アーランの即時式モデル

到達目標

- アーランの即時式モデルを理解する
- アーランの即時式モデルを使って，サーバがs台あり、待ち行列がないときのジョブ棄却率を求めることができるようになる
- アーランの即時式モデルについて，自分の言葉で説明できるようになる
  - 待ち行列モデルの種類，待ち行列長，システム内のジョブ数，到着したジョブの振るまい，定常確率

あらすじ

1. 「M/M/S 待ち行列モデル」の定常状態

2. 「アーランの即時式モデル」と「M/M/S 待ち行列モデル」の関係
   - システム内の占有サーバ数がSのとき，到着したジョブは棄却される

3. アーランの即時式モデルを使った，ジョブ棄却率の計算

M/M/S 待ち行列モデル

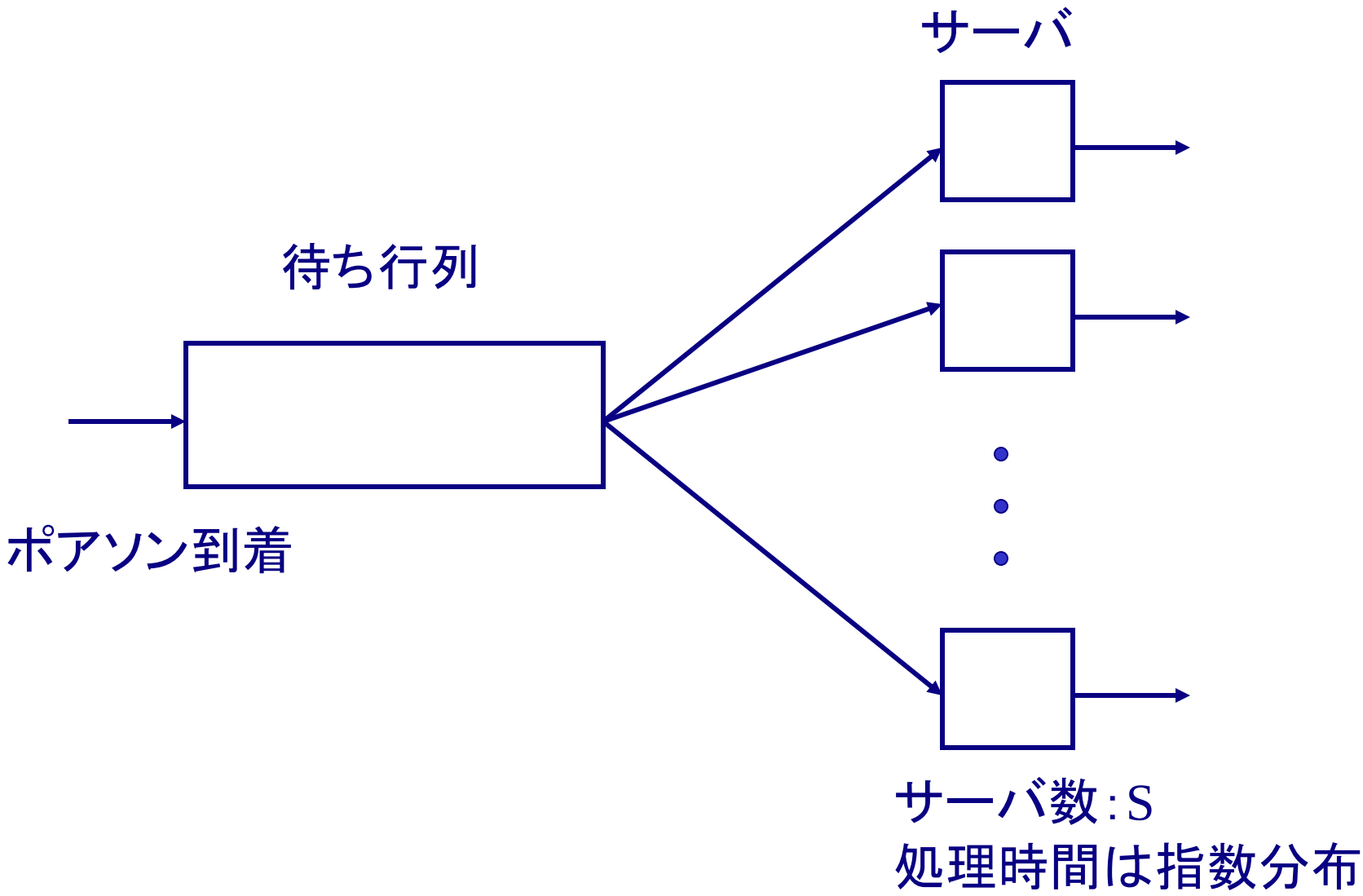
M/M/S 待ち行列モデル

到着過程: ポアソン分布

処理時間分布: 指数分布

サーバ数: S個

M/M/S 待ち行列モデル



M/M/S 待ち行列モデルの解析

- 状態遷移
  - システム内のジョブ数を「状態」と見る
- 定常状態
  - 定常状態での、各「状態」の確率を求める
- システムの「処理率」を求める

状態

状態0：　システム内のジョブ数が　0

状態1：　システム内のジョブ数が　1

⋮

状態S:　システム内のジョブ数がS

⋮

状態遷移

- ジョブが到着:

  状態 k から状態 k+1 に遷移

- ジョブが完了:

  状態 k から状態 k−1 に遷移

遷移確率に関する方程式

- 微小時間　Δt（限りなく0に近い）についての式
- 「ジョブの到着」と「ジョブの完了」は、「同時」には起きない
- 「ジョブの完了」の確率は、Δt と μの式
- 「ジョブの到着」の確率は、Δt と λの式

「ジョブの完了」に関する式

- k≦S ならば（kは状態番号）
  - サーバに空きがある

$$k\mu\Delta t\,(1-\mu\Delta t) \overset{k-1}{\fallingdotseq} k\mu\Delta t$$

- k＞Sならば（kは状態番号）
  - サーバは全て忙しい

$$S\mu\Delta t\,(1-\mu\Delta t) \overset{S-1}{\fallingdotseq} S\mu\Delta t$$

「ジョブの到着」に関する式

$\lambda \Delta t$

状態遷移

- ジョブが到着:

  状態 k から状態 k+1 に遷移

  $\lambda \Delta t$

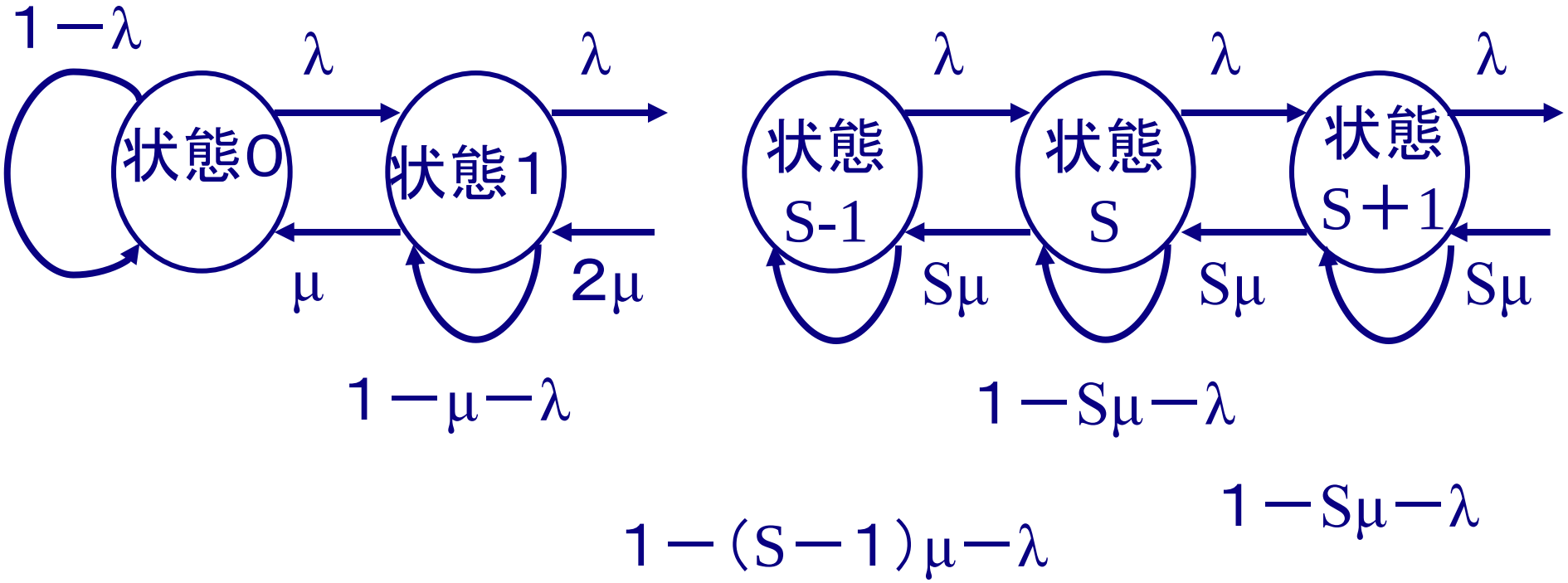
- ジョブが完了:

  状態 k から状態 k－1 に遷移

  k≦S ならば: $k\mu\Delta t$

  k＞Sならば: $S\mu\Delta t$

状態遷移図



定常状態方程式

0＜n＜S では

$$(\lambda + n\mu)P_n = \lambda P_{n-1} + (n+1)\mu P_{n+1}$$

S ≦n　では

$$(\lambda + S\mu)P_n = \lambda P_{n-1} + S\mu P_{n+1}$$

$$\lambda P_0 = \mu P_1$$

定常確率をP_0で表す

0＜n＜S では

$$Pn = \left(\frac{\lambda}{\mu}\right)^{n} P0 \frac{}{n!}$$

S ≦n　では

$$Pn = \left(\frac{\lambda}{\mu}\right)^{n} P0 \frac{}{S!S}\ n\text{-}s$$

システム処理能力ρ

$\rho = \lambda/\mu$

- $\lambda \Delta t$: 「時間 $(t, t + \Delta t)$ に到着するジョブ数」の平均
- $\mu \Delta t$: 「サーバがジョブを処理中の間，Δt 内に完了する処理数」の平均

待ち合わせが生じる確率

- システム内のジョブ数が S 以上

- $$Pqueue = \sum_{n=S}^{\infty} Pn$$

$$= \sum_{n=S}^{\infty} \left(\frac{\lambda}{\mu}\right)^{n} \frac{P0}{S!S^{n-S}}$$

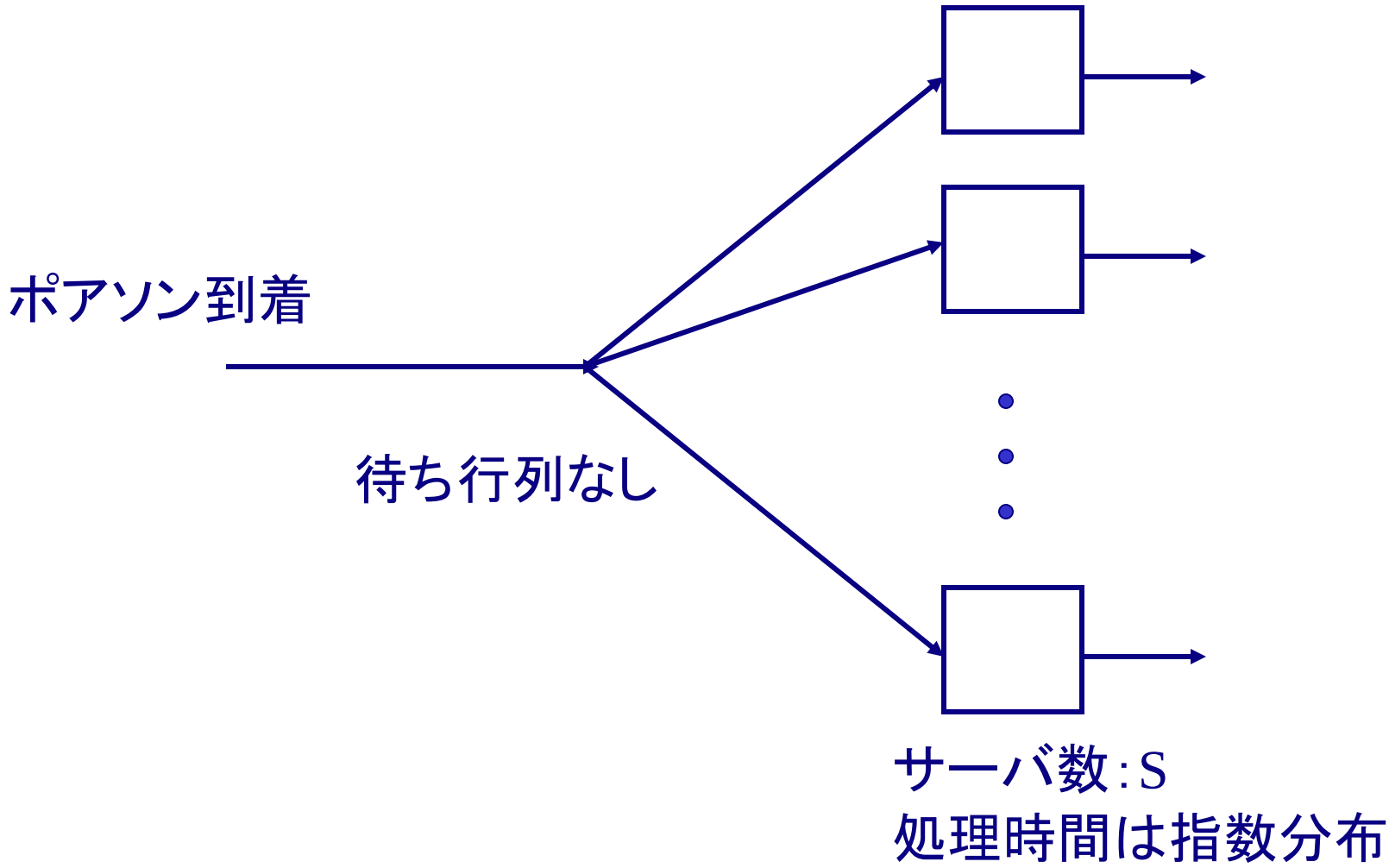
$$= \sum_{n=S}^{\infty} P0 \frac{(S\rho)^{n}}{S!} \frac{1}{S^{n-S}}$$

アーランの即時式モデル

アーランの即時式モデル

- サーバがs台あり、待ち行列がない（M/M/S/S モデル）ときのジョブ棄却率を求めよう

待ち行列モデルM/M/S/S



アーランの即時式モデル

- 待ち行列モデル: M/M/S/S
- 入力回線数: 無限
- 待ち行列長: L＝0
- システム内の最大占有サーバ数: K＝S

「即時式」の意味

- 待ち行列なし（待ち行列長が0）
- システム内の占有サーバ数がSのとき，到着したジョブは棄却される

ジョブのモデル

- ジョブの到着
  - 平均λのポアソン分布
  - → ある時刻に客が到着してから時間t内に次の客が到着する確率はポアソン分布に従う: $1-e^{-\lambda t}$
  - 微少時間Δtの間にジョブが到着する確率: $\lambda \Delta t$
- ジョブの処理時間
  - 平均$1/\mu$の指数分布
  - 微少時間Δtの間に処理が終わる確率: $\mu \Delta t$

システム処理能力: $\rho = \lambda/\mu$

課題

- ジョブ到着と，使用されるサーバ数の関係を明らかにする
- 「最適」なサーバ数の設計などに利用

状態

状態0:　系内のジョブの数が　0

状態1:　系内のジョブの数が　1

⋮

状態S:　系内のジョブの数が　S

（状態はSまで）

状態遷移

- ジョブが到着:

  1) k＜S のとき

  状態 k から状態 k+1 に遷移

  2) k=S のとき

  状態Sのまま(新しい到着は棄却される)

- ジョブの回線の占有終わり:

  状態 k から状態 k−1 に遷移

状態遷移

- ジョブが到着:

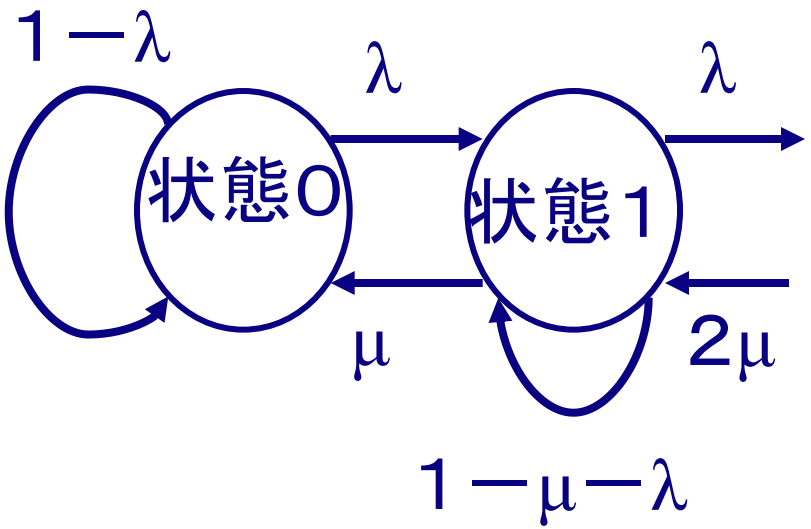
  1) k＜S のとき

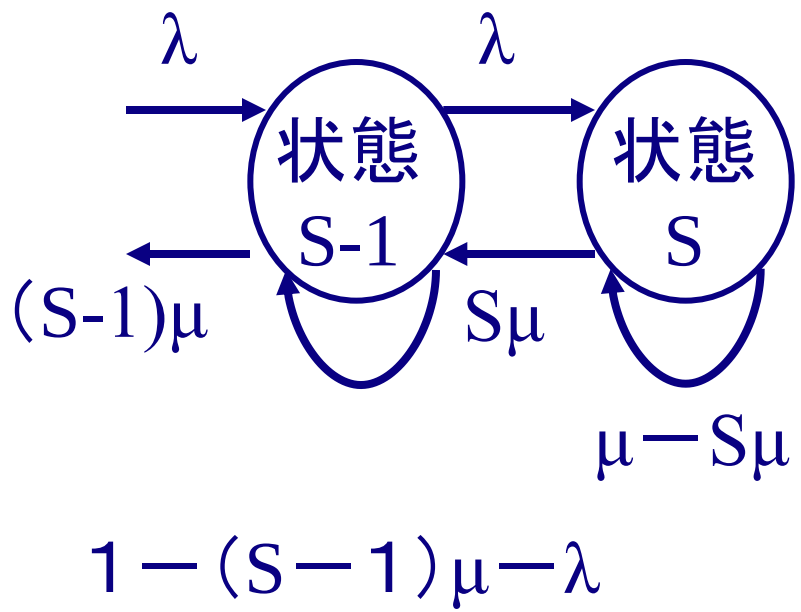
  状態 k から状態 k+1 に遷移: $\lambda \Delta t$

- ジョブの回線の占有終わり:

  状態 k から状態 k−1 に遷移 : $k\mu \Delta t$

状態遷移図





定常状態方程式

$$(\lambda + n\mu)P_n = \lambda P_{n-1} + (n+1)\mu P_{n+11}$$

$$\lambda P_0 = \mu P_1$$

$$\lambda P_{n-1} = S\mu P_n$$

棄却率

- 定常状態で，系内にn個のジョブ（0≦n≦s）がある確率を Pn とすると

$$P_n(t) = \frac{\frac{\rho^n}{n!}}{1 + \rho + \frac{\rho^2}{2} + \ldots + \frac{\rho^s}{s!}}$$

- 棄却率：
  - システム内の占有サーバ数が Sになる確率：Ps
  - 上の式で，n=s として計算する

例題1．アーランの即時式モデル

- アーランの即時式モデルを使って，サーバがs台あり、n 個のジョブがある確率 Pn を求めるプログラムを作成し，実行せよ
  - λとμをいろいろ変化させて実行せよ
  - また，λ > μのときの振る舞いを確認せよ

例題1．アーランの即時式モデル プログラムの説明

- ソース：erlung.c
- 変数：Lambda, Mu, s
- 返り値：n人が系内にいる確率Pn
- 実行後パラメータ3つを入力するとoutfile.datというファイルにデータを出力する
  - 入力例：0.05,0.1,3

例題1．アーランの即時式モデル プログラムの使い方

- コンパイルして実行後メッセージが出るのでパラメータを入力するとファイルにデータが出力される。
  - 入力データは「,」で区切る
- Gnuplotに対応した形式で出力するので次のようなコマンドをGnuplotで使うとグラフ化できる

plot ‘outfile.dat’ with points

```
#include<stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

main()
{
    FILE *outfile;
    int s, n, n_fact;
    double lambda, mu, n_sum,r, rho;
    if ((outfile=fopen("output.dat", "w")) == NULL) {
        printf("can't open file %s¥n", outfile);
        exit(0);
    }
    printf("Input (lambda,mu,s) :");
    scanf("%lf,%lf,%d", &lambda, &mu, &s);
    rho=lambda/mu;

    fprintf(outfile,"#n          Pn¥n");
```

```
    for (n=1;n<=s;n++) {
        int i, s_fact, n_fact; /*_fact=factorial*/
        double bunbo, bunshi;
        n_fact=1;                   /*calculate n factorial*/
        for(i=1; i<n+1; i++) {
            n_fact = n_fact*i;
        }
        bunbo=1.0;  /*Inistial value of BUNBO*/
        s_fact = 1;
        for(i=0; i<s; i++){
            s_fact = s_fact*(i+1);          /*s factorial*/
            bunbo = bunbo+ (pow(rho,i+1)/s_fact); /*bunbo*/
       }
       bunshi=pow(rho,n)/n_fact;
       fprintf(outfile,"%d     %f¥n", n, bunshi/bunbo);    /*ファイルに出力*/
   }
   fclose(outfile);
}
```

例題2．呼損率（Loss Probability）プログラム

- ソース：lossprob.c
- 変数：MaxLambda, MinLambda, Mu, s
- 返り値：Lambdaを変化させたときの呼損率
- アーランの公式を用いて呼損率を計算。アーランの公式でn=sとする。LambdaをMaxからMinまで100個に分割して計算。
- 実行後パラメータ4つ入力するとloss.datというファイルにデータを出力する
  - 入力例：0.01,0.1,0.1,3

```
 #include<stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#define NUM_DIV 100
main()
{
  /*
    lambda = arraival rate
    mu = process rate
    rho = utilization rate
   */
    FILE *outfile;
    int s, n, n_fact,mintime,rate,j;
    double lambda,mu,n_sum,r,rho,maxrate,minrate;
    if ((outfile=fopen("loss.dat", "w")) == NULL) {
        printf("can't open file %s¥n", outfile);
        exit(0);
    }
    printf("Input (Max arrival rate,Min arrival rate, Process rate ,Number of Servers) :");
    scanf("%lf,%lf,%lf,%d",&maxrate, &minrate, &mu, &s);
```

```
    fprintf(outfile,"#lambda              Pn¥n");
    lambda=0.0;
    for (j=0;j<NUM_DIV;j++) {
        lambda = (maxrate-minrate)*(j+1)/NUM_DIV;
        rho = lambda/mu;
        int i, s_fact, n_fact; /*_fact=factorial*/
        double bunbo,bunshi;
        n_fact=1;                   /*calculate n factorial*/
        for(i=1; i<s+1; i++){
              n_fact = n_fact*i;
        }
        bunbo = 1.0;  /*Inistial value of BUNBO*/
        s_fact = 1;
        for(i=0; i<s; i++){
              s_fact = s_fact*(i+1);        /*s factorial*/
              bunbo = bunbo+ (pow(rho,i+1)/s_fact); /*bunbo*/
        }
        bunshi = pow(rho,s)/n_fact;
        fprintf(outfile,"%f      %f¥n",lambda,bunshi/bunbo);
    }
    fclose(outfile);
}
```