M/M/1 single server queueing model

135761B 大城海斗

2015 年 11 月 26 日

1 M/M/1 モデルの 3 つの評価ポイント

M/M/1 モデルにおいて重要な評価ポイントは，(1) 平均待ち時間，(2) 平均待ち人数，(3) システムの稼働時間の割合である．シミュレーションにおいて，これらは複数回に渡って検証されるべきだと考えた．

私が今回行った実験では，プログラムを 100 回実行し，その結果を用いて度数分布 Frequency distribution を作成した．なお，客の人数は 1000 人に固定した上で，

1. 客の来る間隔 [minute] の上限を 1.000，客一人当たりの作業時間 [minute] の上限を 0.5000 としたもの (section2)
2. 客の来る間隔の上限を 1.000 とし，客一人当たりの作業時間の上限を 0.2500，1.0000 としたもの (section3)
3. 客の来る間隔の上限を 0.500，2.000 とし，客一人当たりの作業時間の上限を 0.5000 としたもの (section4)

の入力パラメータの組み合わせで出力された結果を評価する．

2 客の来る間隔の上限を 1 分，客一人当たりの作業時間上限を 0.5 分とした場合

2.1 Average delay in queue(平均待ち時間)

平均待ち時間の度数分布のヒストグラムは図 1 のようになる．横軸が平均待ち時間，縦軸はプログラムの結果がその平均待ち時間となった回数を示している．プログラムを 100 回実行した結果，階級値 4.65×10^{-1}つまり 0.465 分 が客の平均の待ち時間となる場合が多かった．このことから，利用者は 30 秒も待つことなくサービスを受けられることが読み取れる．

ここで，階級数は 8 個となっているが，これは観測値の個数（プログラムの実行回数）100 に対してスタージェスの公式 Sturges' formula を用いて次のように求めた．

$$1 + log_2 100 = 7.643856189774725 \approx 8$$

これが階級の個数を決めるひとつの指標となるので，以下のヒストグラムでは階級数を 8 としてグラフを表現することとする．

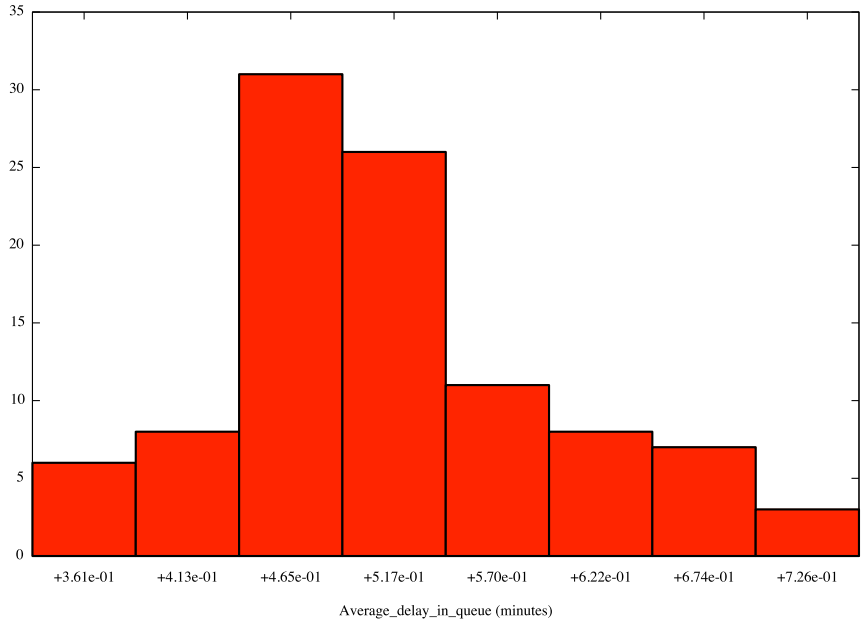

図 1　Frequency distribution of average delay in queue

2.2　Average number of customers in the queue(**平均待ち人数**)

平均待ち人数のヒストグラムは次の様になった（図 2）．グラフから，平均待ち人数は 0.468 人前後になることが多いと分かり，長い列ができるような状態でないことだと読み取れる．

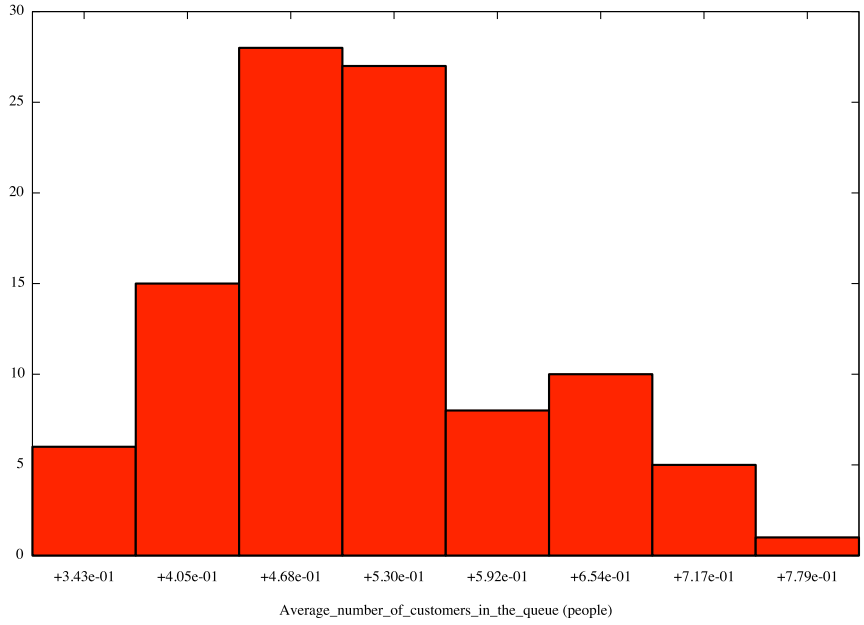

図 2　Frequency distribution of average number of customers in the queue

2.3 Efficiency of utilization of the server(**システムの稼働時間の割合**)

続いて，システムの稼働時間の割合である．これは銀行であれば，受付の人 (server) の仕事をしている時間を表すものである．システムの稼働時間の割合のヒストグラムは図 3 のようになった．1000 人の客を相手にしても受付の稼働率は 50% 程度である．受付は非常に忙しいということではないことが判断できる．

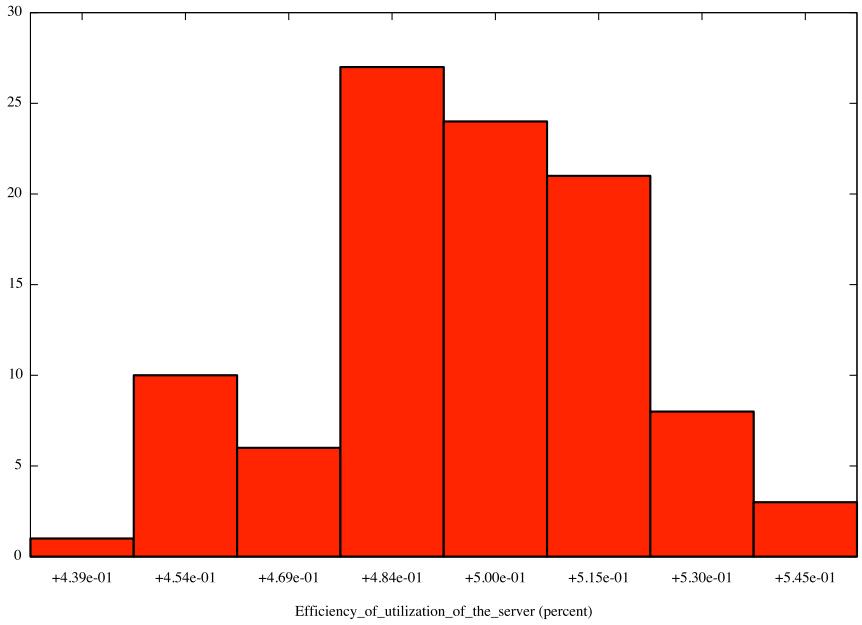

図 3　Frequency distribution of efficiency of utilization of the server

3　客の来る間隔の上限を 1 分，客一人当たりの作業時間の上限を変更した場合

3.1 Average delay in queue(**平均待ち時間**)

客一人当たりに対する作業時間の上限を 0.25 分とした場合のヒストグラムを図 4 ，1.00 分とした場合のヒストグラムを図 5 に示す．一人当たりに対する作業時間を短くすると，section2 の図 1 と比較して，多少のばらつきはあるものの，平均待ち時間が 8.81×10^{-2} 分に短縮されていることが分かる．反対に，一人当たりに対する作業時間を長くすると，客の平均待ち時間が 1.53×10^{1} まで増えていることが分かる．つまり，一人一人の客を素早く処理すれば平均待ち時間が減るということになる．

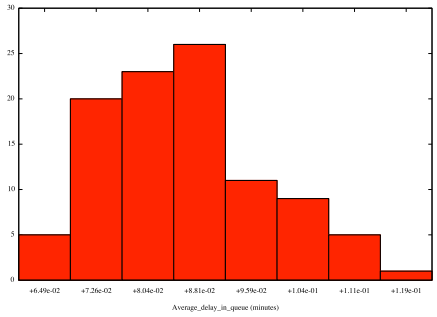

図 4　Frequency distribution of average delay in queue (0.2500)

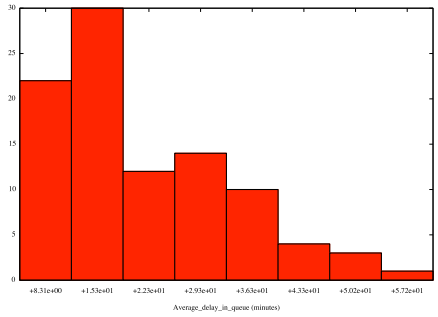

図 5　Frequency distribution of average delay in queue (1.0000)

3.2　Average number of customers in the queue(**平均待ち人数**)

平均待ち人数においても，ヒストグラムの「山」にあたる部分がより小さい値を取っているほうが良いシステムだと言える．客一人当たりに対する作業時間の上限を 0.25 分とした場合のヒストグラムを図 6 ，1.00 分とした場合のヒストグラムを図 7 に示す．客一人当たりに対する作業時間の上限を小さくした場合，平均待ち人数が 7.77×10^{-2}や 8.56×10^{-2} 人になる場合が多く，客はあまり並んでいないことが分かる．一方で，作業時間が増えると 14.8 人が平均して並んでいる状況が増えてくることが分かる．つまり，受付での作業時間を短縮することで待ち行列の人数を減らすことが期待できる．

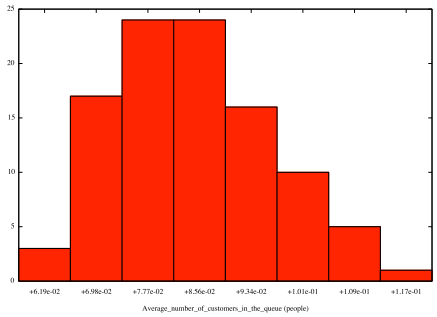

図 6　Frequency distribution of Average number of customers in the queue (0.2500)

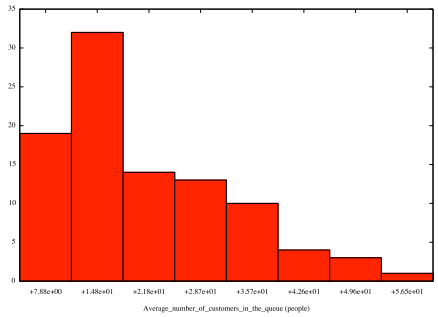

図 7　Frequency distribution of Average number of customers in the queue (1.0000)

3.3　Efficiency of utilization of the server(**システムの稼働時間の割合**)

同様にシステムの稼働時間の割合をプロットした結果が図 8 と図 9 となる．客一人当たりにかかる時間の上限を下げると，システム稼働率は 24% 程度になることが多く，反対に，上限を引き上げると，稼働率は 95% に達する場合が多かった．作業時間を減らすと受付の人 (server) は暇になることが分かる．

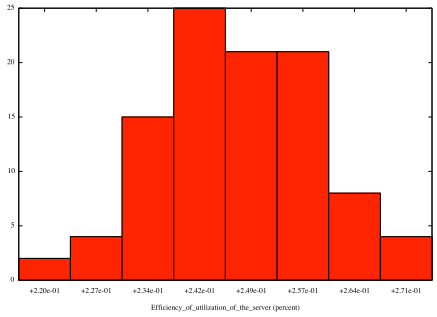

図 8　Frequency distribution of Efficiency of utilization of the server (0.2500)

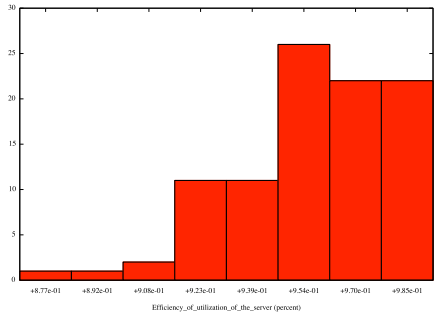

図 9　Frequency distribution of Efficiency of utilization of the server (1.0000)

4　客の来る間隔の上限を変更し，客一人当たりの作業時間上限を 0.5 分とした場合

4.1　Average delay in queue(平均待ち時間)

客の来店する間隔を変更した場合の平均待ち時間を見てみる．入力パラメータである，客の来る間隔の上限を引き下げると，その分頻繁に客が来る確率が高くなる．反対に上限を引き上げると，客はたまにしか来ないというシチュエーションを作れる．

客の来る間隔の上限を 0.5000 分とした場合に得られたヒストグラムは図 10 であり，上限を 2.000 分と設定した場合には，図 11 が得られた．図 10 より，客の来る間隔の上限を狭めると 7.65 分程度の待ち時間となる場合が多くなる結果となった．一方で，来客の間隔の上限を広げると，平均待ち時間が 0.176 分となる場合が多くなった（図 11）．このことから，頻繁に来客するようなシステムでは待ち時間が大きくなることが分かる．

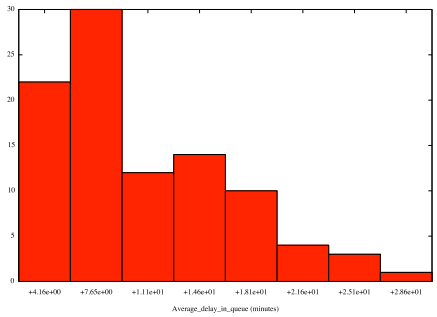

図 10　Frequency distribution of average delay in queue (0.5000)

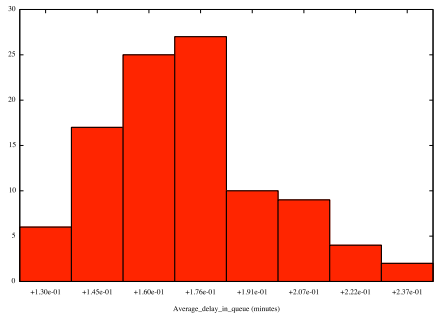

図 11　Frequency distribution of average delay in queue (2.0000)

4.2　Average number of customers in the queue(平均待ち人数)

次に，平均待ち人数についても見てみる．図 12 より，やはり待ち人数は 14.8 人と多くなっている傾向が強い．一方で，来客の間隔の上限を引き下げると，7.77×10^{-2}や 8.56×10^{-2} 人と少なくなった（図 13）．このことから，来客の間隔の上限を引き上げると列に並んでいる人数は少なくなる傾向が読み取れる．

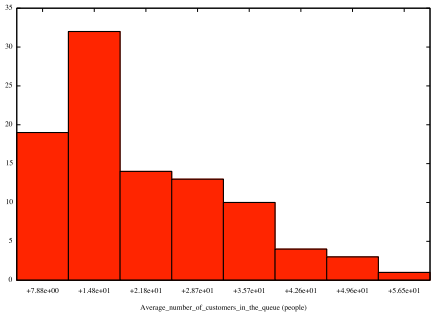

図 12　Frequency distribution of Average number of customers in the queue (0.5000)

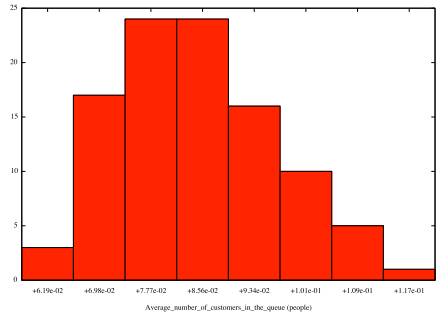

図 13　Frequency distribution of Average number of customers in the queue (2.0000)

4.3　Efficiency of utilization of the server(**システムの稼働時間の割合**)

システムの稼働時間の割合についても同様に見てみる．図 14 が客の来る間隔の上限を 0.5000 分にした時のヒストグラムである．稼働率は 95% 前後を示すことが多くなる結果となった．一方で，図 15 は客の来る間隔の上限を 2.0000 分に設定した時のヒストグラムである．システムの稼働率は 24% となっており，受付 (server) は暇な時間が多いと分かる．

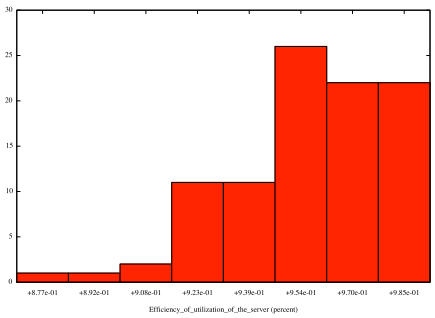

図 14　Frequency distribution of Efficiency of utilization of the server (0.5000)

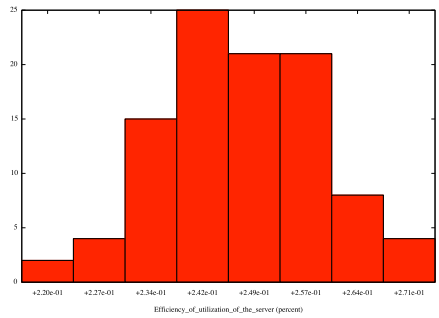

図 15　Frequency distribution of Efficiency of utilization of the server (2.0000)

5　まとめ

section2 から section4 に渡ってプログラムを実行させたときに得られる M/M/1 モデルの 3 つの評価点をそれぞれヒストグラムを用いて示した．

平均待ち人数とシステム稼働時間の割合に関して言えば，客の人数を $n = 1000$ と固定した場合，客の来る間隔の上限のデフォルトの値を 1/2 倍して得られたヒストグラムと，客一人当たりの作業時間の上限のデフォルトの値を 2 倍して得られたヒストグラムは同じものであることが分かった．例えば，図 6 と 図 13 のヒストグラムは同じであり，図 8 と図 15 も同じヒストグラムである．客の来る間隔の上限のデフォルトの値を 2 倍して得られたヒストグラムと，客一人当たりの作業時間の上限のデフォルトの値を 1/2 倍して得られたヒストグラムも同じ形になっている．つまり客の人数が同じであれば，客の来る間隔の上限を n 倍した場合と客一人当たりの作業時間の上限が $1/n$ 倍に設定した場合に得られる，平均待ち人数とシステムの稼働時間の割合は等価であると考えられる．

6　使用したプログラム

今回，シミュレーションに使用したプログラムは講義ページに公開されていた先輩のプログラム [1] を用いた（ソースコード 1）．乱数生成にあらかじめ定めたシード値をコマンドライン引数として渡すように改良を加えた（l.26 27, 88, 135 137）．ソースコード 1 を 100 回実行し，M/M/1 モデルの 3 つの評価ポイントを mm1.out から抽出するシェルスクリプトをソースコード 2 に示す．ソースコード 2 で抽出したデータを度数分布のヒストグラムのデータに加工するプログラムはソースコード 3 となる．

ソースコード 1　mm1.c

```
1  #include "mm1.h"
2
3  /*main（メイン関数）
4
5  事前準備
6  touch mm1.in mm1.out
7
8  入力ファイル (mm1.in)の書式
9  --
10 [(float)mean interval limit（客の来る間隔 [minute]の上限）] [(float)mean service
       limit（客一人当たりの作業時間[minute]の上限）] [(int)deleyed required（客の人数[people]）]
11 --
12
13 入力ファイル (mm1.in)の記入例
14 --
15 1.000 0.5000 1000
16 --
17 */
18
19 //false->Customer Number Limit Mode
20 //true->Fixed Time Mode
21 bool limit_time_mode=false;
22
23
24 int main(int argc, char **argv){
25
26     int s;
27     sscanf(argv[1], "%d", &s);
28
29     char *infilename = "mm1.in";
30     char *outfilename = "mm1.out";
31
32     //Read Exec options（実行オプションの読み込み）
33     int option;
34         while( (option = getopt(argc, argv,"i:o:th"))!=-1 ){
35                 switch(option){
36                         case 'i':
37                                 infilename = optarg;
38                                 break;
39             case 'o':
40                                 outfilename = optarg;
41                                 break;
42             case 't':
43                                 limit_time_mode = true;
44                                 break;
45                         case 'h':
```

```
                    default:
                            usage(argv[0]);
                            return 0;
                            break;
            }
        }

    /*Open input and output file (入力ファイルと出力ファイルを開く)*/
    infile = fopen(infilename,"r");
    outfile = fopen(outfilename,"w");

    /*Specify the number of events for the timing function (イベント数を定義)*/
    //Customer Number Limit Mode
    if(!limit_time_mode) num_events = 2;//到着 (Arrival Event)と出発 (Departure Event)
    //Time Limit Mode
    else num_events = 3;//到着
        (Arrival Event)と出発 (Departure Event)と終了 (Simulation End Event)

    /*Read input parameter.(パラメータの読み込み)*/
    if(!limit_time_mode) {//Customer Number Limit Mode
        fscanf(infile,"%f␣%f␣%d",&mean_interarrival,&mean_service,&num_deleyed_required);
        printf("input␣%f␣%f␣%d\n",mean_interarrival,mean_service,num_deleyed_required);
    }else{//Time Limit Mode
        fscanf(infile,"%f␣%f␣%f",&mean_interarrival,&mean_service,&time_end);
        printf("input␣%f␣%f␣%f\n",mean_interarrival,mean_service,time_end);
    }

    if(!limit_time_mode) printf("Exec␣Limit␣Costomer␣Number␣Mode\n");
    else printf("Exec␣Limit␣Time␣Mode\n");

    /*Write Report heading and input parameter.(入力されたパラメータを書き出し)*/
    fprintf(outfile,"Single-server␣queueing␣system␣\n");
    fprintf(outfile,"Mean␣interarrival␣time␣Limit␣%11.3f␣minutes␣\n",mean_interarrival);
    fprintf(outfile,"Mean␣service␣time␣Limit␣%16.3f␣minutes␣\n",mean_service);

    if(!limit_time_mode) fprintf(outfile,"Number␣of␣customers␣%14d␣\n",num_deleyed_required
        );
    else fprintf(outfile,"Length␣of␣the␣Simulation␣%9.3f␣minute␣\n",time_end);


    printf("mean␣interarrival␣%f␣\n",mean_interarrival);
    printf("mean␣service␣%f␣\n",mean_service);

    /*Initialize the simulation.(シミュレーションの初期化)*/
    initialize(s);

    bool sim_run=true;
    //int arrival_id=0,dep_id=0;
    while(sim_run){
        /*Determine the next event.(次のイベントを定義)*/
        timing();

        /*Update time-average statistical accumulators (のべ時間の更新)*/
        update_time_avg_status();

        /*Invoke the appropriate event function.(イベントの実行)*/
        switch(next_event_type){
            case 1:
                arrival_id++;
```

```
103                 printf("[%10.3f] Customer %d is Arrival\n",sim_clock,arrival_id);
104                 arrival();//Arrival Event(到着イベント)
105                 break;
106             case 2:
107                 dep_id++;
108                 printf("[%10.3f] Customer %d is Departure\n",sim_clock,dep_id);
109                 depart();//Departure Event(出発イベント)
110                 break;
111             case 3://for Limit Time Mode
112                 /*Invoke the report generator and end the simulation.(レポートを出力してシミュ
                        レーションを修了)*/
113                 report();
114                 break;
115         }
116         if(!limit_time_mode){//Limit Customer Number Mode
117             /*Quit the simulation when more delays are not needed.(もう客が来なくてもいいならシ
                    ミュレーションを終わる)*/
118             if(num_custs_delayed > num_deleyed_required) sim_run=false;
119         }else{//Limit Time Mode
120             /*Quit the simulation when time to simulation end.(シミュレーションの終了時間になっ
                    たらシミュレーションを終わる)*/
121             if(next_event_type==3) sim_run=false;
122         }
123     }
124
125     /*Invoke the report generator and end the simulation.(レポートを出力してシミュレーションを
            修了)*/
126     if(!limit_time_mode) report();//Limit Customer Number Mode
127
128     fclose(infile);
129     fclose(outfile);
130
131     return 0;
132 }
133
134 /*Initialization function (初期化)*/
135 void initialize(seed){
136     /*Set rondom seed*/
137     srand(seed);
138
139     /*Initialize the Simulation sim_clock (シミュレーション内の時計の初期化)*/
140     sim_clock = 0.0f;
141
142     /*Initialize the state variables (状態の初期化)*/
143     server_status = IDLE;
144     num_in_q = 0;
145     time_last_event = 0.0;
146
147     /*Initialize the statical counters (カウンタの初期化)*/
148     num_custs_delayed = 0;
149     total_of_delays = 0.0;
150     area_num_in_q = 0.0;
151     area_server_status = 0.0;
152
153     /*Initialize event list.(イベントリストの初期化)
154         Since no customers are present,(客が存在しないなら)
155         the departure (service completion) event is eliminated from consideration.
156         (「departure」イベントは無視する)*/
157     time_next_event[1] = sim_clock + expon(mean_interarrival);
```

```
158     /*Guaranteening that first event will be an arrival
159     (最初のイベントをArrival にするために、([timing()]
            min_time_next_event より) 大きな数を設定しておく)*/
160     time_next_event[2] = 1.0e+30;
161     /*if Limit Time Mode, set Time to Simulation end*/
162     if(limit_time_mode) time_next_event[3] = time_end;
163 }
164
165
166 /*Timing function*/
167 void timing(void){
168     int i;
169     float min_time_next_event = 1.0e+29;
170
171     next_event_type = 0;
172
173     /*Determine the event type of next event occur (次に発生するイベント生成する)*/
174     for(i=1; i <= num_events; ++i){
175         if(time_next_event[i] < min_time_next_event){
176             min_time_next_event = time_next_event[i];
177             next_event_type = i;
178         }
179     }
180
181     /*Check to see whether the event list is empty (イベントリストが空でないかチェックする)*/
182     if(next_event_type == 0){
183         /*the event list is empty, so stop
                simulation (イベントリストが空なのでシミュレーション停止)*/
184         fprintf(outfile,"\nEvent␣list␣empty␣at␣sim_clock␣%f",sim_clock);
185         exit(1);
186     }
187
188     /*The event list is not empty, so advance the simulation
            sim_clock (イベントリストが空でないので、時計を進める)*/
189     sim_clock = min_time_next_event;
190 }
191
192 /*Arrival event function (到着イベント)*/
193 void arrival(void){
194     float delay;
195
196     /*Schedule next arrival (次の到着をスケジュールに追加)*/
197     time_next_event[1] = sim_clock + expon(mean_interarrival);
198
199     /*Check the whether server is busy (店員が忙しいかどうかチェック)*/
200     if(server_status == BUSY){
201         /*server is busy, so increment number of customers in
                queue (店員が忙しいので、待ち行列内の客の数を増やす)*/
202         ++num_in_q;
203
204         /*Check to see whether an overflow condition status (客があふれていないかチェック)*/
205         if(num_in_q > Q_LIMIT){
206             /*the queue has overflowed, so stop the
                    simulation (客があふれているのでシミュレーション停止)*/
207             fprintf(outfile,"\nOverflow␣of␣the␣array␣time_arrival␣at␣sim_clock␣%f",
                    sim_clock);
208             exit(2);
209         }
210
```

```
211         /*there is still room in the queue,(まだ待ち行列内にいるので)
212             so store the sim_clock of arrival of the arriving costomer at the (new) end of
                    time_arrival
213             (time_arrival に客の到着時間を追加する)*/
214         time_arrival[num_in_q] = sim_clock;
215
216     }else{
217         /*server is idle,(店員はヒマ)
218             so arriving customer has delay of zero.(なので待ち時間はなし)
219
220             the following two statement are for program clarity and do not affect the
                    result of the simulation
221             (一応明記しておくが、シミュレーションの結果には影響がない)*/
222         delay = 0.0;
223         total_of_delays += delay;
224
225         /*Increment the number of customers delayed,(待っていた客の数を増やし)
226             and make server busy (店員を仕事中にする)*/
227         ++num_custs_delayed;
228         server_status = BUSY;
229
230         /*Schedule a dedeparture(service completion) (出発イベントをスケジュールに追加)*/
231         time_next_event[2] = sim_clock + expon(mean_service);
232     }
233 }
234
235 /*Departure event function (出発イベント)*/
236 void depart(void){
237     int i;
238     float delay;
239
240     /*Check to see whether the queue is empty (客が待っているか調べる)*/
241     if(num_in_q == 0){
242         /*queue is empty so make server idle and eliminate the departure event from
                consideration
243         (待ってる人がいないので店員をヒマにし、出発イベントを無視する)*/
244         server_status = IDLE;
245         //min_time_next_event より確実に大きいので timing で発生されなくなる
246         time_next_event[2]= 1.0e+30;
247     }else{
248         /*queue is nonempty,(だれかが待っている)
249             so decrement the number of customers in queue.(行列から一人減らす)*/
250         --num_in_q;
251
252         /*Compute the delay of customer who is beginning service and update the total delay
                accumulator
253         (その人が待っていた時間を計算し、総待ち時間を増やす)*/
254         delay = sim_clock - time_arrival[1];
255         total_of_delays += delay;
256
257         /*Increment the number of costomers delayed,(待っていた客の数を増やし)
258             and schedule departure.(出発イベントをスケジュールに追加する)*/
259         ++num_custs_delayed;
260         time_next_event[2] = sim_clock + expon(mean_service);
261
262         /*Move each customer in queue (if any) up one place.(待ち行列から客を削除し前につめる)
                */
263         for(i=1; i <= num_in_q; ++i) time_arrival[i] = time_arrival[i+1];
264     }
```

```
265 }
266
267 /*Report generator function (結果表示)*/
268 void report(void){
269     /*Compute and write estimates of desired mesures of perfomance.
270     ()*/
271     fprintf(outfile,"\nAvarage␣delay␣in␣queue␣%11.3f␣minutes",total_of_delays/
            num_custs_delayed);//平均待ち時
            間
272     fprintf(outfile,"\nAvarage␣number␣of␣customers␣in␣queue␣%10.3f",area_num_in_q/sim_clock
            );//平均待ち人
            数
273     fprintf(outfile,"\nEfficiency␣of␣utilization␣of␣the␣server␣%15.3f",area_server_status/
            sim_clock);//店員の仕事
            率
274     fprintf(outfile,"\ntime␣simulation␣ended␣%12.3f",sim_clock);//シミュレーション時間
275     fprintf(outfile,"\nVisited␣%d␣customers",arrival_id);//来店客数
276     fprintf(outfile,"\nServiced␣%d␣customers",dep_id);//来店客数
277 }
278
279 /*Update area acumulator for time-average (のべ時間の更新)*/
280 void update_time_avg_status(void){
281     float sim_clock_since_last_event;
282
283     /*Compute sim_clock since last event,(最終イベントからの経過時間を計算し)
284         and update last-event-sim_clock marker.(最終イベント時間を更新)*/
285     sim_clock_since_last_event = sim_clock - time_last_event;
286     time_last_event = sim_clock;
287
288     /*Update area under number-in-queue function.(のべ待ち時間更新)*/
289     area_num_in_q += num_in_q * sim_clock_since_last_event;
290
291     /*Update area under server-busy indicator function.(のべ作業時間の更新)*/
292     area_server_status += server_status * sim_clock_since_last_event;
293 }
294
295
296 /*Exponential variate genaration function (指数の変量を計算)*/
297 float expon(float mean){
298     /*Exponential function
299         f(x) = (1/mean)e^(-x/mean) : (for x>=0)
300             = 1-e^(-x/mean)
301             = -mean*ln(U)
302     */
303     float u;
304
305     /*Generate a U(0,1) random variate (0<=u<=1の乱数)*/
306     //u = rand(1);//rand.h
307     u = (float)rand()/RAND_MAX;
308
309     /*Return an exponential random variate with mean "mean"
310     (mean を基に計算した Exponential Variate を返す)*/
311     return -mean * log(u);
312
313 }
314
315 void usage(char *myname){
316     printf("%s␣[hdt][-i␣Input_File_Name][-o␣Output_File_Name]\n",myname);
317     printf("\t␣-t:Limit␣Time␣Mode␣(default:Limit␣Costomer␣Number␣Mode)\n");
```

```
318     printf("\t␣-i␣<Input_File_Name>:set␣Input␣File␣(default:mm1.in)\n");
319     printf("\t␣-o␣<output_File_Name>:set␣Output␣File␣(default:mm1.out)\n");
320     printf("\t␣-h:show␣this␣usage\n");
321     printf("\n");
322     printf("Input␣File's␣format␣(Limit␣Costomer␣Number␣Mode)\n---␣\n");
323     printf("[(float)mean␣interva␣llimit[minute]]␣[(float)mean␣service␣limit[minute]]␣[(int)
            deleyed␣required[people]]\n");
324     printf("ex)1.000␣0.5000␣1000\n");
325     printf("\n");
326     printf("Input␣File's␣format␣(Limit␣Time␣Mode)\n---␣\n");
327     printf("[(float)mean␣interva␣llimit[minute]]␣[(float)mean␣service␣limit[minute]]␣[(
            float)End␣time[minute]]\n");
328     printf("ex)1.000␣0.5000␣100\n");
329 }
```

ソースコード 2　extracter.sh

```
1  #!/bin/zsh
2  # the number of executions
3  n=100
4
5  #extract 3 evaluations from 'mm1.out'.
6  function extract () {
7      avarage_delay=‘awk '/Avarage␣delay␣in␣queue/␣{print␣$5}' mm1.out‘
8      avarage_number_of_customers_in_queue=‘awk '/Avarage␣number␣of␣customers␣in␣queue/␣{
           print␣$7}' mm1.out‘
9      efficiency_of_utilization_of_the_server=‘awk '/Efficiency␣of␣utilization␣of␣the␣server/
           ␣{print␣$7}' mm1.out‘
10 }
11
12 # make a data file.
13 function make_data_file () {
14     echo "$avarage_delay,$avarage_number_of_customers_in_queue,
           $efficiency_of_utilization_of_the_server" >> evaluation.dat
15 }
16
17 # remove evaluation.dat when it exists.
18 if [ -e evaluation.dat ]; then
19     rm evaluation.dat
20     echo "removed␣evaluation.dat"
21 fi
22
23 for i in ‘seq 1 $n‘
24 do
25     # executing the C program -> extracting 3 evaluations -> making a data file.
26     ./mm1 $i && extract && make_data_file
27 done
```

ソースコード 3　fd.py

```
1  # this program receives an integer parameter and outputs frequency distribution's␣data.
2  # Usage:
3  # % python -V
4  # Python 3.4.2
5  # % python fd.py 0 # outputs frequency distribution data of the average delay in queue
6  # % python fd.py 1 # outputs frequency distribution data of the average number of
       customers in the queue
```

```
# % python fd.py 2 # outputs frequency distribution data of the efficiency of utilization
    of the server
import numpy as np
import math
import sys
import csv

parameters = sys.argv
#parameters[1] = colomn

# open the data file that includes 3 evaluations.
f = open('evaluation.dat', 'r')
reader = csv.reader(f)
data = []
for row in reader:
    data.append(row[int(parameters[1])])
f.close()

values = []
for k in range(0,len(data)):
    values.append(float(data[k]))

# determine the number of class by Sturges' formula.
frequency_number = math.ceil(1+math.log(100,2))
# determine class.
maximum = max(values)
minimum = min(values)
interval = np.linspace(minimum, maximum, frequency_number+1)

# determine each of the class's frequency.
frequency = [0]*(len(interval)-1)
for i in range(0,len(interval)-1):
    for v in values:
        if interval[i] <= v and v < interval[i+1]:
            frequency[i] += 1
for v in values:
    if interval[-1] == v:
        frequency[-1] += 1

# output data.
for i in range(0,len(interval)-1):
    print("{0:+9.2e} {1}".format((interval[i]+interval[i+1])/2,frequency[i]))
```

参考文献

[1] mm1.zip
https://ie.u-ryukyu.ac.jp/~asharif/pukiwiki/index.php?%A5%B7%A5%DF%A5%E5%A5%EC%A1%BC%A5%B7%A5%E7%A5%F3

[2]「統計学入門 (基礎統計学)」 東京大学教養学部統計学教室 東京大学出版会