ΗΜΜΥ ΕΜΠ

6ο εξάμηνο

Συστήματα Αναμονής

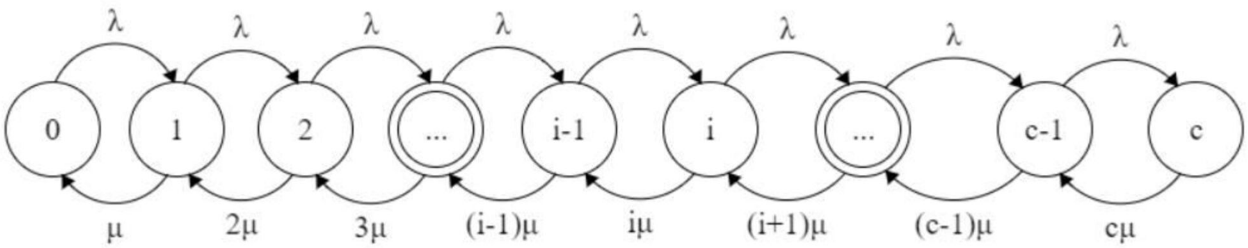
Lab 4

Ήβη Χατζή

[redacted]

Άσκηση 1: Ανάλυση και Σχεδιασμός τηλεφωνικού κέντρου

(1)



Εξισώσεις ισορροπίας:

$\lambda p_0 = \mu p_1$

$\lambda p_1 = 2\mu p_2$

...

$\lambda p_{c-1} = c\mu p_c$

Λύνοντας τες παίρνουμε $p_k = \frac{1}{k!}\left(\frac{\lambda}{\mu}\right)^k p_0 = \frac{\rho^k}{k!} p_0, \;\; k \leq c$

$p_0 + p_1 + \cdots + p_c = 1 \;\Rightarrow\; \sum_{k=1}^{c} \frac{\rho^k}{k!} p_0 = 1 \;\Rightarrow\; p_0 = \frac{1}{\sum_{k=1}^{c} \frac{\rho^k}{k!}}$

Άρα τελικά $P_{blocking} = p_c = \frac{\rho^c}{c!} \frac{1}{\sum_{k=1}^{c} \frac{\rho^k}{k!}}$

Ο μέσος ρυθμός απωλειών πελατών από την ουρά είναι $\lambda P_{blocking}$.

Υλοποιούμε τη συνάρτηση στην Octave:

```
% factorial erlang function
function res=erlangb_factorial(r,c)
  denom=0;
  for i=0:c
    denom+=(r^i)/factorial(i);
  endfor
  res=(r^c)/factorial(c)/denom;
endfunction
```

Ελέγχουμε ότι λειτουργεί σωστά:

```
erlangb_factorial(10,10)= 0.214582
erlangb(10,10)= 0.214582
```

(2) Υλοποιούμε τη συνάρτηση στην Octave:

```
% iterative erlang function
function res=erlangb_iterative(r,c)
  res=1;
  for i=0:c
    res=r*res/(r*res+i);
  endfor
endfunction
```

Ελέγχουμε ότι λειτουργεί σωστά:

```
erlangb(10,10)= 0.214582
erlangb_iterative(10,10)= 0.214582
```

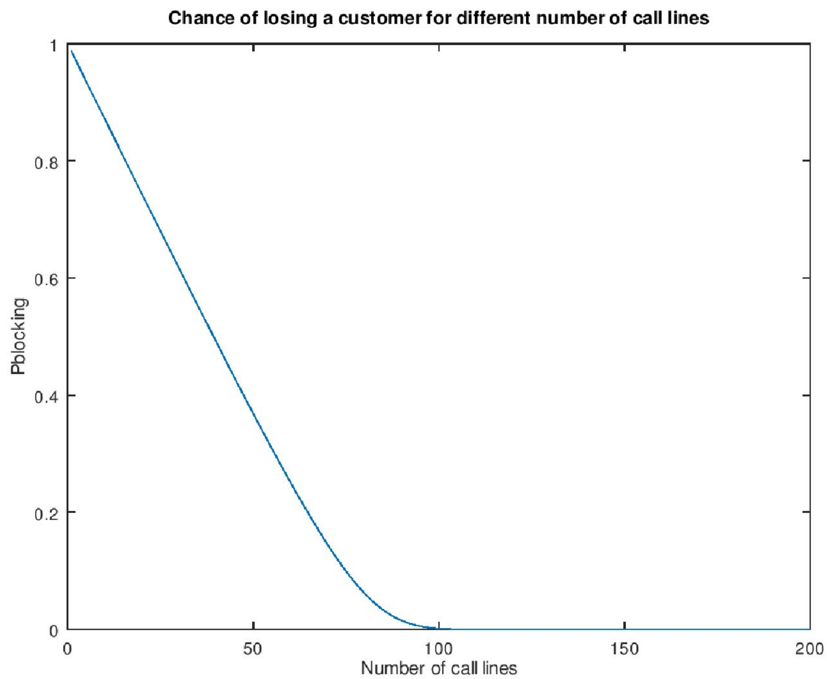
(3) Παρατηρούμε ότι ενώ ο επαναληπτικός αλγόριθμος λειτουργεί κανονικά, η συνάρτηση με τα παραγοντικά δίνει αποτέλεσμα NaN (not a number). Αυτό συμβαίνει διότι η octave αδυνατεί να διαχειριστεί μεγάλους αριθμούς όπως 1024^{1024}. Ο επαναληπτικός αλγόριθμος σε κάθε βήμα εκτελεί μια διαίρεση και κρατάει τα ενδιάμεσα αποτελέσματα σε χαμηλές τιμές (αφού είναι τα Pblocking για μικρότερα c), με αποτέλεσμα να αποφεύγει το πρόβλημα της συνάρτησης με παραγοντικό.

```
erlangb_factorial(1024,1024)= NaN
erlangb(1024,1024)= 0.0245243
erlangb_iterative(1024,1024)= 0.0245243
```

(4α) Έχουμε 200 εργαζόμενους σε 200 γραμμές άρα $\lambda = 200$, και χρησιμοποιούμε ως πρότυπο τον πιο απαιτητικό χρήστη οπότε $\frac{1}{\mu} = \frac{23}{60}$.

Επομένως έχουμε $\rho = 200\frac{23}{60} = 76.67\ Erlangs$

(4β) Προφανώς όσο περισσότερες τηλεφωνικές γραμμές έχουμε τόσο μικρότερη πιθανότητα έχει το σύστημα να είναι γεμάτο:



(4γ) Θέλουμε τον ελάχιστο αριθμό γραμμών ώστε η πιθανότητα απόρριψης να είναι μικρότερη από 1%. Βρίσκουμε από Octave ότι είναι 93:

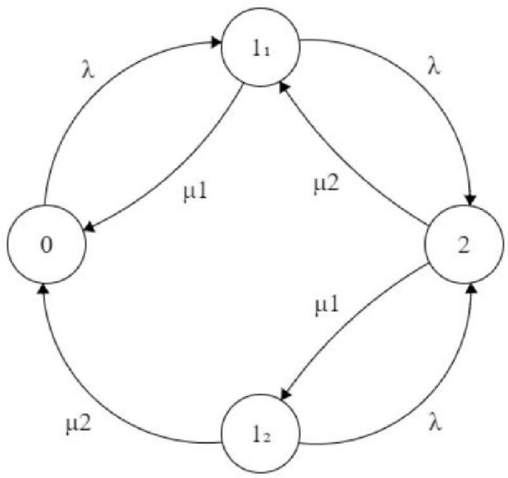
```
Fewest lines required: 93
Chance of losing a customer with 93 lines: 0.00836795
Chance of losing a customer with 92 lines: 0.0102363
```

Κώδικας για την άσκηση 1: αρχείο *ask1.m*

```
pkg load queueing
% factorial erlang function
function res=erlangb_factorial(r,c)
  denom=0;
  for i=0:c
    denom+=(r^i)/factorial(i);
  endfor
  res=(r^c)/factorial(c)/denom;
endfunction

% iterative erlang function
function res=erlangb_iterative(r,c)
  res=1;
  for i=0:c
    res=r*res/(r*res+i);
  endfor
endfunction

% check if functions work
printf("erlangb_factorial(10,10)= %d\n", erlangb_factorial(10,10));
printf("erlangb(10,10)= %d\n",erlangb(10,10));
printf("erlangb_iterative(10,10)= %d\n",erlangb_iterative(10,10));
printf("\nerlangb_factorial(1024,1024)= %d\n",
erlangb_factorial(1024,1024));
printf("erlangb(1024,1024)= %d\n",erlangb(1024,1024));
printf("erlangb_iterative(1024,1024)=
%d\n",erlangb_iterative(1024,1024));

% plot Pblocking
r=200*23/60;
c=1:200;
for i=1:200
  pblocking(i)=erlangb_iterative(r,i);
endfor
figure(1)
plot(c,pblocking);
title("Chance of losing a customer for different number of call
lines");
xlabel("Number of call lines"); ylabel("Pblocking");

% find smallest number of lines so that Pblocking<0.01
lines=1;
while pblocking(lines)>=0.01
  lines++;
endwhile
printf("\nFewest lines required: %d\n",lines);
printf("Chance of losing a customer with %d lines:
%d\n",lines,pblocking(lines));
printf("Chance of losing a customer with %d lines: %d\n",lines-
1,pblocking(lines-1));
```

Άσκηση 2: Σύστημα εξυπηρέτησης με δύο ανόμοιους εξυπηρετητές

(1) Διάγραμμα ρυθμών μεταβάσεων:



Για να πάμε στην κατάσταση 1_2 πρέπει να είμαστε στην κατάσταση 2 και να αδειάσει ο $1^{ος}$ εξυπηρετητής.

Είναι $\lambda = 1$, $\mu_1 = 0.8$, $\mu_2 = 0.4$. Εξισώσεις ισορροπίας:

$\lambda p_0 = \mu_1 p_{11} + \mu_2 p_{12}$

$(\mu_1 + \lambda) p_{11} = \lambda p_0 + \mu_2 p_2$

$(\mu_2 + \lambda) p_{12} = \mu_1 p_2$

$(\mu_1 + \mu_2) p_2 = \lambda p_{11} + \lambda p_{12}$

Συνθήκη κανονικοποίησης: $p_0 + p_{11} + p_{12} + p_2 = 1$

Η επίλυση του συστήματος εξισώσεων μας δίνει:

$p_0 = 0.25$

$p_{11} = 0.214$

$p_{12} = 0.195$

$p_2 = 0.341$

Η πιθανότητα απόρριψης πελάτη είναι $P_{blocking} = p_2 = 0.341$

Ο μέσος αριθμός πελατών στο σύστημα είναι: $E[n(t)] = 0 \cdot p_0 + p_{11} + p_{12} + 2p_2 = 1.091$

(2)

Συμπληρώνουμε τα κενά στα thresholds:

```
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);
```

Κριτήριο σύγκλισης είναι αν σε δύο διαδοχικές προσομοιώσεις ο μέσος όρος πελατών διαφέρει λιγότερο από 0.0001.

```
abs(mean_clients - previous_mean_clients) < 0.0001
```

Επιβεβαιώνουμε ότι οι πιθανότητες συμφωνούν με αυτές που υπολογίσαμε θεωρητικά:

```
 0.24945
 0.21206
 0.19602
 0.34247
```

Κώδικας για την άσκηση 2: αρχείο *ask2.m*

```
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) < 0.0001
       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));
```