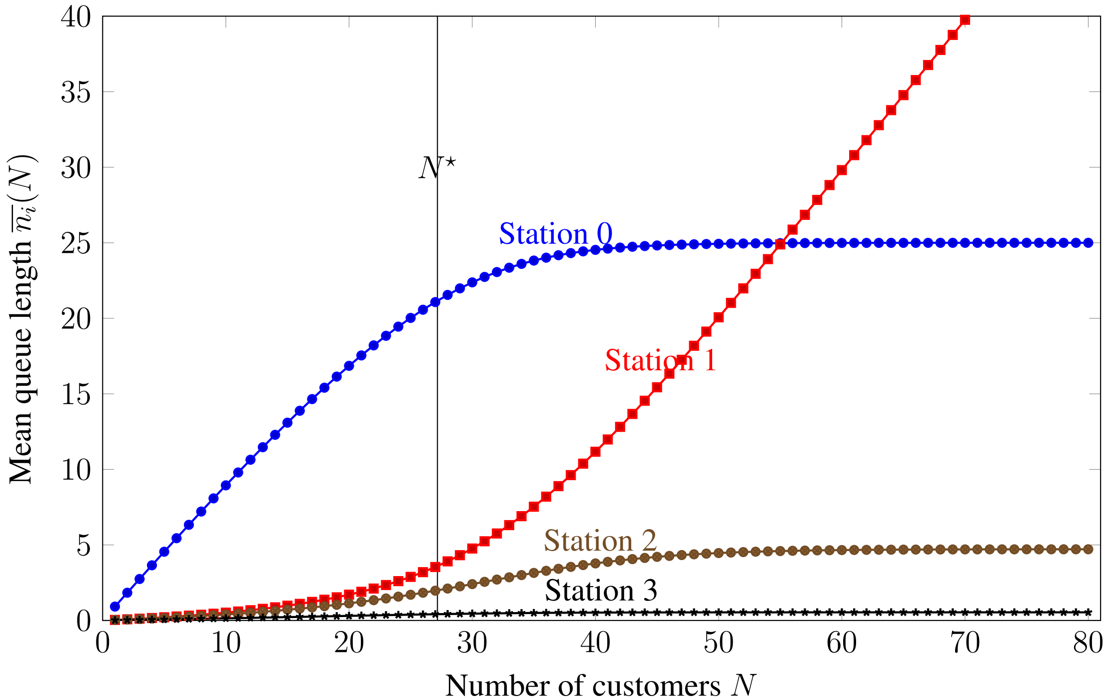

Figure 3: Behaviour of the average queue length of the four stations.

standalone [x11names,table]xcolor tikz pgfplots

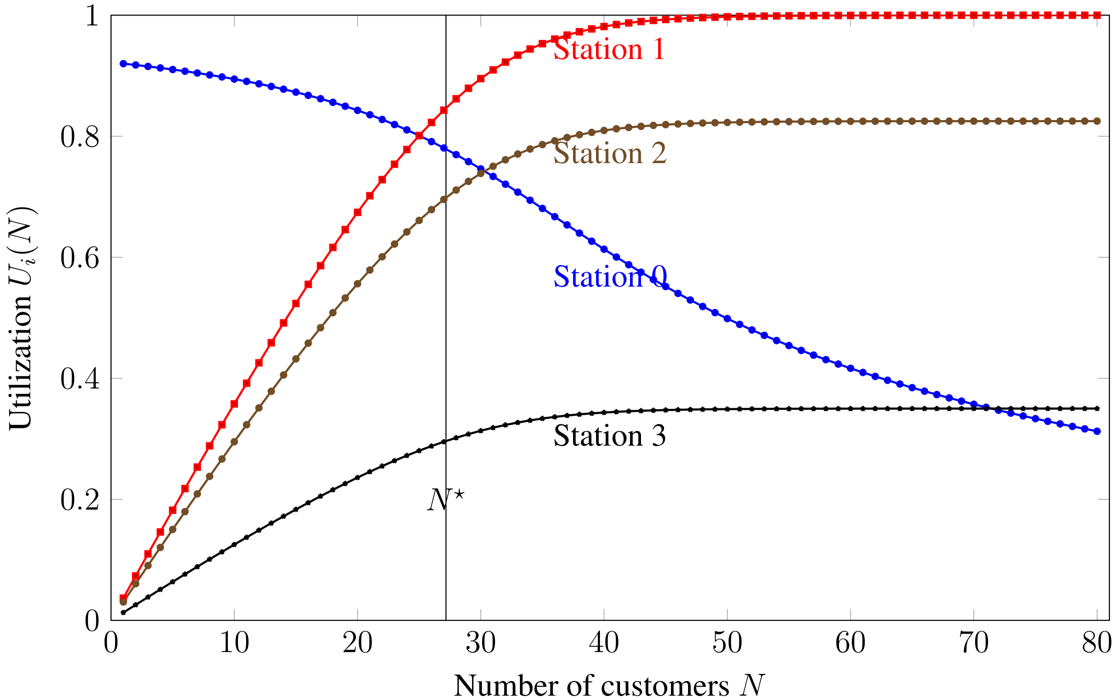

Figure 4: Behaviour of the utilization of the four stations.

Station 1 (the CPU) is the bottleneck station: we can see how, in comparison to the other stations, the queue becomes infinitely large in average while the other queues converge to a plateau.
As expected, in the fourth graph the bottleneck station reaches utilization 1 after the saturation point. It is interesting to note that the utilization of the delay station keeps getting smaller even after the bottleneck stations is saturated and the others have converged to a steady value. This is not a surprise, since in our bottleneck analysis we got that the utilization should have been equal to 0 as $N \to \infty$.