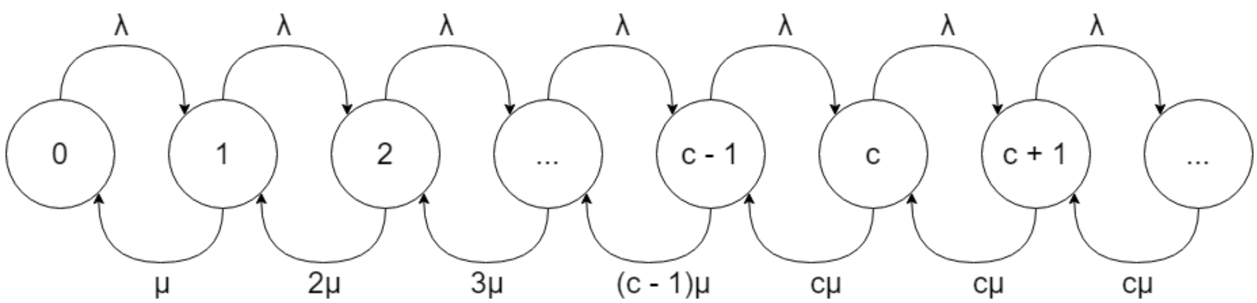
Συστήματα Αναμονής – 4η Σειρά Ασκήσεων

Γεώργιος Κυριακόπουλος – el18153

<u>Ανάλυση και Σχεδιασμός τηλεφωνικού κέντρου:</u>

1. Ακολουθεί το διάγραμμα ρυθμού μεταβάσεων του συστήματος M/M/c/c:



Για την απόδειξη του τύπου Erlang-B έχουμε:

$$k\mu P_k = \lambda P_{k-1} \Rightarrow P_k = \frac{\lambda}{k\mu} P_{k-1} \Rightarrow P_k = \frac{\rho}{k} P_{k-1}, k = 1, 2, \dots, c$$

Με την επίλυση αυτής της αναδρομικής σχέσης παίρνουμε:

$$k = 1 \rightarrow P_1 = \rho P_0$$

$$k = 2 \rightarrow P_2 = \rho P_1 = \frac{\rho^2}{2!} P_0$$

$$k = 3 \rightarrow P_3 = \rho P_2 = \frac{\rho^3}{3!} P_0$$

Βλέπουμε επομένως ότι ισχύει η παρακάτω σχέση:

$$P_k = \frac{\rho^k}{k!} P_0, k = 1, 2, \dots, c$$

Εάν χρησιμοποιήσουμε και την κανονικοποίηση έχουμε:

$$P_0 + P_1 + \cdots + P_c = 1 \Rightarrow \sum_{k=0}^{c} P_k = 1 \Rightarrow P_0 = \frac{1}{\sum_{k=0}^{c} \frac{p^k}{k!}} \Rightarrow P_{rejecting} = P_c = \frac{\rho^c}{c!} P_0$$

$$\Rightarrow P_{blocking} = \frac{\frac{\rho^c}{c!}}{\sum_{k=0}^{c} \frac{p^k}{k!}}, \rho = \frac{\lambda}{\mu}$$

Ξέρουμε, επίσης ότι ο μέσος ρυθμός απωλειών πελατών από την ουρά ισούται με $\lambda - \gamma = \lambda - \lambda(1 - P_{blocking}) = \lambda P_{blocking}$.

Για την υλοποίηση της συνάρτησης erlangb_factorial παραθέτουμε τον παρακάτω κώδικα, καθώς και την έξοδο του προγράμματος που επιβεβαιώνει την ορθότητα της συνάρτησης:

```
% Probability of blocking calculation using a custom function based o
n the Erlang-B formula.

clc;
clear all;
close all;
pkg load queueing;

function answer = erlangb_factorial(rho, c)
  num = (rho^c)/factorial(c);
  denom = 0;
  for i = 0:c
    denom += (rho^i)/factorial(i);
  endfor
  answer = num/denom;
endfunction

printf("Probability of blocking with rho = 100 and c = 20\ncalculated
 by our custom function:\n");
disp(erlangb_factorial(100,20));
printf("Probability of blocking with rho = 100 and c = 20\ncalculated
 by the erlangb function from the queueing package:\n");
disp(erlangb(100,20));

printf("Probability of blocking with rho = 200 and c = 50\ncalculated
 by our custom function:\n");
disp(erlangb_factorial(200,50));
printf("Probability of blocking with rho = 200 and c = 50\ncalculated
 by the erlangb function from the queueing package:\n");
disp(erlangb(200,50));
```

```
Probability of blocking with rho = 100 and c = 20
calculated by our custom function:
0.8024
Probability of blocking with rho = 100 and c = 20
calculated by the erlangb function from the queueing package:
0.8024
Probability of blocking with rho = 200 and c = 50
calculated by our custom function:
0.7516
Probability of blocking with rho = 200 and c = 50
calculated by the erlangb function from the queueing package:
0.7516
```

2. Με παρόμοια λογική για την υλοποίηση της συνάρτησης erlangb_iterative παραθέτουμε τον παρακάτω κώδικα, καθώς και την έξοδο του προγράμματος που επιβεβαιώνει την ορθότητα της συνάρτησης:

```
% Probability of blocking calculation using a custom function based o
n the iterative Erlang-B formula.

clc;
clear all;
close all;
pkg load queueing;

function answer = erlangb_iterative(rho, c)
  answer = 1;
  for i = 0:c
    answer = rho*answer/(rho*answer+i);
  endfor
endfunction

printf("Probability of blocking with rho = 100 and c = 20\ncalculated
 by our custom function:\n");
disp(erlangb_iterative(100,20));
printf("Probability of blocking with rho = 100 and c = 20\ncalculated
 by the erlangb function from the queueing package:\n");
disp(erlangb(100,20));

printf("Probability of blocking with rho = 200 and c = 50\ncalculated
 by our custom function:\n");
disp(erlangb_iterative(200,50));
printf("Probability of blocking with rho = 200 and c = 50\ncalculated
 by the erlangb function from the queueing package:\n");
disp(erlangb(200,50));
```

```
Probability of blocking with rho = 100 and c = 20
calculated by our custom function:
0.8024
Probability of blocking with rho = 100 and c = 20
calculated by the erlangb function from the queueing package:
0.8024
Probability of blocking with rho = 200 and c = 50
calculated by our custom function:
0.7516
Probability of blocking with rho = 200 and c = 50
calculated by the erlangb function from the queueing package:
0.7516
```

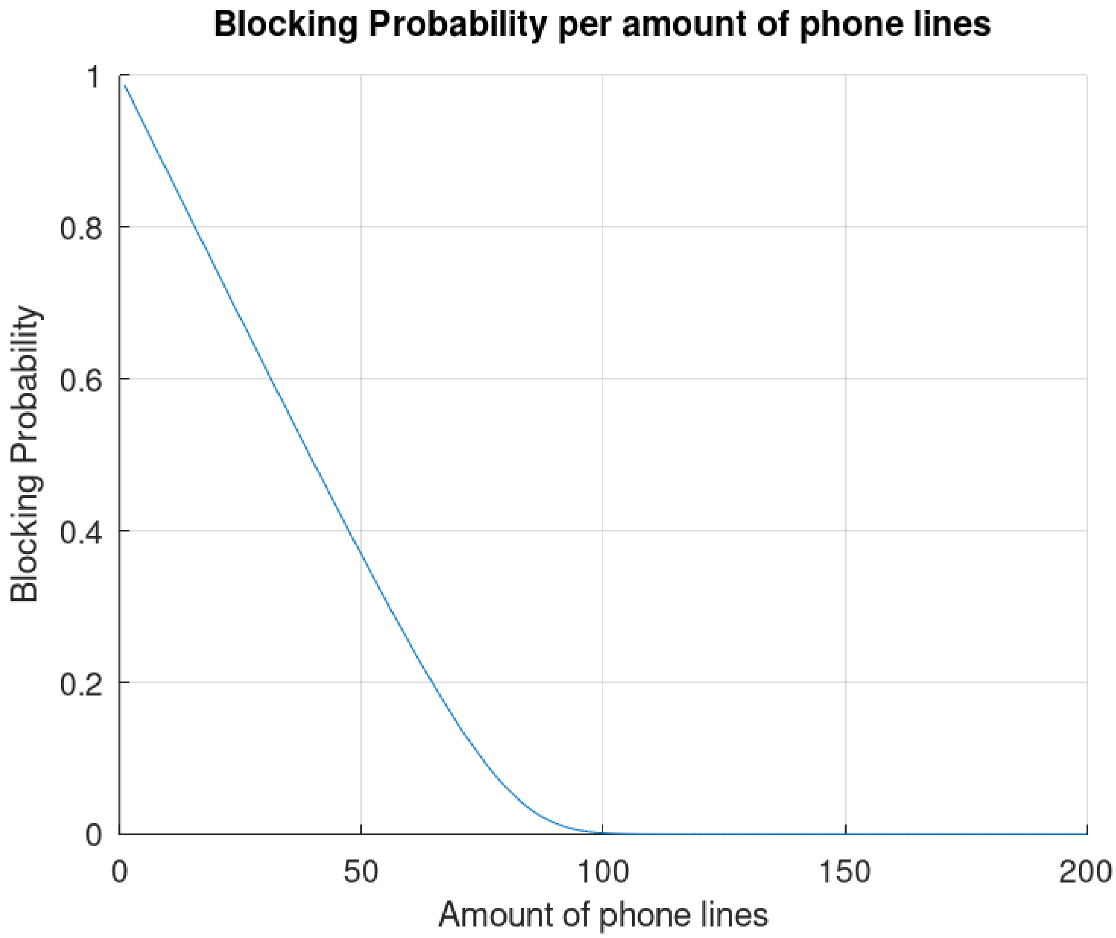
3. Ακολουθεί η έξοδος της σύγκρισης των δύο συναρτήσεων:

```
Probability of blocking with rho = 1024 and c = 1024
calculated by our erlangb_factorial function:
NaN
Probability of blocking with rho = 1024 and c = 1024
calculated by our erlangb_iterative function:
0.024524
```

Παρατηρούμε ότι η erlangb_factorial δεν βγάζει αποτέλεσμα, καθώς χρησιμοποιεί πολύ μεγάλα νούμερα (1024^{1024} και $1024!$), τα οποία δεν μπορεί να διαχειριστεί το Octave. Σε αντίθεση η erlangb_iterative, λόγω της υλοποίησης της, βγάζει αποτέλεσμα κανονικά.

4. α) Με πρότυπο τον πιο απαιτητικό χρήστη που χρησιμοποιεί στην ώρα αιχμής το τηλέφωνο του για 23 λεπτά σε μία ώρα, υπολογίζουμε τη συνολική ένταση του φορτίου ως $\rho = 200\frac{23}{60} \Rightarrow \rho = 76.667\ Erlangs.$

β) Ακολουθεί το διάγραμμα της πιθανότητας απόρριψης πελάτη από το σύστημα ως προς τον αριθμό των τηλεφωνικών γραμμών (1 έως 200), με τη χρήση της erlangb_iterative, όπως φαίνεται και με βάση τον κώδικα που παρατίθεται στο τέλος του ερωτήματος.



γ) Παρακάτω φαίνεται η έξοδο του τρόπου που υπολογίσαμε τις ελάχιστες τηλεφωνικές γραμμές που χρειαζόμαστε στο δίκτυο αυτό, ώστε η πιθανότητα απόρριψης τηλεφωνικής κλήσης να είναι μικρότερη από 1%. Αυτές βρέθηκαν ίσες με 93, λιγότερες από τις μισές συγκριτικά με τις 200 που πληρώνει η εταιρεία. Στη συνέχεια ακολουθεί και ο κώδικας που χρησιμοποιήθηκε για να εξάγουμε τα συμπεράσματα αυτά:

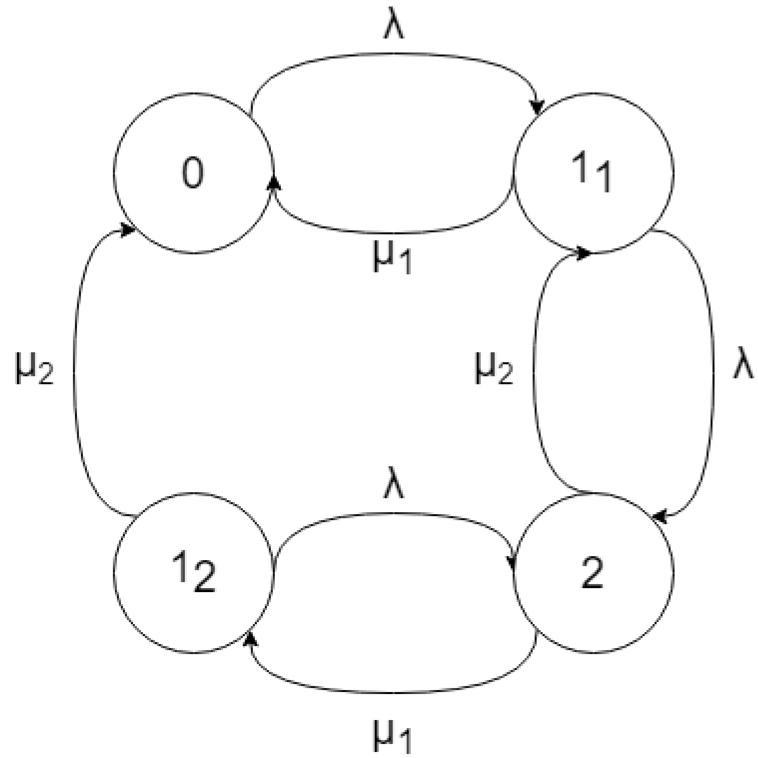
```
The minimum amount of phone lines needed in order
to have a blocking probability less than 0.01 is:
93
```

```
% Call center simulation with 200 workers and 23 minutes per hour max
 load by the most demanding worker.

clc;
clear all;
close all;
pkg load queueing;

function answer = erlangb_iterative(rho, c)
  answer = 1;
```

```
  for i = 0:c
    answer = rho*answer/(rho*answer+i);
  endfor
endfunction

rho = 200*23/60;

for i = 1:200
  pblocking(i) = erlangb_iterative(rho, i);
endfor

figure(1);
hold on;
title("Blocking Probability per amount of phone lines");
xlabel("Amount of phone lines");
ylabel("Blocking Probability");
plot(pblocking);
grid on;

printf("The minimum amount of phone lines needed in order\nto have a
blocking probability less than 0.01 is:\n");
disp(min_lines = find(pblocking < 0.01)(1))
```

Σύστημα εξυπηρέτησης με δύο ανόμοιους εξυπηρετητές:

1. Παρακάτω φαίνεται το διάγραμμα ρυθμών μεταβάσεων του συστήματος:



Μέσω των εξισώσεων ισορροπίας για το σύστημα μας και στη συνέχεια με χρήση της εξίσωσης κανονικοποίησης, λύνουμε το σύστημα μας ως εξής:

$$\mu_1 = \frac{1}{1.25} = 0.8, \mu_2 = \frac{1}{2.5} = 0.4$$

$$\lambda P_0 = \mu_1 P_{1_1} + \mu_2 P_{1_2} \Rightarrow P_0 = 0.8 P_{1_1} + 0.4 P_{1_2}$$

$$\mu_1 P_2 + \mu_2 P_2 = \lambda P_{1_1} + \lambda P_{1_2} \Rightarrow P_2 = \frac{5}{6}\left(P_{1_1} + P_{1_2}\right)$$

$$\mu_1 P_{1_1} + \lambda P_{1_1} - \lambda P_0 + \mu_2 P_2 \Rightarrow P_{1_1} = \frac{5}{9} P_0 + \frac{2}{9} P_2$$

$$\mu_2 P_{1_2} + \lambda P_{1_2} = \mu_1 P_2 \Rightarrow P_{1_2} = \frac{4}{7} P_2$$

Επομένως μετά από πολλές πράξεις μεταξύ των πάνω σχέσεων καταλήγουμε ότι:

$$P_{1_1} = 0.85937P_0$$

$$P_{1_2} = 0.78125P_0$$

$$P_2 = 1.36719P_0$$

Με χρήση της εξίσωσης της κανονικοποίησης:

$$P_0 + P_{1_1} + P_{1_2} + P_2 = 1 \Rightarrow 4.00781P_0 = 1 \Rightarrow$$

$$P_0 = 0.24951$$

$$P_{1_1} = 0.21442$$

$$P_{1_2} = 0.19493$$

$$P_2 = 0.34113$$

$$P_{blocking} = P_2 = 0.34113$$

Για το μέσο αριθμό πελατών στο σύστημα έχουμε:

$$E[n(t)] = \sum_{k=0}^{2} kP_k = P_{1_1} + P_{1_2} + 2P_2 = 1.09161$$

2. Με βάση και τον έτοιμο κώδικα στο demo4.m, αλλά και με το διάγραμμα ρυθμού μεταβάσεων ορίζουμε τα thresholds ως εξής:

$$threshold\_1a\ =\ lambda/(lambda\ +\ m1);$$
$$threshold\_1b\ =\ lambda/(lambda\ +\ m2);$$
$$threshold\_2\_first\ =\ lambda/(lambda\ +\ m1\ +\ m2);$$
$$threshold\_2\_second\ =\ (lambda\ +\ m1)/(lambda\ +\ m1\ +\ m2);$$

Ουσιαστικά, εάν βρισκόμαστε στην κατάσταση $1_1 = 1_a$, έχουμε δύο επιλογές. Είτε να πάμε στην 0 με μ_1, είτε να πάμε στην 2 με λ. Επομένως έχουμε ένα διάστημα $\lambda + \mu_1$ το οποίο με για να συμφωνεί και με τη λογική που είναι ήδη υλοποιημένη στον κώδικα το χωρίζουμε στο λ. Ομοίως, εάν βρισκόμαστε στην κατάσταση $1_2 = 1_b$, έχουμε δύο επιλογές. Είτε να πάμε στην 0 με μ_2, είτε να πάμε στην 2 με λ, οπότε χωρίζουμε το διάστημα $\lambda + \mu_2$ στο λ με παρόμοια λογική. Τέλος εάν βρισκόμαστε στη κατάσταση 2, έχουμε τρεις επιλογές και επομένως χωρίζουμε το διάστημα με βάση 2 σημεία σε 3 διαστήματα. Το διάστημα μας είναι το $\lambda + \mu_1 +$

μ_2 και το χωρίζουμε αρχικά στο λ και έπειτα στο $\lambda + \mu_1$, ώστε να έχουμε τα αντίστοιχα διαστήματα με μήκος λ, μ_1 και τέλος μ_2, για τις αντίστοιχες μεταβάσεις.

Για τα κριτήρια σύγκλισης της προσομοίωσης παρατηρούμε ότι έχουμε ένα κριτήριο, αυτό της διαφοράς του μέσου αριθμού πελατών στο σύστημα, με το σύστημα μας να συγκλίνει εάν αυτή είναι μικρότερη του 0.00001 δηλαδή του 0.001%.

Στη συνέχεια, ακολουθεί η έξοδος του κώδικα με τις πιθανότητες που υπολογίζει, όπου παρατηρούμε ότι συμφωνούν (φυσικά μέσα σε περιθώρια λάθους λόγω της τυχαιότητας) με τις πιθανότητες που υπολογίσαμε στο προηγούμενο ερώτημα (με σειρά $P_0, P_{1_1}, P_{1_2}, P_2$):

```
0.2483
0.2164
0.1937
0.3416
```

Τέλος, παρατίθεται και ο κώδικας που χρησιμοποιήθηκε για αυτό το ερώτημα:

```
% Simulation of a customer service with 2 different servers.


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 1 > 0
  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));
```