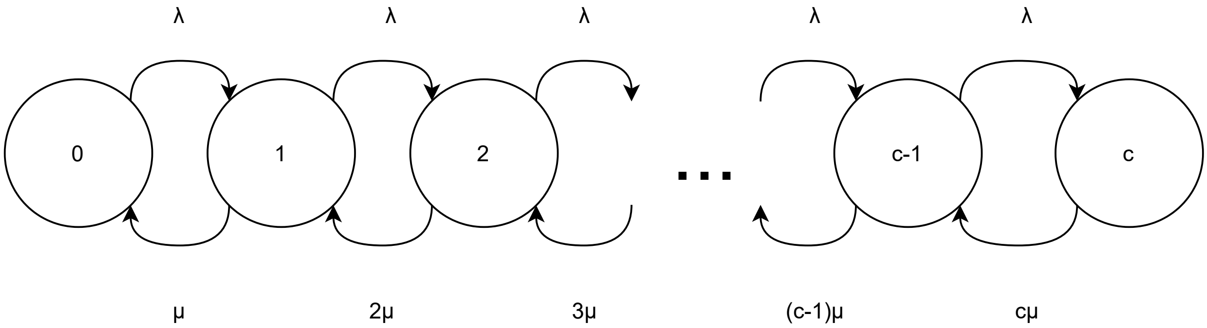
# Συστήματα Αναμονής
# 4η εργαστηριακή άσκηση

Νικόλαος Παγώνας, el18175

## Ανάλυση και Σχεδιασμός τηλεφωνικού κέντρου

### (1)

Σχεδιάζουμε το διάγραμμα ρυθμού μεταβάσεων για το σύστημα M/M/c/c:




Με βάση το παραπάνω διάγραμμα, διατυπώνουμε τις τοπικές εξισώσεις ισορροπίας:

$$\begin{aligned} P_k &= \frac{\lambda}{k\mu} P_{k-1} \\ &= \frac{\lambda}{k\mu} \cdot \frac{\lambda}{(k-1)\mu} \cdot \frac{\lambda}{(k-2)\mu} \cdots \frac{\lambda}{2\mu} \cdot \frac{\lambda}{\mu} \cdot P_0 \\ &= \frac{1}{k!} \left(\frac{\lambda}{\mu}\right)^k P_0 \\ &= \frac{\rho^k}{k!} P_0 \end{aligned}$$

Και λόγω της ιδιότητας της κανονικοποίησης:

$$\sum_{k=0}^{c} P_k = 1 \implies \sum_{k=0}^{c} \frac{\rho^k}{k!} P_0 = 1 \implies P_0 = \frac{1}{\displaystyle\sum_{k=0}^{c} \frac{\rho^k}{k!}} \implies \boxed{P_c = P_{blocking} = \frac{\rho^c/c!}{\displaystyle\sum_{k=0}^{c} \frac{\rho^k}{k!}}}$$

Επομένως:

$$\begin{aligned} \text{Μέσος ρυθμός απωλειών} &= \lambda - \gamma \\ &= \lambda \cdot P_{\text{blocking}} \\ &= \lambda \cdot \frac{\rho^c / c!}{\displaystyle\sum_{k=0}^{c} \frac{\rho^k}{k!}} \end{aligned}$$

Τέλος, υλοποιούμε την συνάρτηση `erlangb_factorial`, η οποία υπολογίζει το

$$B(\rho, c) = \frac{\rho^c / c!}{\sum_{k=0}^{c} \frac{\rho^k}{k!}}, \text{ ως εξής:}$$

```
function B_result = erlangb_factorial(rho, c)
  summation = 0;
  for k = 0:c
    summation = summation + rho^k/factorial(k);
  endfor

  B_result = rho^c/factorial(c)/summation;
endfunction
```

Η ορθή λειτουργία επαληθεύτηκε με χρήση της συνάρτησης `erlang` του πακέτου `queueing`.

### (2)

Τώρα θα υλοποιήσουμε ξανά την παραπάνω συνάρτηση, αυτή τη φορά με επαναληπτικό τρόπο. Ονομάζουμε την νέα υλοποίηση `erlangb_iterative`:

```
function B_result = erlangb_iterative(rho, c)
  retval = 0;

  B = ones(1);

  for n = 1:c
    B(n+1) = ( rho .* B(n) ) / ( rho .* B(n) + n );
  endfor

  B_result = B(c+1);


endfunction
```

Και πάλι ελέγχουμε την ορθότητα μέσω της συνάρτησης `queueing/erlang`.

### (3)

Αν τρέξουμε τις συναρτήσεις `erlangb_factorial` και `erlangb_iterative` με παραμέτρους ρ=c=1024, παρατηρούμε ότι η πρώτη επιστρέφει `NaN` (Not a Number). Αυτό συμβαίνει διότι για το Octave, $1024^{1024} = \text{Inf}$ και $1024! = \text{Inf}$, οπότε έχουμε διαίρεση Inf/Inf η οποία επιστρέφει NaN. Αντίθετα, η δεύτερη επιστρέφει το σωστό αποτέλεσμα (0.024524), γιατί ο αλγόριθμος είναι επαναληπτικός και διατηρεί τις τιμές του σε λογικά όρια, από μετάβαση σε μετάβαση.

**(4)**

**(a)**

Μία μοντελοποίηση για έναν πελάτη (όχι η μοναδική) είναι να θεωρήσουμε:

$$\lambda = 1\ \frac{\text{κλήση}}{\text{ώρα}}\ \text{και}\ \frac{1}{\mu} = 23\frac{\text{λεπτά}}{\text{κλήση}}$$

Έτσι,

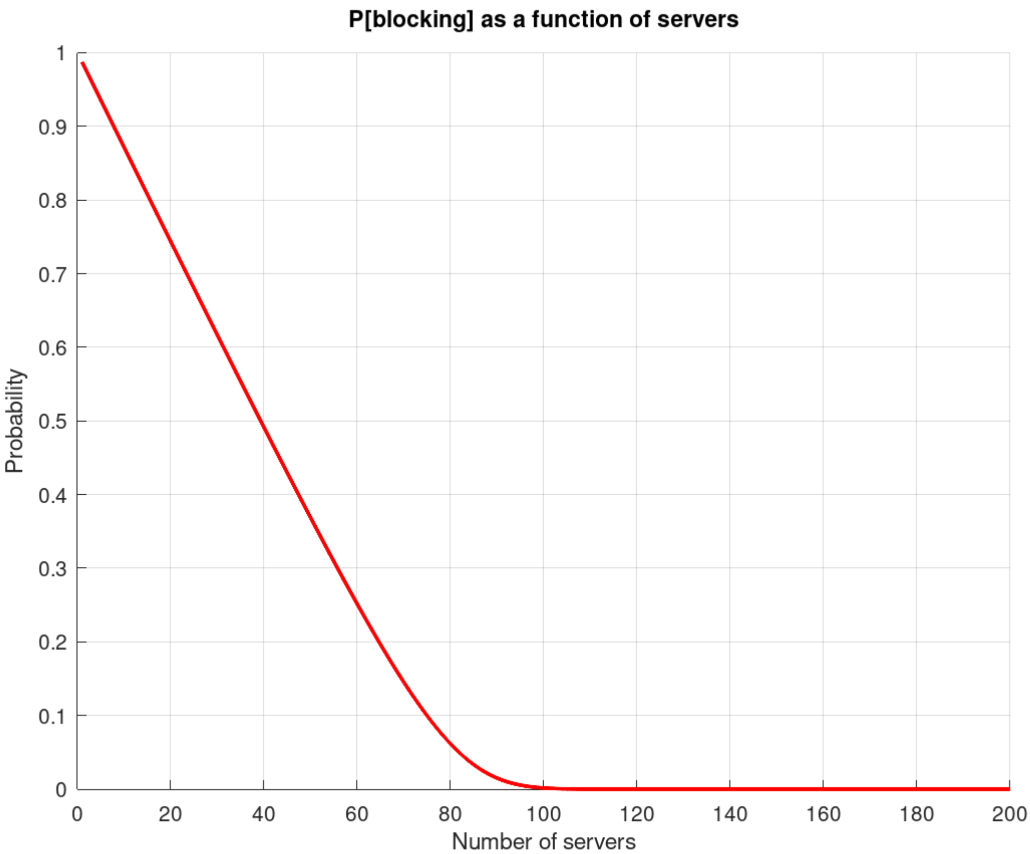
$$\rho_1 = \frac{\lambda}{\mu} = 1\ \frac{\text{κλήση}}{\text{ώρα}} \cdot 23\ \frac{\text{λεπτά}}{\text{κλήση}} = \frac{1\ \text{κλήση}}{60\ \text{λεπτά}} \cdot \frac{23\ \text{λεπτά}}{1\ \text{κλήση}} = \frac{23}{60}\ Erlangs.$$

Για 200 πελάτες έχουμε:

$$\rho_{200} = 200 \cdot \rho_1 = 76.67\ Erlangs.$$

**(b)**

Απεικονίζουμε την πιθανότητα απόρριψης πελάτη συναρτήσει του αριθμού των τηλεφωνικών γραμμών (1...200):



### (c)

Ο ελάχιστος αριθμός τηλεφωνικών γραμμών που χρειαζόμαστε προκειμένου να έχουμε πιθανότητα απόρριψης μικρότερη του 1% είναι 93 εξυπηρετητές, οι οποίοι δίνουν P[blocking] = 0.00836795 (0.83%).

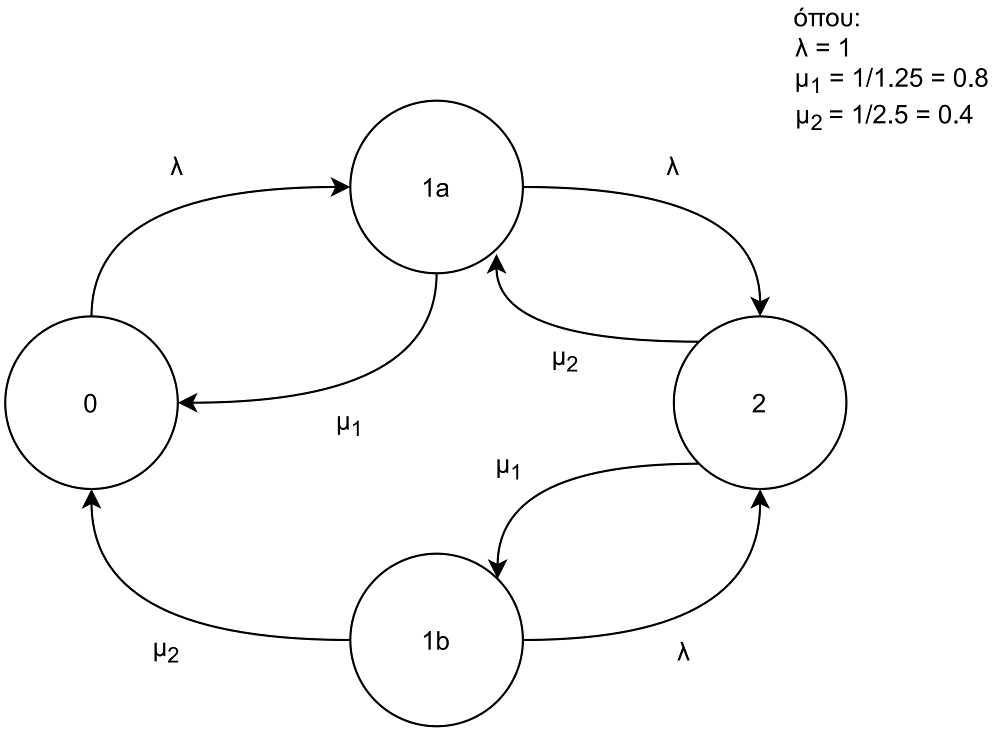
## Κώδικας που χρησιμοποιήθηκε για το ερώτημα 4

```
clc;
clear all;
close all;

rho_for_one = 23/60;
total_rho = 200 * rho_for_one;

Pblocking = zeros(1);

found_min_servers_needed = false;

for c = 1:200
  Pblocking(c) = erlangb_iterative(total_rho, c);
  if Pblocking(c) < 0.01 && !found_min_servers_needed
    min_servers_needed = c;
    found_min_servers_needed = true;
  endif
endfor

line = linspace(0.01, 0.01, 200);

hold on;

plot(Pblocking,"r", "linewidth", 2);
grid on;
xlabel("Number of servers");
ylabel("Probability");
title("P[blocking] as a function of servers");
xticks(0:20:200);
yticks(0:0.1:1);

saveas(1,"call_center.png");

printf("Minimum number of servers needed = %d\n", min_servers_needed);
printf("%d servers give P[blocking] = %d\n", min_servers_needed, Pblocking(
    min_servers_needed));
```

## Σύστημα εξυπηρέτησης με δύο ανόμοιους εξυπηρετητές

### (1)

Σχεδιάζουμε το διάγραμμα ρυθμών μεταβάσεων του συστήματος στην κατάσταση ισορροπίας:




Για τις εξισώσεις ισορροπίας παίρνουμε την τομή που περικλείει μόνο τον κόμβο 0, την τομή που περικλείει μόνο τον κόμβο 1a και την τομή που περικλείει μόνο τον κόμβο 1b, καθώς και την ιδιότητα της κανονικοποίησης, οπότε έχουμε:

$$1 \cdot P_0 = 0.8 \cdot P_{1a} + 0.4 \cdot P_{1b} \tag{1}$$

$$0.8 \cdot P_{1a} + 1 \cdot P_{1a} = 1 \cdot P_0 + 0.4 \cdot P_2 \tag{2}$$

$$1 \cdot P_{1b} + 0.4 \cdot P_{1b} = 0.8 \cdot P_2 \tag{3}$$

$$P_0 + P_{1a} + P_{1b} + P_2 = 1 \tag{4}$$

Λύνοντας το παραπάνω 4x4 σύστημα προκύπτουν:

$$P_0 = 0.24951, P_{1a} = 0.21442, P_{1b} = 0.19493, P_2 = 0.34113$$

Επομένως,

$$P[\text{blocking}] = P_2 = 0.34113 = 34.1\%$$

Για τον μέσο αριθμό πελατών στο σύστημα ισχύει:

$$E[n] = 0 \cdot P_0 + 1 \cdot (P_{1a} + P_{1b}) + 2 \cdot P_2 = 1.09161 \text{ πελάτες}$$

**(2)**

Για τα κατώφλια έχουμε:

$$\text{threshold\_1a} = \frac{\lambda}{\lambda + \mu_1}$$

$$\text{threshold\_1b} = \frac{\lambda}{\lambda + \mu_2}$$

$$\text{threshold\_2\_first} = \frac{\lambda}{\lambda + \mu_1 + \mu_2}$$

$$\text{threshold\_2\_second} = \frac{\lambda + \mu_1}{\lambda + \mu_1 + \mu_2}$$

Η λογική είναι ότι στα threshold 1a,1b και 2_first, όσο μεγαλύτερο είναι το λ, τόσο μεγαλύτερη και η πιθανότητα ο τυχαίος αριθμός να πέσει μέσα στο διάστημα [0,threshold]. Το threshold_2_second έχει φτιαχτεί έτσι ώστε να χωρίσουμε το διάστημα [0,1] σε τρία κομμάτια:

- $\left[0 \ , \ \dfrac{\lambda}{\lambda + \mu_1 + \mu_2}\right]$
- $\left[\dfrac{\lambda}{\lambda + \mu_1 + \mu_2} \ , \ \dfrac{\lambda + \mu_1}{\lambda + \mu_1 + \mu_2}\right]$
- $\left[\dfrac{\lambda + \mu_1}{\lambda + \mu_1 + \mu_2} \ , \ 1\right]$,

Τα οποία έχουν μήκος $\dfrac{\lambda}{\lambda + \mu_1 + \mu_2}$, $\dfrac{\mu_1}{\lambda + \mu_1 + \mu_2}$ και $\dfrac{\mu_2}{\lambda + \mu_1 + \mu_2}$ αντίστοιχα.

Γενικά για κάθε κατάσταση, χωρίζουμε το διάστημα [0,1] σε τόσα υποδιαστήματα, όσες και οι διαφορετικές καταστάσεις στις οποίες μπορούμε να πάμε από την κατάσταση αυτή.

Το κριτήριο σύγκλισης είναι:

```
if abs(mean_clients - previous_mean_clients) < 0.00001
    break;
endif
```

Στην ουσία ελέγχουμε ανά 1000 μεταβάσεις (`if mod(time, 1000) == 0`) αν ο μέσος αριθμός πελατών αλλάζει πολύ λίγο (κατά 0.00001), δηλαδή αν έχουμε σύγκλιση.

Οι πιθανότητες που προκύπτουν από το πρόγραμμα είναι:

```
P[0] = 0.247081
P[1a] = 0.213623
P[1b] = 0.198007
P[2] = 0.341289
```

οι οποίες συμφωνούν κατά πολύ μεγάλο βαθμό με τις θεωρητικές πιθανότητες που υπολογίστηκαν στο προηγούμενο ερώτημα.

### Κώδικας που χρησιμοποιήθηκε για την προσομοίωση, με συμπληρωμένα τα κενά

```
clc;
clear all;
close all;


lambda = 1;
m1 = 0.8;
m2 = 0.4;

threshold_1a = lambda / (lambda + m1);
threshold_1b = lambda / (lambda + m2);
threshold_2_first = lambda / (lambda + m1 + m2);
threshold_2_second = (lambda + m1) / (lambda + m1 + m2);

current_state = 0;
arrivals = zeros(1,4);
total_arrivals = 0;
maximum_state_capacity = 2;
previous_mean_clients = 0;
delay_counter = 0;
time = 0;

while true
  time = time + 1;

  if mod(time,1000) == 0
    for i=1:1:4
      P(i) = arrivals(i)/total_arrivals;
    endfor

    delay_counter = delay_counter + 1;

    mean_clients = 0*P(1) + 1*P(2) + 1*P(3) + 2*P(4);

    delay_table(delay_counter) = mean_clients;

    if abs(mean_clients - previous_mean_clients) < 0.00001
      break;
    endif
    previous_mean_clients = mean_clients;
  endif

  random_number = rand(1);

  if current_state == 0
      current_state = 1;
      arrivals(1) = arrivals(1) + 1;
      total_arrivals = total_arrivals + 1;
  elseif current_state == 1
    if random_number < threshold_1a
      current_state = 3;
      arrivals(2) = arrivals(2) + 1;
      total_arrivals = total_arrivals + 1;
    else
      current_state = 0;
    endif
  elseif current_state == 2
    if random_number < threshold_1b
      current_state = 3;
      arrivals(3) = arrivals(3) + 1;
      total_arrivals = total_arrivals + 1;
    else
      current_state = 0;
    endif
  else
      if random_number < threshold_2_first
        arrivals(4) = arrivals(4) + 1;
        total_arrivals = total_arrivals + 1;
      elseif random_number < threshold_2_second
        current_state = 2;
```

```
      else
        current_state = 1;
      endif
   endif

endwhile

display(P(1));
display(P(2));
display(P(3));
display(P(4));

fd = fopen("simulation_results.txt", "w");
fprintf(fd, "P[0] = %d\nP[1a] = %d\nP[1b] = %d\nP[2] = %d\n",
        P(1), P(2), P(3), P(4));
fclose(fd);
```