システム解析特論レポート

愛媛大学大学院理工学研究科電子情報工学専攻

ICTスペシャリスト育成コース

中山　颯

課題I

より，

　ここで，到着間隔tに到着する人数は１人であり，ある時間tまでに待ち行列に１人が到着する確率は，全体の事象が起こる確率１から，来ない時の確率である（１）式を引けばよい．

　よって，

課題Ⅱ

　本課題でのM/M/c待ち行列システムを考える際、n= 0のとき、0< n < cのとき、c≤0のときの3つの状態で状態方程式を考える必要がある。

* n=0の時

Δt→０とすると，

t→∞とすると

* 0< n < cの時

Δt→0とすると

t→∞とすると

* のとき

Δt→0とすると

t→∞とすると

ここで、n = 1 の場合から順に書き下してみる．

これらの式をすべて足し合わせると打ち消し合いが起こるので、結果的に以下の式を得ることが

できる。

ここで、待ち行列の取りうる状態確率すべての和は１となることを踏まえる。

すなわち，状態確率は下記のように定まる．

課題III

　イベント駆動型M/M/1シミュレーションの実装を行う．実装したコードを付録1に示し，平均到着率を1.0，平均サービス時間を2.0とした場合の結果を表１に示す．

表1．課題３の実行結果

|  |  |  |
| --- | --- | --- |
|  | システム内客数 | システム内時間 |
| 理論値 | 1 | 1 |
| 計測値 | 1.007105 | 1.008026 |

課題Ⅴ

　課題Ⅲで実装したシミュレータの正当性を確認するために，シミュレーションによって得られた結果の95%信頼区間内に理論値が含まれているかどうか確認する．平均到着率を1.0，平均サービス率を2.0とし，シミュレーションで得られた結果は以下の表1の通りである．

1.システム内客数について，

　シミュレータによって得られた結果は以下の通りである

　T分布の95%信頼区間は次の通りである．

理論値μ＝1が95%信頼区間に含まれていることから，システム内客数のシミュレーションの結果の正当性が証明された．

2.システム内時間について

　システム内客数と同様に，

とし，

理論値μ＝1が95%信頼区間に含まれていることから，システム内時間のシミュレーションの結果の正当性が証明された．

　以上より，

課題VI

　平均サービス率（μ）を1.0として，平均到着率（λ）を0.1から１まで，0.1刻みで変化させ，各パラメータセットに対する上記統計量をグラフとして作成した．以下の表１，図１，図２にその結果を示す． また，理論値の計算式は以下の通りである．

　表２の結果より，課題Vで確認した通り，理論値と計測値の値がほとんど同じように推移していることがわかる．また，平均到着率が平均サービス率と同じ値になった時，理論値は発散し，計測値も急激に悪くなることがわかる．また，平均到着率（λ）を1.1にした場合，システム内客数は640.854172，システム内時間は583.283089と急激に悪化した．

よって，平均到着率が平均サービス率以上になった場合，急激に待ち行列が長くなり，サービス提供時間が長くなることが推測される．

表２．平均到着率に対するシステム内客数とシステム内時間（計測値，理論値）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 平均到着率 | システム内客数（計測値） | システム内時間（計測値） | システム内客数（理論値） | システム内時間（理論値） |
| 0.1 | 0.111048 | 1.124035 | 0.111111 | 1.111111 |
| 0.2 | 0.245079 | 1.262102 | 0.25 | 1.25 |
| 0.3 | 0.429234 | 1.436368 | 0.428571 | 1.428571 |
| 0.4 | 0.679748 | 1.673827 | 0.666667 | 1.666667 |
| 0.5 | 1.014021 | 2.013952 | 1 | 2 |
| 0.6 | 1.54112 | 2.546054 | 1.5 | 2.5 |
| 0.7 | 2.300676 | 3.285565 | 2.333333 | 3.333333 |
| 0.8 | 4.017867 | 5.022862 | 4 | 5 |
| 0.9 | 8.232347 | 9.181194 | 9 | 10 |
| 1 | 55.649606 | 56.001118 | - | - |
| 1.1 | 640.854172 | 583.283089 | - | - |

図１．平均到着率に対するシステム内客数の理論値と計測値の推移

図２．平均到着率に対するシステム時間の理論値と計測値の推移

２．課題B

　下記の3種類の待ち行列システムに関して，実装を行い，どのシステムか効率的であるかを考察する．

1. 3台のレジにそれぞれ別々の待ち行列を作るシステムA
2. 待ち行列は1つとして3台のレジのうち，空いたレジに待ち行列の先頭の客が移動してサービスを受けるシステムB（いわゆるフォーク並び）
3. 待ち行列とレジは1つとして1人で3人分の仕事をこなせる超弩級のスーパー店員がいるシステムC

　また，プログラムは平均到着率を1.0，平均サービス時間を2.0として実行する．また，それぞれ実装したプログラムソースを，それぞれ付録2,3,4に示す．

２.1 システムA

　システムAの場合，３つのレジと３つの待ち行列を作るため，平均サービス率は変わらないが，平均到着率が1/3になるシステムとなる．よって，平均利用率ρも1/3となる．論理値の計算式は以下のようになる．また，結果は，以下の表４の通りとなった．

表３．システムAの実行結果

|  |  |  |
| --- | --- | --- |
|  | システム内客数 | システム内時間 |
| 理論値 | 0.600000 | 1.800000 |
| 計測値 | 1.175727 | 1.857948 |

２.２ システムB

　システムBでは、論理値の計算式は以下の式４，５のようになる．また，結果を以下の表４に示す．

表４．システムBの実行結果

|  |  |  |
| --- | --- | --- |
|  | システム内客数 | システム内時間 |
| 理論値 | 1.400000 | 1.40000 |
| 計測値 | 0.504248 | 0.503388 |

２.３ システムC

　システムCはM/M/1モデルの平均サービス時間を3倍としたシステムである．論理値の計算式を式５，６，７に示す．また，結果を以下の表４に示す．

表５．システムBの実行結果

|  |  |  |
| --- | --- | --- |
|  | システム内客数 | システム内時間 |
| 理論値 | 0.200000 | 0.200000 |
| 計測値 | 0.197735 | 0.200348 |

２.４ 考察

　結果より，理論値では，システムC＞システムA＞システムBの順で，システム内の客数や待ち時間が少ないシステムとなり，実測値では，システムC＞システムB＞システムAの順で，システム内の客数や待ち時間が少ないシステムとなった．また，システムAとシステムBに関しては，理論値と計測値の結果に信頼区間95%以上の差があることが見られる．この結果から，理論値の計算式自体が正しくないか，作成したプログラムにミスがあることが推測される．

　また，一般的には，システムC＞システムB＞システムAの順で，待ち時間が短いシステムになることが知られている.作成したシステムは，同じ順で待ち時間が短くなっているが，作成したシミュレータが正しく実装できているかどうかは，論理値の正しい計算式を導き出す必要がある．

付録1

main.cpp

|  |
| --- |
| #include <stdio.h>  #include "cmdline.h"  #include "MM1.hpp"  #include <tuple>  using namespace std;  int main(int argc, char \*\*argv)  {      // 初期化      MM1 mm1;                       // MM1 クラスの宣言      cmdline::parser p;             // コマンドライン引数用の宣言      double rho;                    // 利用率      double system\_people = 0.0;    // システム内客数      double system\_time = 0.0;      // システム内a時間      tuple<double, double> results; // シミュレーション結果取得用変数      double start\_time = 1000.0;    // 過渡状態とみなすまでの時間      double end\_time = 11000.0;     // シミュレーション終了時間      int trials = 10;      /\* コマンドライン引数の定義      // −a 数値 とすると到着率　初期値は 1.0      // −s 数値 とするとサービス率を代入　初期値は 2.0 \*/      p.add<double>("arrival",'a', "arrival rate", false, 1.0);      p.add<double>("service",'s', "service rate", false, 2.0);      p.parse\_check(argc, argv);      double lambda = p.get<double>("arrival"); // 到着率      double mu = p.get<double>("service");     // サービス率      rho = (lambda / mu); // 到着率およびサービス率から利用率算出      FILE \*fp;      if ((fp = fopen("Results.txt", "w")) == NULL)      {          printf("file open error !!\n");          getchar();          exit(EXIT\_FAILURE);      }      // シミュレーション開始 (trials 回繰り返してその平均値を取得)      for (int i = 0; i < trials; i++)      {          results = mm1.simulation(lambda, mu, start\_time, end\_time);          system\_people = system\_people + get<0>(results); // システム内客数を取得          system\_time = system\_time + get<1>(results);     // システム内時間を取得      }      fprintf(fp, "\n");      system\_people = system\_people / trials;      system\_time = system\_time / trials;      // 理論値および観測値を保存      //ここ直すだけ      double logical\_ave\_customers = rho / (1 - rho);      double logical\_ave\_wait\_time = (1/mu) \*  (1 / (1 - rho));      fprintf(fp, "%s\t %s\t %s\n", "ρ", "システム内客数", "システム内時間");      fprintf(fp, "% lf\t % lf\t % lf\n", rho, logical\_ave\_customers, logical\_ave\_wait\_time);      fprintf(fp, "% lf\t % lf\t % lf\n", rho, system\_people, system\_time);      // printf("% lf\t % lf\t % lf\n", rho, system\_people, system\_time);      fclose(fp);  } |

MM1.cpp

|  |
| --- |
| #include "MM1.hpp"  std::tuple<double, double> MM1::simulation(double lambda, double mu, double startTime, double endTime)  {      // 初期化      vector<double> service;    // 窓口      queue<double> system;      // 待ち行列      Random random;             // ランダムクラスの宣言      SimStat simStat = StandBy; // 過渡状態、過渡状態以降を定義      double service\_interval;   // サービス間隔      double arrive;             // 到着間隔      currentTime = 0.0;      simEndJobs = 0;      stayTime = 0.0;      visitors = 0.0;      // 処理開始（最初のイベントをイベントキューに登録）      event.time = currentTime + random.expDistribution(lambda);      event.eventState = ARRIVE;      eventQueue.push(event);      // 終了時間まで繰り返す      while (currentTime < endTime)      {          // 最初のイベントを取り出す          event = eventQueue.top();          eventQueue.pop();          // 現在時刻をイベント発生時刻まですすめる          currentTime = event.time;          switch (event.eventState)          {          // 到着イベントの場合          case ARRIVE:              // 待ち行列への登録              system.push(event.time);              // イベントキューへのサービス到着イベントの登録              event.time = currentTime + random.expDistribution(lambda);              event.eventState = ARRIVE;              eventQueue.push(event);              break;          // サービス終了イベントの場合          case FINISH:              // 過渡状態中の統計値はリセットする              if (currentTime > startTime && simStat == StandBy)              {                  simStat = Execution;                  stayTime = 0;                  visitors = 0;                  simEndJobs = 0;              }              // 統計処理（待ち時間，客数等のカウント）              stayTime += currentTime - service[0];              visitors += system.size();              simEndJobs++;              // 窓口のクリア              service.clear();              break;          }          // 窓口があいていて、待ち行列に人がいる場合          // if (service.empty() && system.empty() == false)          if (service.empty() && system.empty() == false)          {              // 待ち行列の先頭から窓口への移動              service.push\_back(system.front());              system.pop();              // イベントキューへのサービス到着イベントの登録              event.time = currentTime + random.expDistribution(mu);              event.eventState = FINISH;              eventQueue.push(event);          }          // printf("%f \n", currentTime);      }      // 結果の返却      // return std::forward\_as\_tuple(システム内客数, システム内時間);      return std::forward\_as\_tuple(visitors / simEndJobs, stayTime / simEndJobs);  } |

MM1.hpp

|  |
| --- |
| #include <iostream>  #include <cmath>  #include <iomanip>  #include <queue>  #include <random>  #include <tuple>  #include <vector>  #include "Random.hpp"  // イベントの種類を定義  enum EventState  {      ARRIVE = 0,      FINISH = 1  };  // シミュレーション状態を定義  enum SimStat  {      StandBy = 0,      Execution = 1,      Finish = 2  };  // イベント型を定義  struct EVENT  {      // イベントの種類      EventState eventState;      // イベントの発生時間      double time;      // Priority queue 用のオーバロード関数      bool operator>(const EVENT &event2) const      {          return time > event2.time;      }  };  class MM1  {      private:          double currentTime;                                              // 現在時刻          EVENT event;                                                     // イベントを格納する変数          priority\_queue<EVENT, vector<EVENT>, greater<EVENT>> eventQueue; // イベントキュー          int simEndJobs;                                                  // シミュレーション終了までに発生したジョブ数          double stayTime;                                                 // 待ち時間          double visitors;                                                 // 客数      public:          std::tuple<double, double> simulation(double rambda, double mu, double startTime, double endTime);  }; |

Random.cpp

|  |
| --- |
| #include "Random.hpp"  double Random::expDistribution(double variable)  {     double expDist = 0.0;                             // 指数乱数用の宣言     mt19937 mt(rnd());                                // メルセンヌ・ツイスター用の宣言     uniform\_real\_distribution<> uni\_random(0.0, 1.0); // 0−1 までの一様分布乱数     // 指数乱数の取得     expDist = -log(1.0-uni\_random(mt))/variable;     return expDist;  } |

Random.hpp

|  |
| --- |
| #include <cmath>  #include <iomanip>  #include <iostream>  #include <random>  using namespace std;  class Random  {  private:      random\_device rnd; // 乱数用のシード  public:      double expDistribution(double variable); // 指数乱数生成関数  }; |

付録２（システムA）

main.cpp

|  |
| --- |
| #include <stdio.h>  #include "cmdline.h"  #include "MM3.hpp"  #include <tuple>  #define SERVICE\_NUM 3  using namespace std;  int main(int argc, char \*\*argv)  {      // 初期化      MM3 mm3;                       // MM3 クラスの宣言      cmdline::parser p;             // コマンドライン引数用の宣言      double rho;                    // 利用率      double system\_people = 0.0;    // システム内客数      double system\_time = 0.0;      // システム内a時間      tuple<double, double> results; // シミュレーション結果取得用変数      double start\_time = 1000.0;    // 過渡状態とみなすまでの時間      double end\_time = 11000.0;     // シミュレーション終了時間      int trials = 10;      /\* コマンドライン引数の定義      // −a 数値 とすると到着率　初期値は 1.0      // −s 数値 とするとサービス率を代入　初期値は 2.0 \*/      p.add<double>("arrival",'a', "arrival rate", false, 1.0);      p.add<double>("service",'s', "service rate", false, 2.0);      p.parse\_check(argc, argv);      double lambda = p.get<double>("arrival") / SERVICE\_NUM; // 到着率 \* 1/3      double mu = p.get<double>("service");     // サービス率      rho = (lambda / mu); // 到着率およびサービス率から利用率算出      FILE \*fp;      if ((fp = fopen("Results.txt", "w")) == NULL)      {          printf("file open error !!\n");          getchar();          exit(EXIT\_FAILURE);      }      // シミュレーション開始 (trials 回繰り返してその平均値を取得)      for (int i = 0; i < trials; i++)      {          results = mm3.simulation(lambda, mu, start\_time, end\_time);          system\_people = system\_people + get<0>(results); // システム内客数を取得          system\_time = system\_time + get<1>(results);     // システム内時間を取得          // fprintf(fp, "%lf,%lf,%lf\n", rho, get<0>(results), get<1>(results));          // fprintf(fp, "%lf,", get<0>(results));      }      fprintf(fp, "\n");      system\_people = system\_people / trials;      system\_time = system\_time / trials;      // 理論値および観測値を保存      //ここ直すだけ      double logical\_ave\_customers = rho / (1 - rho) \* 3;      double logical\_ave\_wait\_time = (1/mu) \*  (1 / (1 - rho)) \* 3;      // システム利用率ρ，システム内客数（サービスを受けている人＋サービスを待っている人），平均システム遅延（システム到着から出るまで）      fprintf(fp, "%s\t %s\t %s\n", "ρ", "システム内客数", "システム内時間");      fprintf(fp, "% lf\t % lf\t % lf\n", rho, logical\_ave\_customers, logical\_ave\_wait\_time);      fprintf(fp, "% lf\t % lf\t % lf\n", rho, system\_people, system\_time);      fclose(fp);  } |

MM3.cpp

|  |
| --- |
| #include "MM3.hpp"  #include <algorithm>  #include <random>  #define SERVICE\_NUM 3  std::tuple<double, double> MM3::simulation(double lambda, double mu, double startTime, double endTime)  {      // 初期化      vector<double> service;    // 窓口      queue<double> systems[SERVICE\_NUM]; //待ち行列      Random random;             // ランダムクラスの宣言      SimStat simStat = StandBy; // 過渡状態、過渡状態以降を定義      double service\_interval;   // サービス間隔      double arrive;             // 到着間隔      currentTime = 0.0;      simEndJobs = 0.0;      stayTime = 0.0;      visitors = 0.0;      // 処理開始（最初のイベントをイベントキューに登録）      event.selectedLine = random.genRand(0, 2);      event.time = currentTime + random.expDistribution(lambda);      event.eventState = ARRIVE;      eventQueue.push(event);      // 窓口初期化      for (int i = 0; i < SERVICE\_NUM; i++){          service.push\_back(-1.0);      }      int cnt\_selectedLines[3] = {0,0,0};      // 終了時間まで繰り返す      while (currentTime < endTime)      {          // 最初のイベントを取り出す          event = eventQueue.top();          eventQueue.pop();          // 現在時刻をイベント発生時刻まですすめる          currentTime = event.time;          switch (event.eventState)          {          // 到着イベントの場合          case ARRIVE:              // 待ち行列への登録              systems[event.selectedLine].push(event.time);              // イベントキューへのサービス到着イベントの登録              event.selectedLine = random.genRand(0, 2);              event.time = currentTime + random.expDistribution(lambda);              event.eventState = ARRIVE;              eventQueue.push(event);              break;          // サービス終了イベントの場合          case FINISH:              // 過渡状態中の統計値はリセットする              if (currentTime > startTime && simStat == StandBy)              {                  simStat = Execution;                  stayTime = 0.0;                  visitors = 0.0;                  simEndJobs = 0.0;              }              // 統計処理（待ち時間，客数等のカウント）              int cnt\_in\_service = 0;              for (int i = 0; i < SERVICE\_NUM; i++)              {                  if(service[i] != -1.0 ){                      cnt\_in\_service++;                  }              }              cnt\_selectedLines[event.selectedLine]++;              visitors += cnt\_in\_service + systems[0].size() + systems[1].size() + systems[2].size(); //自分のサービス分，１引く              // printf("visitors = %f\n", visitors);              // printf("%f - %f ,staytime = %f\n", currentTime, service[event.selectedLine], currentTime - service[event.selectedLine]);              stayTime += (currentTime - service[event.selectedLine]);              // printf("currentTime= %f, service[%d] = %f, stayTime = %f\n", currentTime, event.selectedLine, service[event.selectedLine], currentTime - service[event.selectedLine]);              simEndJobs++;              // 窓口のクリア（先頭の要素だけ削除）              if(service.size() > 0){                  // service.erase(service.begin() + selectedLine);                  service[event.selectedLine] = -1.0;              }              break;          }          // 3つの列の窓口で，並んでる列の窓口に人がいなくて，待ち行列に人がいる場合          for (int i = 0; i < SERVICE\_NUM; i++){              if (service[i] == -1.0 && systems[i].empty() == false)              {                  // 待ち行列の先頭から窓口への移動                  service[i] = systems[i].front();                  systems[i].pop();                  // debug                  // {for(int i=0;i<service.size();i++) printf("%f,",service[i]);}                  // printf("size = %lu\n", service.size());                  // イベントキューへのサービス到着イベントの登録                  event.selectedLine = i;                  event.time = currentTime + random.expDistribution(mu);                  event.eventState = FINISH;                  eventQueue.push(event);                  break;              }          }      }      for (int i = 0; i < SERVICE\_NUM; i++){          printf("%d, ", cnt\_selectedLines[i]);      }      printf("\n");      // 結果の返却      // return std::forward\_as\_tuple(システム内客数, システム内時間);      return std::forward\_as\_tuple(visitors / simEndJobs, stayTime / simEndJobs);  } |

Random.cpp

|  |
| --- |
| #include "Random.hpp"  double Random::expDistribution(double variable)  {     double expDist = 0.0;                             // 指数乱数用の宣言     mt19937 mt(rnd());                                // メルセンヌ・ツイスター用の宣言     uniform\_real\_distribution<> uni\_random(0.0, 1.0); // 0−1 までの一様分布乱数     // 指数乱数の取得     expDist = -log(1.0-uni\_random(mt))/variable;     return expDist;  }  int Random::genRand(int start, int end)  {      std::random\_device rnd;     // 非決定的な乱数生成器を生成      std::mt19937 mt(rnd());     //  メルセンヌ・ツイスタの32ビット版、引数は初期シード値      std::uniform\_int\_distribution<> rand0\_2(start, end);        // [0, 2] 範囲の一様乱数      return rand0\_2(mt);  } |

付録3（システムB）

main.cpp

|  |
| --- |
| #include <stdio.h>  #include "cmdline.h"  #include "MM3.hpp"  #include <tuple>  using namespace std;  int main(int argc, char \*\*argv)  {      // 初期化      MM3 mm3;                       // MM3 クラスの宣言      cmdline::parser p;             // コマンドライン引数用の宣言      double rho;                    // 利用率      double system\_people = 0.0;    // システム内客数      double system\_time = 0.0;      // システム内a時間      tuple<double, double> results; // シミュレーション結果取得用変数      double start\_time = 1000.0;    // 過渡状態とみなすまでの時間      double end\_time = 11000.0;     // シミュレーション終了時間      int trials = 10;      /\* コマンドライン引数の定義      // −a 数値 とすると到着率　初期値は 1.0      // −s 数値 とするとサービス率を代入　初期値は 2.0 \*/      p.add<double>("arrival",'a', "arrival rate", false, 1.0);      p.add<double>("service",'s', "service rate", false, 2.0);      p.parse\_check(argc, argv);      double lambda = p.get<double>("arrival"); // 到着率      double mu = p.get<double>("service");     // サービス率      rho = (lambda / mu); // 到着率およびサービス率から利用率算出      FILE \*fp;      if ((fp = fopen("Results.txt", "w")) == NULL)      {          printf("file open error !!\n");          getchar();          exit(EXIT\_FAILURE);      }      // シミュレーション開始 (trials 回繰り返してその平均値を取得)      for (int i = 0; i < trials; i++)      {          results = mm3.simulation(lambda, mu, start\_time, end\_time);          system\_people = system\_people + get<0>(results); // システム内客数を取得          system\_time = system\_time + get<1>(results);     // システム内時間を取得          // fprintf(fp, "%lf,%lf,%lf\n", rho, get<0>(results), get<1>(results));          fprintf(fp, "%lf,", get<0>(results));      }      fprintf(fp, "\n");      system\_people = system\_people / trials;      system\_time = system\_time / trials;      // 理論値および観測値を保存      //ここ直すだけ      double logical\_ave\_customers = (rho \* (9-rho) \* (rho + 3)) / ((3-rho) \* ((rho \* rho) - 4 \* rho + 6));      double logical\_ave\_wait\_time = logical\_ave\_customers / lambda;      // システム利用率ρ，システム内客数（サービスを受けている人＋サービスを待っている人），平均システム遅延（システム到着から出るまで）      fprintf(fp, "% lf\t % lf\t % lf\n", rho, logical\_ave\_customers, logical\_ave\_wait\_time);      fprintf(fp, "% lf\t % lf\t % lf\n", rho, system\_people, system\_time);      fclose(fp);  } |

MM3.cpp

|  |
| --- |
| #include "MM3.hpp"  #include <algorithm>  std::tuple<double, double> MM3::simulation(double lambda, double mu, double startTime, double endTime)  {      // 初期化      vector<double> service;    // 窓口      queue<double> system;      // 待ち行列      Random random;             // ランダムクラスの宣言      SimStat simStat = StandBy; // 過渡状態、過渡状態以降を定義      double service\_interval;   // サービス間隔      double arrive;             // 到着間隔      currentTime = 0.0;      simEndJobs = 0;      stayTime = 0.0;      visitors = 0.0;      // 処理開始（最初のイベントをイベントキューに登録）      event.time = currentTime + random.expDistribution(lambda);      event.eventState = ARRIVE;      eventQueue.push(event);      // 終了時間まで繰り返す      while (currentTime < endTime)      {          // 最初のイベントを取り出す          event = eventQueue.top();          eventQueue.pop();          // 現在時刻をイベント発生時刻まですすめる          currentTime = event.time;          switch (event.eventState)          {          // 到着イベントの場合          case ARRIVE:              // 待ち行列への登録              system.push(event.time);              // イベントキューへのサービス到着イベントの登録              event.time = currentTime + random.expDistribution(lambda);              event.eventState = ARRIVE;              eventQueue.push(event);              break;          // サービス終了イベントの場合          case FINISH:              // 過渡状態中の統計値はリセットする              if (currentTime > startTime && simStat == StandBy)              {                  simStat = Execution;                  stayTime = 0;                  visitors = 0;                  simEndJobs = 0;              }              // 統計処理（待ち時間，客数等のカウント）              stayTime += currentTime - service[0];              visitors += system.size() + service.size() - 1.0;              simEndJobs++;              // 窓口のクリア（先頭の要素だけ削除）              if(service.size() > 0){                  service.erase(service.begin());              }              break;          }          // 3つの窓口のうち，一つでもあいていて、待ち行列に人がいる場合          // if (service.empty() && system.empty() == false)          if (service.size() < 3 && system.empty() == false)          {              // 待ち行列の先頭から窓口への移動              service.push\_back(system.front());              system.pop();              // 窓口の対応終了時間が早い順(降順)で並び替え              std::sort(service.begin(), service.end(), std::greater<int>());              // debug              // {for(int i=0;i<service.size();i++) printf("%f,",service[i]);}              // printf("size = %lu\n", service.size());              // イベントキューへのサービス到着イベントの登録              event.time = currentTime + random.expDistribution(mu);              event.eventState = FINISH;              eventQueue.push(event);          }          // queue<double> tmp = system;          // for (int i = 0; i < tmp.size(); i++){          //     printf("%f, cnt = %d", tmp.front(), i);          //     tmp.pop();          // }          // printf("\n");      }      // 結果の返却      // return std::forward\_as\_tuple(システム内客数, システム内時間);      return std::forward\_as\_tuple(visitors / simEndJobs, stayTime / simEndJobs);  } |

付録4（システムC）

main.cpp

|  |
| --- |
| #include <stdio.h>  #include "cmdline.h"  #include "MM1.hpp"  #include <tuple>  using namespace std;  int main(int argc, char \*\*argv)  {      // 初期化      MM1 mm1;                       // MM1 クラスの宣言      cmdline::parser p;             // コマンドライン引数用の宣言      double rho;                    // 利用率      double system\_people = 0.0;    // システム内客数      double system\_time = 0.0;      // システム内a時間      tuple<double, double> results; // シミュレーション結果取得用変数      double start\_time = 1000.0;    // 過渡状態とみなすまでの時間      double end\_time = 11000.0;     // シミュレーション終了時間      int trials = 10;      /\* コマンドライン引数の定義      // −a 数値 とすると到着率　初期値は 1.0      // −s 数値 とするとサービス率を代入　初期値は 2.0 \*/      p.add<double>("arrival",'a', "arrival rate", false, 1.0);      p.add<double>("service",'s', "service rate", false, 2.0);      p.parse\_check(argc, argv);      double lambda = p.get<double>("arrival"); // 到着率      double mu = p.get<double>("service");     // サービス率      rho = (lambda / (mu\*3)); // 到着率およびサービス率から利用率算出      FILE \*fp;      if ((fp = fopen("Results.txt", "w")) == NULL)      {          printf("file open error !!\n");          getchar();          exit(EXIT\_FAILURE);      }      // シミュレーション開始 (trials 回繰り返してその平均値を取得)      for (int i = 0; i < trials; i++)      {          results = mm1.simulation(lambda, mu, start\_time, end\_time);          system\_people = system\_people + get<0>(results); // システム内客数を取得          system\_time = system\_time + get<1>(results);     // システム内時間を取得          // fprintf(fp, "%lf,%lf,%lf\n", rho, get<0>(results), get<1>(results));          fprintf(fp, "%lf,", get<0>(results));      }      fprintf(fp, "\n");      system\_people = system\_people / trials;      system\_time = system\_time / trials;      // 理論値および観測値を保存      //ここ直すだけ      double logical\_ave\_customers = rho / (1 - rho);      double logical\_ave\_wait\_time = (1/(mu\*3)) \*  (1 / (1 - rho));      // システム利用率ρ，システム内客数（サービスを受けている人＋サービスを待っている人），平均システム遅延（システム到着から出るまで）      fprintf(fp, "% lf\t % lf\t % lf\n", rho, logical\_ave\_customers, logical\_ave\_wait\_time);      fprintf(fp, "% lf\t % lf\t % lf\n", rho, system\_people, system\_time);      fclose(fp);  } |

MM3.cpp

|  |
| --- |
| #include "MM1.hpp"  std::tuple<double, double> MM1::simulation(double lambda, double mu, double startTime, double endTime)  {      // 初期化      vector<double> service;    // 窓口      queue<double> system;      // 待ち行列      Random random;             // ランダムクラスの宣言      SimStat simStat = StandBy; // 過渡状態、過渡状態以降を定義      double service\_interval;   // サービス間隔      double arrive;             // 到着間隔      currentTime = 0.0;      simEndJobs = 0;      stayTime = 0.0;      visitors = 0.0;      // 処理開始（最初のイベントをイベントキューに登録）      event.time = currentTime + random.expDistribution(lambda);      event.eventState = ARRIVE;      eventQueue.push(event);      // 終了時間まで繰り返す      while (currentTime < endTime)      {          // 最初のイベントを取り出す          event = eventQueue.top();          eventQueue.pop();          // 現在時刻をイベント発生時刻まですすめる          currentTime = event.time;          switch (event.eventState)          {          // 到着イベントの場合          case ARRIVE:              // 待ち行列への登録              system.push(event.time);              // イベントキューへのサービス到着イベントの登録              event.time = currentTime + random.expDistribution(lambda);              event.eventState = ARRIVE;              eventQueue.push(event);              break;          // サービス終了イベントの場合          case FINISH:              // 過渡状態中の統計値はリセットする              if (currentTime > startTime && simStat == StandBy)              {                  simStat = Execution;                  stayTime = 0;                  visitors = 0;                  simEndJobs = 0;              }              // 統計処理（待ち時間，客数等のカウント）              stayTime += currentTime - service[0];              visitors += system.size();              simEndJobs++;              // 窓口のクリア              service.clear();              break;          }          // 窓口があいていて、待ち行列に人がいる場合          // if (service.empty() && system.empty() == false)          if (service.empty() && system.empty() == false)          {              // 待ち行列の先頭から窓口への移動              service.push\_back(system.front());              system.pop();              // イベントキューへのサービス到着イベントの登録              event.time = currentTime + random.expDistribution(3\*mu);              event.eventState = FINISH              eventQueue.push(event);          }          // printf("%f \n", currentTime);      }      // 結果の返却      // return std::forward\_as\_tuple(システム内客数, システム内時間);      return std::forward\_as\_tuple(visitors / simEndJobs, stayTime / simEndJobs);  } |