ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ

ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

3η Ομάδα Ασκήσεων

Συστήματα Αναμονής (Queuing Systems)

ΔΗΜΗΤΡΙΟΣ ΓΕΩΡΓΟΥΣΗΣ

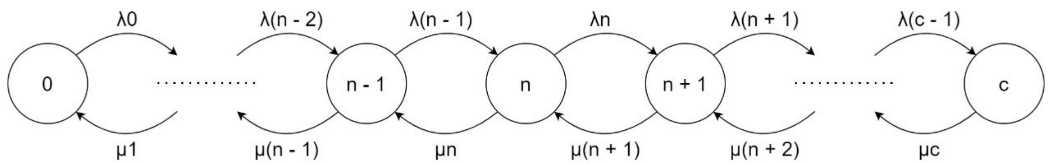
03119005

georg.dimi00@gmail.com

Ανάλυση και Σχεδιασμός τηλεφωνικού κέντρου

Στο σύστημα M/M/c/c έχουμε ότι υπάρχουν c εξυπηρετητές και μέγιστη χωρητικότητα c πελατών. Άρα έχουμε καταστάσεις συστήματος 0, 1, …, c. Οι αφίξεις ακολουθούν την Poisson με μέσο ρυθμό λ και οι εξυπηρετήσεις ακολουθούν την εκθετική κατανομή με μέσο ρυθμό μ. Ορίζουμε ρ = λ/μ.

(1)



$$\lambda_n = \lambda, n = 0, 1, \dots, c-1$$

$$\mu_n = n\mu, n = 1, 2, \dots, c$$

Χρησιμοποιούμε τις εξισώσεις ισορροπίας:

$$\lambda_{n-1}p_{n-1} = \mu_n p_n \Leftrightarrow \lambda p_{n-1} = n\mu p_n \Leftrightarrow p_n = \frac{\rho}{n}\cdot p_{n-1}, n = 1, \dots, c$$

Λύνουμε την αναδρομή:

$$\begin{cases} p_n = \dfrac{\rho}{n}p_{n-1} \\ \dfrac{\rho}{n}p_{n-1} = \dfrac{\rho}{n}\dfrac{\rho}{n-1}p_{n-2} \\ \dots \\ \dfrac{\rho^{n-2}}{n(n-2)\cdot \ldots \cdot 3}p_2 = \dfrac{\rho^{n-1}}{n(n-1)\cdot \ldots \cdot 2}p_1 \\ \dfrac{\rho^{n-1}}{n(n-2)\cdot \ldots \cdot 2}p_1 = \dfrac{\rho^{n}}{n(n-1)\cdot \ldots \cdot 1}p_0 \end{cases} \Longrightarrow (\pi\rho\acute{o}\sigma\theta\varepsilon\sigma\eta\ \kappa\alpha\tau\acute{\alpha}\ \mu\acute{\varepsilon}\lambda\eta)$$

Και παίρνουμε ότι:

$$p_n = \frac{\rho^n}{n!}p_0, n = 0, 1, \dots, c$$

Για το p0 έχουμε:

$$1 = \sum_{k=0}^{c} p_k = p_0 \sum_{k=0}^{c}\frac{\rho^k}{k!}\ \acute{\alpha}\rho\alpha\ p_0 = \left(\sum_{k=0}^{c}\frac{\rho^k}{k!}\right)^{-1}$$

Συνεπώς:

$$p_n = \frac{\frac{\rho^n}{n!}}{\sum_{k=0}^{c}\frac{\rho^k}{k!}}, n = 0, 1, \dots, c$$

Η πιθανότητα απόρριψης ενός πελάτη είναι ίδια με τη πιθανότητα το σύστημα να βρίσκεται σε κορεσμό (όλη η χωρητικότητά του είναι καλυμμένη). Άρα:

$$P_{blocking} = p_c = \frac{\frac{\rho^c}{c!}}{\sum_{k=0}^{c} \frac{\rho^k}{k!}}$$

Throughput: $\gamma = \lambda\left(1 - P_{blocking}\right)$

Ο μέσος ρυθμός απωλειών θα είναι ο ρυθμός αφίξεων πλην του throughput:

$$\lambda - \gamma = \lambda - \left(\lambda - \lambda \cdot P_{blocking}\right) = \lambda \cdot P_{blocking} = \lambda \cdot \frac{\frac{\rho^c}{c!}}{\sum_{k=0}^{c} \frac{\rho^k}{k!}}$$

Γράψαμε τον παρακάτω κώδικα:

```
clc;
clear all;
close all;

pkg load queueing;

function res = erlangb_factorial (r,c)
  sum = 0;
  for k = 0:1:c
    sum +=(power(r,k)/factorial(k));
  endfor
  res = (power(r,c)/factorial(c))/sum;
endfunction

display(["erlangb_factorial(10,10) = ",...
          num2str(erlangb_factorial(10,10))]);
display(["erlangb(10,10) = ",num2str(erlangb(10,10))]);
```

Και παίρνουμε έξοδο:

```
erlangb_factorial(10,10) = 0.21458
erlangb(10,10) = 0.21458
```

Επιβεβαιώνεται η ορθότητα της συνάρτησής μας.

(2) Κώδικας:

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

display(["erlangb_iterative(10,10) = ",...
          num2str(erlangb_iterative(10,10))]);
display(["erlangb(10,10) = ",num2str(erlangb(10,10))]);
```

Έξοδος:

```
erlangb_iterative(10,10) = 0.21458
erlangb(10,10) = 0.21458
```

Επιβεβαιώνεται η ορθότητα της συνάρτησής μας.

(3) Κώδικας:

```
display(["erlangb_factorial(1024,1024) = ",...
         num2str(erlangb_factorial(1024,1024))]);
display(["erlangb_iterative(1024,1024) = ",...
         num2str(erlangb_iterative(1024,1024))]);
display(["erlangb(1024,1024) = ",num2str(erlangb(1024,1024))]);
display("");
```

Έξοδος:

```
erlangb_factorial(1024,1024) = NaN
erlangb_iterative(1024,1024) = 0.024524
erlangb(1024,1024) = 0.024524
```

Βλέπουμε ότι η factorial επιστρέφει Not a Number, ενώ η iterative επιστρέφει το σωστό/αναμενόμενο αποτέλεσμα. Αυτό συμβαίνει, διότι στη factorial πρέπει να υπολογιστούν πολύ μεγάλα παραγοντικά (όλα τα παραγοντικά των αριθμών 0, 1, …, 1024).

(4)

(α) Προσδιορισμός έντασης φορτίου: Χ Erlangs αντιπροσωπεύουν το φόρτο κυκλοφορίας που εξυπηρετείται από έναν εξυπηρετητή που απασχολείται 100Χ% του χρόνου. Στη δική μας περίπτωση ένας εξυπηρετητής απασχολείται για 23 λεπτά κάθε ώρας άρα το για εμάς Χ = 23/60. Έχουμε συνολικά 200 χρήστες άρα
ρ = 200 * 23/60 = 76.67 Erlangs.

(β)

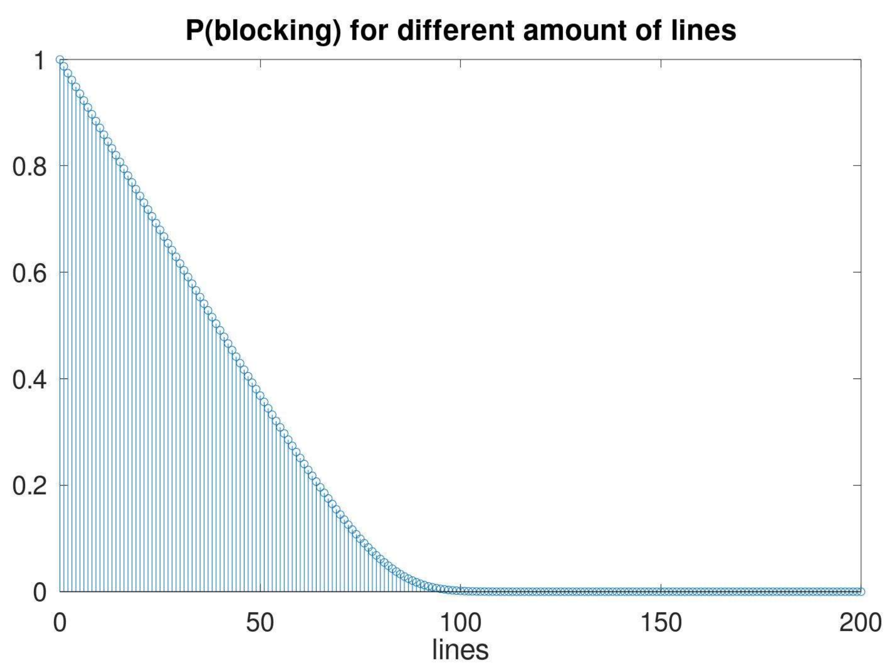
$$P_{blocking} = \frac{\frac{\rho^c}{c!}}{\sum_{k=0}^{c} \frac{\rho^k}{k!}}$$

Έχουμε ήδη προσδιορίσει το ρ του συστήματός μας, θα υπολογίσουμε τη παραπάνω πιθανότητα για c = 1, 2, …, 200.

Κώδικας:

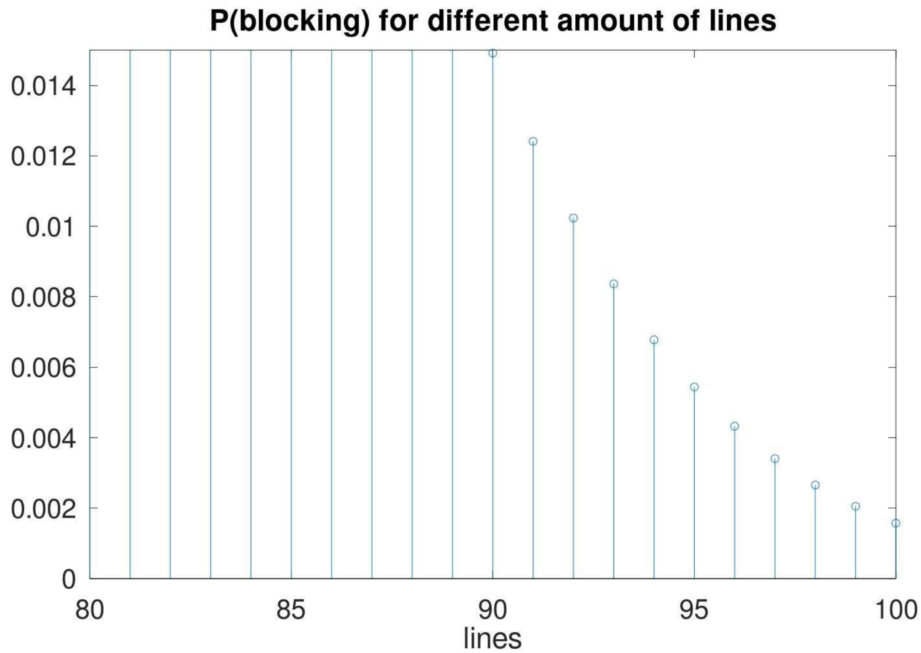
```
function res = erlangb_iterative_graph (r,c)
  res(1) = 1;
  flag = 1; %used to find the lines we need for this problem
  for n = 2:1:(c + 1)
    res(n) = (r*res(n - 1))/(r*res(n-1) + (n - 1));
    if(res(n) < 0.01 && flag == 1)
      display(["We need ",num2str(n-1)," lines"]);
      flag = 0;
    endif
  endfor
endfunction

figure(1);
stem((0:1:200),erlangb_iterative_graph(200*23/60,200));
axis([80 100 0 0.015]);
title("P(blocking) for different amount of lines");
xlabel("lines");
set(gca, "fontsize", 20);
```

Έξοδος:



(γ)

Κάνουμε μια σμίκρυνση στη περιοχή τιμών κοντά στο 0.01:



Το πρόγραμμα μας δίνει επίσης έξοδο:

```
We need 93 lines
```

Το οποίο επαληθεύεται και από το δεύτερο διάγραμμα που δείξαμε παραπάνω.

Θα χρειαστούμε, λοιπόν, 93 τηλεφωνικές γραμμές για να έχουμε πιθανότητα απόρριψης μικρότερη του 0.01.

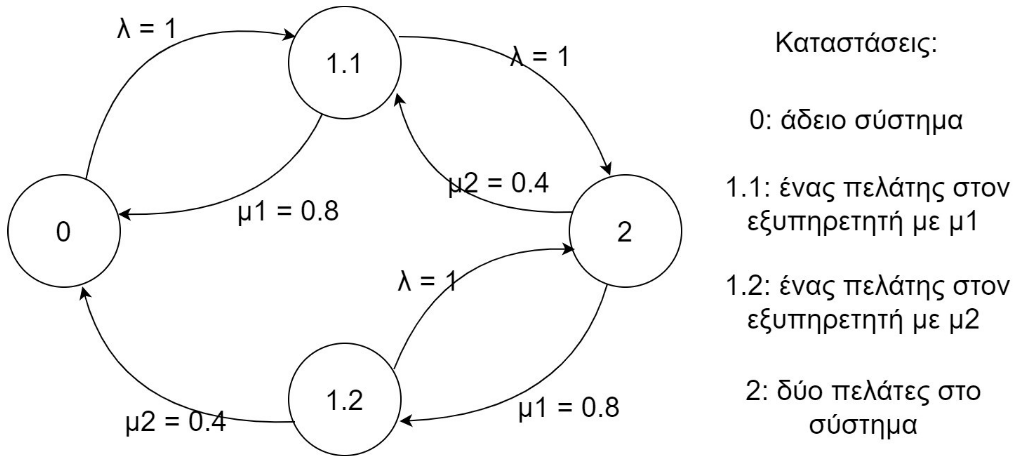
Τέλος, δείχνουμε και τον συνολικό κώδικα του προγράμματος της άσκησης.

```
clc;
clear all;
close all;

pkg load queueing;

function res = erlangb_factorial (r,c)
  sum = 0;
  for k = 0:1:c
    sum +=(power(r,k)/factorial(k));
  endfor
  res = (power(r,c)/factorial(c))/sum;
endfunction

display(["erlangb_factorial(10,10) = ",...
           num2str(erlangb_factorial(10,10))]);
display(["erlangb(10,10) = ",num2str(erlangb(10,10))]);
display("");

function res = erlangb_iterative (r,c)
  res = 1;
  for n = 0:1:c
    res = (r*res)/(r*res + n);
  endfor
endfunction

display(["erlangb_iterative(10,10) = ",...
           num2str(erlangb_iterative(10,10))]);
display(["erlangb(10,10) = ",num2str(erlangb(10,10))]);
display("");

display(["erlangb_factorial(1024,1024) = ",...
           num2str(erlangb_factorial(1024,1024))]);
display(["erlangb_iterative(1024,1024) = ",...
           num2str(erlangb_iterative(1024,1024))]);
display(["erlangb(1024,1024) = ",num2str(erlangb(1024,1024))]);
display("");

%erlangb_iterative calculates at each step the P(blocking)
%for the system M/M/n/n, so we will slightly change the above
%function to save the results

function res = erlangb_iterative_graph (r,c)
  res(1) = 1;
  flag = 1; %used to find the lines we need for this problem
  for n = 2:1:(c + 1)
    res(n) = (r*res(n - 1))/(r*res(n-1) + (n - 1));
    if(res(n) < 0.01 && flag == 1)
      display(["We need ",num2str(n-1)," lines"]);
      flag = 0;
    endif
  endfor
endfunction

figure(1);
stem((0:1:200),erlangb_iterative_graph(200*23/60,200));
%axis([80 100 0 0.015]);
title("P(blocking) for different amount of lines");
xlabel("lines");
set(gca, "fontsize", 20);
```

Σύστημα εξυπηρέτησης με δύο ανόμοιους εξυπηρετητές

λ = 1 πελάτης/sec, μ1 = 0.8 πελάτες/sec, μ2 = 0.4 πελάτες/sec

(1)



(α)

Χρησιμοποιώντας τη λογική της εισερχόμενης – εξερχόμενης ροής από κάθε κατάσταση έχουμε:

$$p_0 \cdot 1 = p_{11} \cdot 0.8 + p_{12} \cdot 0.4 \ (1)$$

$$p_{11} \cdot 1.8 = p_0 \cdot 1 + p_2 \cdot 0.4 \ (2)$$

$$p_{12} \cdot 1.4 = p_2 \cdot 0.8 \qquad (3)$$

Για τη κατάσταση 2 δεν χρειάζεται να γράψουμε την εξίσωση γιατί είναι γραμμικός συνδυασμός των παραπάνω. Η τέταρτη εξίσωση που χρειαζόμαστε είναι η εξίσωση κανονικοποίησης των πιθανοτήτων:

$$p_0 + p_{11} + p_{12} + p_2 = 1 \ (4)$$

Από τις (2) και (3) έχω: $p_0 = 1.8p_{11} - 0.7p_{12}$ (5)

Από (1) και (5) έχω: $p_{11} = 1.1p_{12}$ (6)

Άρα:

$$p_{11} = 1.1p_{12}$$

$$p_2 = 1.75p_{12}$$

$$p_0 = 1.28p_{12}$$

$$(4): 1 = p_0 + p_{11} + p_{12} + p_2 = (1.28 + 1.1 + 1 + 1.75)p_{12}$$

Άρα: $p_{12} \cdot (5.13) = 1 \Longrightarrow p_{12} = \frac{100}{51}$

$$\begin{cases} p_0 = \dfrac{128}{513} = 0.24951 \\ p_{11} = \dfrac{110}{513} = 0.21442 \\ p_{12} = \dfrac{100}{513} = 0.19493 \\ p_2 = \dfrac{175}{513} = 0.34113 \end{cases}$$

(β) Η πιθανότητα απόρριψης πελάτη από το σύστημα είναι η $p_2 = 0.34113$.

(γ) $E(n) = 0 \cdot p_0 + 1 \cdot (p_{11} + p_{12}) + 2 \cdot p_2 = 1.0916\ \pi\varepsilon\lambda\acute{\alpha}\tau\varepsilon\varsigma$

(2) Κώδικας:

```
clc;
clear all;
close all;


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

%states 0, 1, 2, 3
%
%state 0: nobody is in the system
%
%state 1: 1 person is in the system and is processed by
%server with m1
%
%state 2: 1 person is in the system and is processed by
%server with m2
%
%state 3: 2 people are in the system

%arrival at state 1
threshold_1a = lambda/(lambda + m1);

%arrival at state 2
threshold_1b = lambda/(lambda + m2);

%arrival at state 3
threshold_2_first = lambda/(lambda + m1 + m2);

%leaving state 3, but returning to state 2, so the server with m1
%is finished, thus we add m1 in the nominator
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
       display(["The mean number of clients in the system is "...
       ,num2str(previous_mean_clients)]);
       display("");
       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
```

```
for i=0:1:3
  display(["P(",num2str(i),") = ",num2str(P(i+1))]);
endfor
display("");
display(["P(blocking) = ",num2str(P(4))]);
```

(α) Για τα thresholds έχουμε τα εξής:

Αρχικά πρέπει να καταλάβουμε σε ποια κατάσταση αναφέρεται κάθε μία από τις 0, 1, 2, 3.

Μελετώντας τον κώδικα μπορούμε να κάνουμε την εξής αντιστοιχία καταστάσεων:

Καταστάσεις σχεδίου μας (στην αρχή)	Καταστάσεις κώδικα
0	0
1.1	1
1.2	2
2	3

Παρακάτω όταν αναφερόμαστε σε καταστάσεις χρησιμοποιούμε τα ονόματα του κώδικα.

threshold_1a: Συμβολίζει άφιξη όταν είμαστε στη κατάσταση 1 άρα είναι $\lambda/(\lambda + \mu1)$

threshold_1b: Συμβολίζει άφιξη όταν είμαστε στη κατάσταση 2 άρα είναι $\lambda/(\lambda + \mu2)$

threshold_2_first: Συμβολίζει άφιξη όταν είμαστε στη κατάσταση 3 άρα είναι $\lambda/(\lambda + \mu1 + \mu2)$

threshold_2_second: Συμβολίζει αναχώρηση από τη κατάσταση 3 προς τη κατάσταση 2, η οποία γίνεται αν τελειώσει ο εξυπηρετητής με μ1 άρα είναι $(\lambda + \mu1)/(\lambda + \mu1 + \mu2)$

Ο παρονομαστής των threshold_2 έχει και το μ1 και το μ2 αφού μπορούμε να φύγουμε από αυτή τη κατάσταση και με τους 2 τρόπους.

(α) Τα κενά του προγράμματος φαίνονται και στον κώδικα αλλά και στην αμέσως προηγούμενη ανάλυση ότι συμπληρώθηκαν και με ποιον τρόπο το κάναμε.

(β) Τα κριτήρια σύγκλισης της προσομοίωσής μας είναι η διαφορά μεταξύ δύο διαδοχικών μέσων αριθμών πελατών (ανά 1000 βήματα) να είναι μικρότερη από 0.001%.

Δείχνουμε την έξοδο του παραπάνω προγράμματος σε κάποια εκτέλεσή του:

```
The mean number of clients in the system is 1.0912

P(0) = 0.25005
P(1) = 0.21454
P(2) = 0.19415
P(3) = 0.34126

P(blocking) = 0.34126
```

Παραπάνω υπολογίζονται όλα τα ζητούμενα του ερωτήματος (1).

Αντιπαραθέτουμε τους θεωρητικούς μας υπολογισμούς από πριν με αυτά τα αποτελέσματα στον παρακάτω πίνακα.

	Θεωρητικά	MatLab
Μέσος	1.0916	1.0912
P(0)	0.24951	0.25005
P(1)	0.21442	0.21454
P(2)	0.19493	0.19415
P(3)	0.34113	0.34126

Οι αριθμοί της προσομοίωσης είναι πολύ κοντά σε αυτούς που υπολογίσαμε προηγουμένως, αλλά παρουσιάζονται μικρές αποκλίσεις, που είναι αναμενόμενο.

Η συνθήκη ήταν |μ.ο.(τωρινού σταδίου) – μ.ο.(προηγούμενου σταδίου)| < 0.00001.