



UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO

Trabalho de Modelagem e Simulação

Prática 03

Alunos: | Aline de Souza Lima Abreu
| Miguel Henrique de Brito Pereira
| Tarcísio Magno de Almeida Filho
| Vinícius Gonzaga Rocha

Uberlândia
2017

Sumário

Exercício 3.1.1	3
Exercício 3.1.2	5
Exercício 3.1.3	6
Exercício 3.1.4	7
Exercício 3.1.5	8
Exercício 3.1.6	11
Exercício 3.1.7	13
Exercício 3.2.3	14
Exercício 3.2.4	16

Lista de Figuras

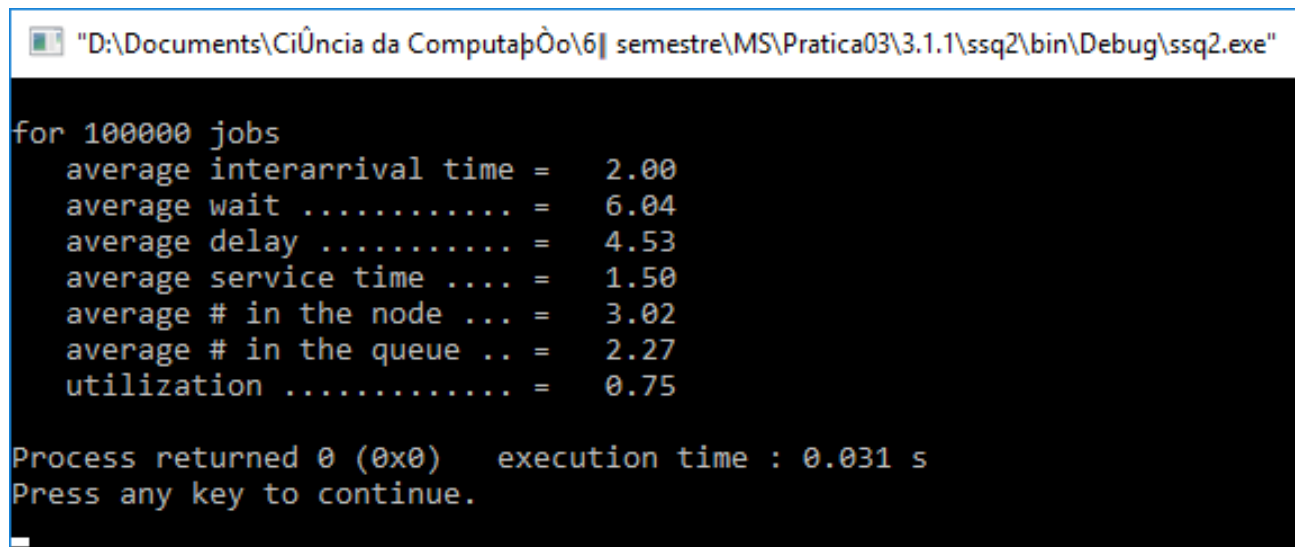
1	Output referente ao número de jobs 100,000	4
2	Estatísticas de estado estável do exemplo 3.1.3	5
3	Output Exemplo 3.1.3 com alterações	8
4	Verificando o output percebe-se que a média é 1.5	9
5	Convergência das steady-state statistics	10
6	Steady-state statistics do exemplo 3.1.4	10
7	Aproximação para 4.27 do tempo médio de delay	10
8	Curvas para $s = 15, 16, \dots, 35$	12
9	Output de 500 execuções de sis2 modificado para usar Equilikely(5,25) 2 vezes .	14
10	Steady-state statistics do exemplo 3.1.6	14
11	Estatística para a mudança proposta no exercício	15
12	Estatística para a mudança proposta no exemplo	16
13	Output referente à type 0 e type 1	17

Exercício 3.1.1

(a) Modify program ssq2 to use Exponential (1.5) service times.

Ver código.

(b) Process a relatively large number of jobs, say 100 000, and determine what changes this produces relative to the statistics in Example 3.1.3?



```
"D:\Documents\Ciência da Computação\6º semestre\MS\Pratica03\3.1.1\ssq2\bin\Debug\ssq2.exe"

for 100000 jobs
  average interarrival time = 2.00
  average wait ..... = 6.04
  average delay ..... = 4.53
  average service time .... = 1.50
  average # in the node ... = 3.02
  average # in the queue .. = 2.27
  utilization ..... = 0.75

Process returned 0 (0x0) execution time : 0.031 s
Press any key to continue.
```

Figura 1: Output referente ao número de jobs 100,000

Estatísticas de estado estável do exemplo 3.1.3:

\bar{r}	\bar{w}	\bar{d}	\bar{s}	\bar{l}	\bar{q}	\bar{x}
2.00	3.83	2.33	1.50	1.92	1.17	0.75

Figura 2: Estatísticas de estado estável do exemplo 3.1.3

As estatísticas que mais sofreram alterações foram as estatísticas relacionadas ao tempo de espera e delay, como consequência também alteraram o tempo na fila e no nó de serviço.

(c) Explain (or conjecture) why some statistics change and others do not.

O tempo de serviço não muda, pois o limite da soma dos valores gerados no intervalo Uniform(1.0, 2.0) tende a 1.5, da mesma forma que a função Exponential(1.5) tende a 1.5.

O delay altera, pois quando se gerava o valor uniforme tínhamos os valores gerados de 1 até 2 para o tempo de serviço, enquanto se gera com o exponencial podemos ter valores muito altos. Isso aumenta o desvio padrão de espera, logo alguns jobs devem esperar muito mais tempo na fila, causando variação do delay, como consequência aumenta-se o tempo de espera, delay, tempo de espera na fila e no nó de serviço.

Exercício 3.1.2

(a) Relative to the steady-state statistics in Example 3.1.3 and the statistical equations in Section 1.2, list all of the consistency checks that should be applicable.

As verificações de consistência óbvias que podem ser aplicadas são $\bar{w}=\bar{d}+\bar{s}$ e $\bar{l}=\bar{q}+\bar{x}$.

(b) Verify that all of these consistency checks are valid.

$$\bar{w}=\bar{d}+\bar{s} = \bar{w} = 2.33 + 1.5 = 3.83$$

$$\bar{l}=\bar{q}+\bar{x} = \bar{l} = 1.17 + 0.75 = 1.92$$

Os valores de \bar{w} e \bar{l} coincidem com os resultados obtidos em ssq2.

Exercício 3.1.3

(a) Given that the Lehmer random number generator used in the library rng has a modulus of 231, what are the largest and smallest possible numerical values (as a function of μ) that the function *Exponential*(μ) can return?

Resposta: O menor número que a Função exponencial pode gerar é 0 e o maior é teoricamente infinito (∞), no caso o modelo computacional será limitado pela precisão de quão próximo de 1 a representação de ponto flutuante permite alcançar.

(b) Comment on this relative to the theoretical expectation that an Exponential (μ) random variate can have an arbitrarily large value and a value arbitrarily close to zero.

Resposta: Os valores gerados pela função exponencial tem domínio no intervalo $[0, \infty]$, permitindo valores grandes ou próximos de zero onde que o limite da média dos valores gerados se aproxima de μ , Logo, os valores podem ser arbitrariamente grandes ou pequenos, desde que a soma desses com sucessivos valores gerados pela função *exponencial*(μ) tendam a μ .

Exercício 3.1.4

(a) Conduct a transition-to-steady-state study like that in Example 3.1.3 except for a service time model that is $Uniform(1.3, 2.3)$. Be specific about the number of jobs that seem to be required to produce steady-state statistics.

O steady state encontrado foi no job 3332180.

```
for 3332180 jobs
  average interarrival time = 2.00
  average wait ..... = 10.11
  average delay ..... = 8.31
  average service time .... = 1.80
  average # in the node ... = 5.06
  average # in the queue .. = 4.16
  utilization ..... = 0.90

Process returned 0 (0x0)  execution time : 0.448 s
Press any key to continue.
```

Figura 3: Output Exemplo 3.1.3 com alterações

(b) Comment.

No exemplo 3.1.3 foram necessários 1207560 jobs para se obter o steady state de todas as estatísticas, depois desse valor há pequenas alterações no valor \bar{w} , pois temos $d.dd$. Já para o exercício 3.1.4, utilizamos $Uniform(1.3, 2.3)$, foram necessários 3332180 jobs para se obter o steady state de todas as estatísticas, várias estatísticas antes desse valor chega ao valor estacionário, porém o que demorou mais foi o \bar{w} , como no livro já havia dito.

Exercício 3.1.5

(a) Verify that the mean service time in Example 3.1.4 is 1.5.

Adicionando a função `Geometric()` ao código do `ssq2` e modificando a função `GetService` assim como no exemplo 3.1.4 obtém-se um tempo médio de serviço de 1.50, após 500 execuções do programa.

```
Executing the modified ssq2 500 times and taking the means of the outputs the results are,
for 10000 jobs
    average service time .... =    1.50

Process returned 0 (0x0)    execution time : 559.959 s
Press any key to continue.
_
```

Figura 4: Verificando o output percebe-se que a média é 1.5

(b) Verify that the steady-state statistics in Example 3.1.4 seem to be correct.

Após executar o ssq2 modificado 500 vezes, foi calculada a média para cada um dos outputs gerados, e podemos verificar que os valores parecem convergir para os valores fornecidos no exemplo 3.1.4. Para fins de ilustração, um gráfico da variável \bar{d} (average delay) foi gerado, para que fosse possível observar como os valores médios vão convergindo para 4.27 de acordo com o crescimento do número de execuções.

```
Executing the modified ssq2 500 times and taking the means of the outputs the results are,
for 10000 jobs
average interarrival time = 2.00
average wait ..... = 5.78
average delay ..... = 4.28
average service time .... = 1.50
average # in the node ... = 2.89
average # in the queue .. = 2.14
utilization ..... = 0.75

Process returned 0 (0x0)   execution time : 558.938 s
Press any key to continue.
```

Figura 5: Convergência das steady-state statistics

\bar{r}	\bar{w}	\bar{d}	\bar{s}	\bar{l}	\bar{q}	\bar{x}
2.00	5.77	4.27	1.50	2.89	2.14	0.75

Figura 6: Steady-state statistics do exemplo 3.1.4

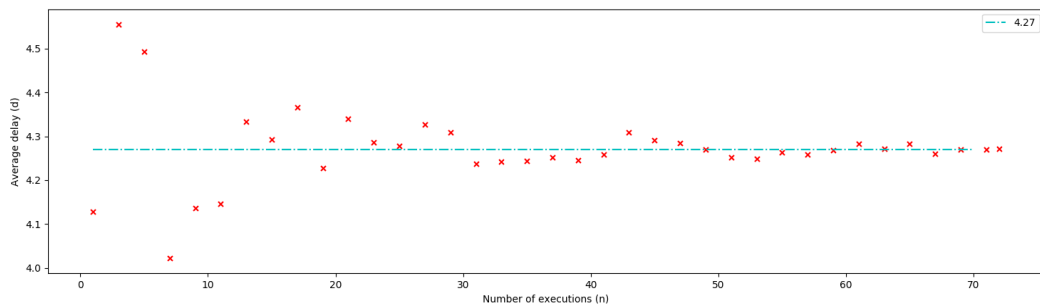


Figura 7: Aproximação para 4.27 do tempo médio de delay

(c) Note that the arrival rate, service rate, and utilization are the same as those in Example 3.1.3, yet all the other statistics are larger than those in Example 3.1.3. Explain (or conjecture) why this is so. Be specific.

Isso ocorre devido ao cálculo de `GetService()` envolver agora o uso de uma variável aleatória geométrica, obtida junto a função `Geometric(p)`, sendo p , no exemplo 0.9, enquanto no exemplo 3.1.3 `GetService()` faz o simples uso de uma variável aleatória uniforme. O uso de uma variável geométrica se faz necessário uma vez que deixa o modelo mais robusto e realístico. No entanto, essa robustez traz consigo um aumento do tempo de espera e consequentemente o delay, e o número de nós no nó de serviço. Como descrito no livro, o parâmetro p está diretamente relacionado a média da `Geometric(p)`, sendo esta $p/(1-p)$. Isso implica que valores de p próximos 0.0 trarão a média para próximo de 0.0, enquanto valores próximos de 1.0 farão a média ser grande. No exemplo, o valor utilizado 0.9 favorece o surgimento de valores altos.

Exercício 3.1.6

(a) Modify program `sis2` to compute data like that in Example 3.1.7. Use the functions `PutSeed` and `GetSeed` from the library `rng` in such a way that one initial seed is supplied by the system clock, printed as part of the program's output and used automatically to generate the same demand sequence for all values of s

A modificações no código do `Sis2` envolvem a adição de uma chamada `PutSeed(-1)`, para que o programa retire uma semente do relógio e logo em seguida uma chamada de `GetSeed` para obter a semente atual e armazena-la em uma variável. Esta semente será usada para todos os valores crescente de s . Além disso, outra modificação realizada no código para que ele computasse os dados como no exemplo foi o cálculo do dependent cost. Este cálculo foi feito ao final de `Sis2`, e ao invés de se printar `setup`, `holding` e `shortage`, foi printado o `Dependet cost`, tendo sido este calculado utilizando-se os mesmos custos do exemplo 3.1.7, (\$ 1000 para `setup`, \$ 25 para `holding` e \$ 700 para `shortage`).

(b) For $s = 15, 16, \dots, 35$ create a figure (or table) similar to the one in Example 3.1.7

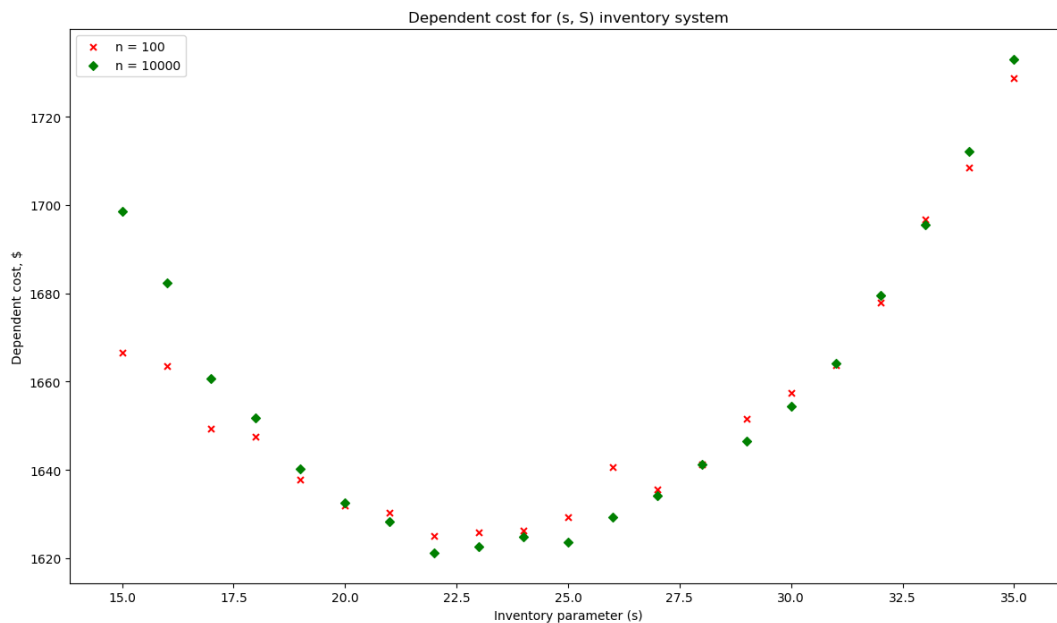


Figura 8: Curvas para $s = 15, 16, \dots, 35$

(c) Comment

Como a mesma semente foi utilizada para todos os valores de s , tendo sido retirada apenas ao início do programa, a mesma sequência de demandas foi gerada. Portanto, qualquer mudança no sistema, foi ocasionada pelo valor variante de s . Todas as suposições acerca do modelo devem se manter constantes por várias semanas, e por esta razão as estatísticas de estado estável não são muito relevantes para este exemplo. É altamente improvável que a distribuição da demanda, o custo dos parâmetros e o inventário se mantenham os mesmos por 2 anos. Por essa razão, as médias baseadas em $n = 1000$ foram calculadas também, e produzem uma sequência muito mais suave com s , e por isso o valor ótimo fica mais evidenciado em aproximadamente 22.

Exercício 3.1.7

(a) Relative to Example 3.1.5, if instead the random variate sequence of demands are generated as

$\bar{d} = \text{Equilikely}(5, 25) + \text{Equilikely}(5, 25)$ for $i = 1, 2, 3, \dots$

then, when compared with those in Example 3.1.6, demonstrate that some of the steady-state statistics will be the same and others will not.

Modificando a função GetDemand as estatísticas obtidas para 500 execuções foram:

```
Executing the modified sis2 500 times and taking the means of the outputs the results are,
for 100000 time intervals, with an average demand of 30.00
and policy parameters (s, S) = (20, 80)

average order ..... = 30.00
setup frequency ..... = 0.39
average holding level .... = 42.61
average shortage level ... = 0.18

Process returned 0 (0x0)   execution time : 552.913 s
Press any key to continue.
```

Figura 9: Output de 500 execuções de sis2 modificado para usar Equilikely(5,25) 2 vezes

e as estatísticas de estado estável do exemplo 3.1.6 são:

\bar{d}	\bar{o}	\bar{u}	\bar{l}^+	\bar{l}^-
30.00	30.00	0.39	42.86	0.26

Figura 10: Steady-state statistics do exemplo 3.1.6

é possível observar que \bar{d} , \bar{o} e \bar{u} se mantiveram semelhantes, enquanto \bar{l}^+ e \bar{l}^- sofreram declínio.

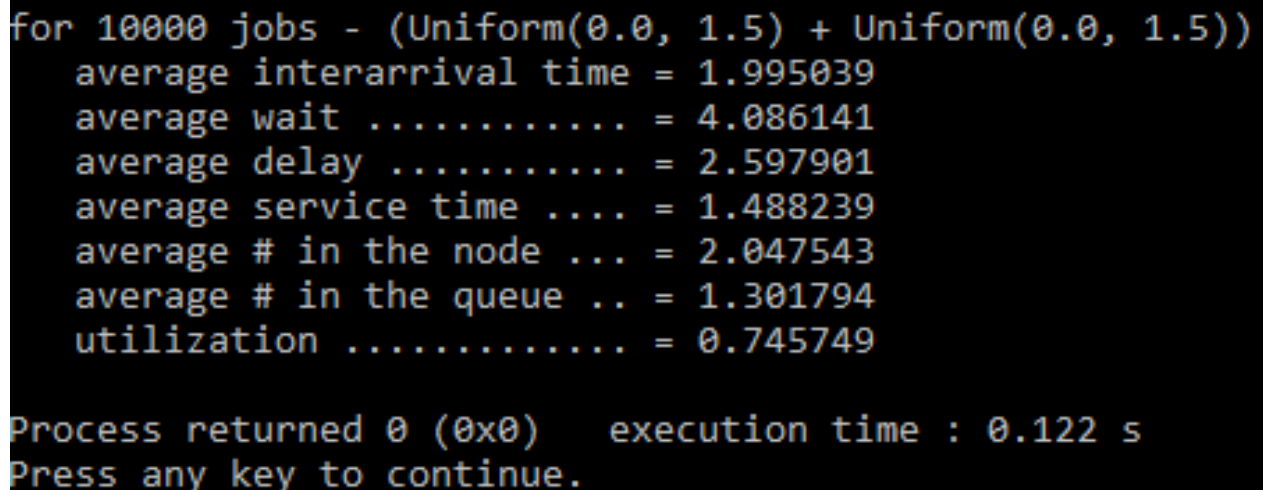
(b) Explain why this is so.

Como explicado no livro, os parâmetros a e b passados para Equilikely() produzirão uma demanda média por intervalo de tempo igual a $(a/b)/2$. No caso de usarmos Equilikely(5,25) + Equilikely(5,25), haverá demanda média de $15 + 15 = 30$, o mesmo que Equilikely(10,50). Por este motivo, \bar{d} , \bar{o} e \bar{u} se mantiveram com valores muito próximos dos valores do exemplo 3.1.6.

Exercício 3.2.3

Modify program ssq2 as suggested in Example 3.2.7 to create two programs that differ only in the function `GetService`. For one of these programs, use the function as implemented in Example 3.2.7; for the other program, use:

```
double GetService(void) {  
    SelectStream(2); /* this line is new */  
    return (Uniform(0.0, 1.5) + Uniform(0.0, 1.5));  
}
```



```
for 10000 jobs - (Uniform(0.0, 1.5) + Uniform(0.0, 1.5))  
    average interarrival time = 1.995039  
    average wait ..... = 4.086141  
    average delay ..... = 2.597901  
    average service time .... = 1.488239  
    average # in the node ... = 2.047543  
    average # in the queue .. = 1.301794  
    utilization ..... = 0.745749  
  
Process returned 0 (0x0)    execution time : 0.122 s  
Press any key to continue.
```

Figura 11: Estatística para a mudança proposta no exercício

(a) For both programs verify that exactly the same average interarrival time is produced (print the average with d:dddddd precision). Note that the average service time is approximately the same in both cases, as is the utilization, yet the service nodes statistics \bar{w} , \bar{d} , \bar{l} and \bar{q} are different.

(b): *Why?*

Resposta: O valor da média \bar{r} é o mesmo pois ela é dependente apenas da função `getArrival`, que é semelhante em ambos os programas. O tempo médio de serviço (\bar{s}) é aproximadamente o mesmo devido ao cálculo da média. O método do exemplo gera números uniformemente distribuídos no intervalo (1.0,2.0), enquanto o do exercício gera pela soma dos intervalos (0.0,1.5) e (0.0,1.5). Sabemos que na distribuição uniforme o limite da média dos números gerados tende a média do intervalo. Assim temos:

$$svg(1.0, 2.0) = 1.5$$

$$avg(0.0, 1.5) + avg(0.0, 1.5) = 0.75 + 0.75 = 1.5$$

Logo, as médias de tempo de serviço são iguais. As outras estatísticas são diferentes devido

```
for 10000 jobs - Uniform(1.0, 2.0)
  average interarrival time = 1.995039
  average wait ..... = 3.861483
  average delay ..... = 2.364763
  average service time .... = 1.496721
  average # in the node ... = 1.935415
  average # in the queue .. = 1.185243
  utilization ..... = 0.750172

Process returned 0 (0x0)   execution time : 0.163 s
Press any key to continue.
```

Figura 12: Estatística para a mudança proposta no exemplo

a forma como os tempos de serviço são gerados. O modo de geração $\text{Uniform}(0.0, 1.5) + \text{Uniform}(0.0, 1.5)$ possui uma menor probabilidade de resultar em um tempo muito próximo a zero, fazendo com que a chance de espera na fila seja menor, ou seja o tempo de delay (\bar{d}) acaba sendo um pouco maior, e logo, todos os dependentes desse também se tornam maior.

Exercício 3.2.4

Modify program ssq2 as suggested in Examples 3.2.8 and 3.2.9.

(a) What proportion of processed jobs are type 0?

Foi usado 10000 jobs, a proporção de jobs type 0:
 $6039 / 10000 = 0,60$

(b) What are \bar{w} , \bar{d} , \bar{s} , \bar{l} , \bar{q} , and \bar{x} for each job type?

```
for 6039 jobs
  average interarrival time = 3.94
  average wait ..... = 7.70
  average delay ..... = 5.72
  average service time .... = 1.98
  average # in the node ... = 1.95
  average # in the queue .. = 1.45
  utilization ..... = 0.50

for 4010 jobs
  average interarrival time = 5.94
  average wait ..... = 7.70
  average delay ..... = 5.72
  average service time .... = 1.98
  average # in the node ... = 1.30
  average # in the queue .. = 0.96
  utilization ..... = 0.33

Process returned 0 (0x0)   execution time : 0.009 s
Press any key to continue.
```

Figura 13: Output referente à type 0 e type 1

(c) What did you do to convince yourself that your results are valid?
 Complementar as verificações $\bar{w}=\bar{d}+\bar{s}$ e $\bar{l}=\bar{q}+\bar{x} = \bar{l}$

- $\bar{w}=\bar{d}+\bar{s} = \bar{w} = 5.72 + 1.98 = 7.70$

- Type 0
 $\bar{l}=\bar{q}+\bar{x} = \bar{l} = 1.45 + 0.50 = 1.95$

- Type 1
 $\bar{l}=\bar{q}+\bar{x} = \bar{l} = 0.96 + 0.33 = 1.30$

Temos mais outras 3:

1. O \bar{s} tem que ser o mesmo para os 2
 $\Rightarrow \text{type 0} = \text{type 1} = 1.98$
2. O arrival rate do tipo 0 + tipo 1 tem que ser igual a \bar{r}
 $\Rightarrow 1/3.94 + 1/5.94 = 0,41 \Rightarrow 1/0,41 = 2,40$
3. O steady state utilization tem que ser igual a soma do arrival rate dos types 0 e 1 sobre o service rate $\Rightarrow 0,41 / 0,5 = 0,83 \Rightarrow 0,33 + 0,55 = 0,83$

(d) Why are \bar{w} , \bar{d} , and \bar{s} the same for both job types but \bar{l} , \bar{q} , and \bar{x} are different?

As estatísticas \bar{w} , \bar{d} e \bar{s} são as mesmas, porque utilizam a mesma forma de geração, a única diferença é que estão usando stream diferentes. Os valores de \bar{l} , \bar{q} e \bar{x} são diferentes pois são estatísticas de tempo, logo eles dependem do tempo total de execução de todos os jobs.