Universidade Estadual Paulista "Júlio de Mesquita Filho"

Simulação de Filas

Relatório do Trabalho de Simulação Estocástica

Leonardo Henry Giorgiani Nogueira
Gustavo Ficher Catarino

Instituto de Geociências e Ciências Exatas
Departamento de Estatística, Matemática Aplicada e Computação

Rio Claro - SP
Novembro de 2019

Sumário

1. Objetivos

Este trabalho tem como objetivo o estudo e a criação de modelos de simulação de filas. Para este estudo, foi proposto um modelo de filas baseado em filas de lanchonetes e cantinas na qual existe um sistema para compras e em seguida um sistema para retiradas. A implementação deste modelo e a análise estatística do mesmo foi o objetivo principal do estudo.

1.1 Lista de Objetivos

Podemos listar os objetivos da seguinte maneira:

- Criar o modelo do sistema de filas proposto;
- Implementar o modelo criado;
- Realizar simulações do modelo;
- Obter estatísticas das simulações;
- Analisar os resultados;
- Buscar melhores modelos para o sistema de filas proposto.

2. Resumo

O principal problema de um sistema de filas é o tempo de espera ou ter atendentes parados, para solucionar esse problema é utilizado modelos de simulação para que se obtenha um modelo que seja "ideal". Para esse trabalho o problema a ser solucionado é baseado no sistema de cantinas e lanchonetes onde existe atendentes que vendem os produtos e entregam uma ficha que é utilizada para pegar o produto em uma fila que está depois do sistema do caixa. Com a função de distribuição de probabilidade dada junto ao problema podemos ver a existência de uma fila com alto tempo de espera no caixa e nenhuma fila na retirada do lanche, e para solucionar esse problema devemos achar qual a relação de atendentes nas duas filas reduz o tempo médio na fila e o tempo médio de funcionário livre. Para implementar e realizar essas simulações foi utilizado técnicas de simulação e técnicas de programação orientada a objetos na linguagem Python.

3. Introdução Teórica

3.1 O que é Simulação?

Uma simulação é uma forma de se imitar um processo estocástico presente em algum sistema presente na vida real utilizando uma descrição matemática, ou modelo na forma de programa computacional. O principal objetivo da simulação é poder analisar um sistema que possui limitações se analisado de outra maneira, como por exemplo simular um novo sistema de filas sem que seja necessário o gasto de recursos ou prever os acontecimentos após a demolição de um edifício.

Desde o inicio da computação as simulações são utilizadas para realizar cálculos, analises, previsões e estudos. As simulações utilizam principalmente do método de Monte Carlo que calcula probabilidades de forma heurística, a ideia principal do método de Monte Carlo é realizar sucessivas simulações de um evento e se obter as probabilidades desejadas.

3.2 Por que simular?

A simulação permite o estudo e experimentação de sistemas complexos. Diferentes sistemas podem ser observados por meio de simulações a fim de se desenvolver melhorias para o sistema estudado. A mudança das entradas da simulação podem demonstrar a forma que o sistema interage com elas mostrando problemas ou vantagens do modelo estudado.

3.3 Aplicações

A simulação pode ser utilizada em qualquer área de estudo, desde simulações de um jogo de dados até a simulação de colisão de partículas.

3.4 Vantagens e Desvantagens

A decisão de se utilizar simulação ou não pode ser tomada levando em conta as vantagens e desvantagens do método de analise.

3.4.1 Vantagens

- Novas politicas de sistemas, tomadas de decisão, mudanças procedurais e entre outras modificações podem ser feitas na simulação sem prejudicar o sistema real.
- Mudanças de hardware podem ser testadas sem gasto de recursos.
- Novas hipóteses podem ser testadas facilmente.
- O tempo pode ser reduzido ou expandido de acordo com a necessidade da simulação
- Situações de gargalo podem ser resolvidas

3.4.2 Desvantagens

- A modelagem precisa seguir corretamente o sistema real, o que deixa as implementações com um nível de complexidade muito alto
- Os resultados podem ser difíceis de interpretar.
- A modelagem e analise podem gastar um tempo significativo.
- Algumas vezes uma analise analítica é melhor que uma simulação

4. Metodologia

4.1 Modelo Conceitual

O modelo possui duas filas em serie, a primeira do tipo $M/E_4/n$ e a segunda do tipo $G/E_4/m$, sendo n o número de atendentes da primeira fila e m o número de atendentes da segunda fila.

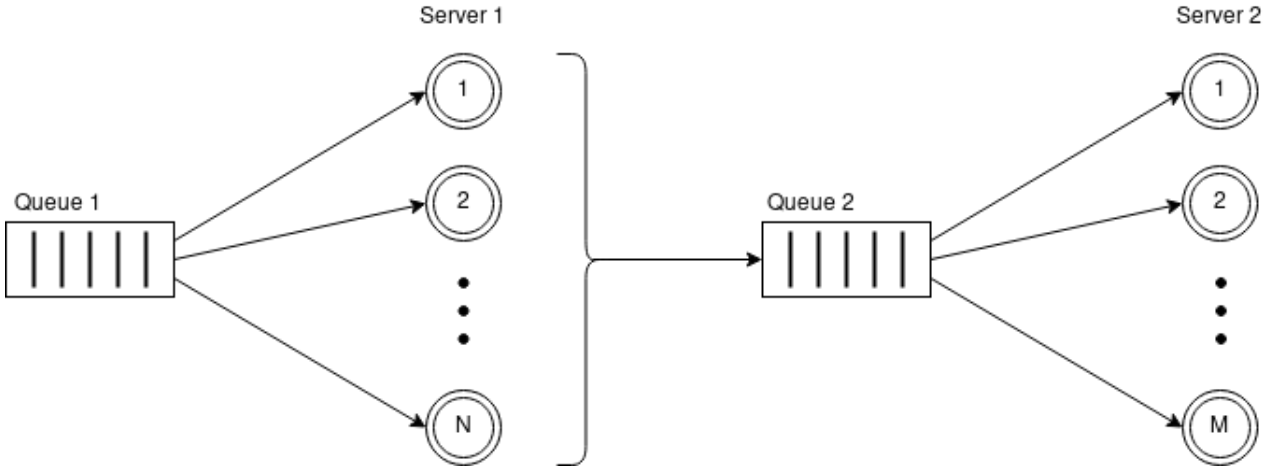

Figura 4.1: Diagrama do modelo de fila

Na primeira fila o tempo entre chegadas possui uma taxa de 4 por minuto em uma distribuição exponencial e o tempo de atendimento uma taxa de 2 por minuto em uma distribuição Erlang com $k = 3$, já na segunda fila o tempo entre chegadas é dado pela saída da primeira fila e o tempo de atendimento possui uma taxa de 1,5 por minuto seguindo uma distribuição Erlang om $k = 4$.

Em uma primeira experimentação da primeira fila, com $n = 1$, é observado que o tempo na fila e o número de pessoas é muito alto, como pode ser visto na tabela 4.1.

Em uma segunda experimentação da primeira fila, com $n = 2$, é observado que o tempo na fila diminui em relação a primeira experimentação, como podemos ver na tabela 4.2.

Aumentando mais ainda o valor de n o tempo médio de espera na fila diminui, mas o tempo entre chegadas na segunda fila diminui fazendo com que o seja necessário também uma analise das duas filas em conjunto.

Tabela 4.1: Últimos clientes na fila do caixa com $n = 1$

TEC	TS	T Real	T inicio	T final	T fila	T sistema	T servidor livre	Tamanho Fila
0.16	1.66	29.18	155.60	157.26	126.42	128.08	0.00	85
0.24	1.76	29.41	157.26	159.01	127.84	129.60	0.00	86
0.41	0.17	29.82	159.01	159.18	129.19	129.36	0.00	87
0.17	2.78	29.99	159.18	161.97	129.19	131.97	0.00	88
0.04	0.58	30.04	161.97	162.54	131.93	132.51	0.00	89

Tabela 4.2: Últimos clientes na fila do caixa com $n = 2$

TEC	TS	T Real	T inicio	T final	T fila	T sistema	T servidor livre	Tamanho Fila
0.24	2.05	29.47	86.90	88.95	57.43	59.48	0.00	80
0.06	2.01	29.53	88.34	90.35	58.81	60.83	0.00	81
0.12	1.21	29.64	88.95	90.16	59.30	60.51	0.00	82
0.02	0.36	29.66	90.16	90.52	60.50	60.85	0.00	83
0.35	0.39	30.01	90.35	90.75	60.34	60.73	0.00	84

4.2 Modelo Computacional

A implementação foi dividida em partes para melhor abstração de cada etapa da simulação. Para uma conversão correta do modelo conceitual para computacional cada servidor possui sua própria fila e a cada chegada de clientes ele é direcionado para o servidor que o atenderá primeiro, simulando assim uma fila única.

4.2.1 Gerador

Para a realização da simulação foi utilizado um gerador congruencial linear com $m = 2 * *(32 - 1) - 1$, $c = 0$ e $a = 16807$. A semente foi gerada utilizando o arquivo randômico do sistema linux.

```
#Random.py

class Entropy:
    @staticmethod
    def rand():
        '''
            Generates a random int using the /dev/urandom file.
        '''
        path = '/dev/urandom'
        with open(path,'rb') as f:
            buffer = f.read(8)
            n = int.from_bytes(buffer, 'little')
        return n
```

```
prev: int

class Congrencial:
    @staticmethod
    def seed(X=None):
        global prev
        prev = X if X is not None and X > 0 else Entropy.rand()

        for _ in range(10): #Used to remove the initial bias of the Congrencial genera
            Congrencial.rand()

    @staticmethod
    def rand(a=16807, c=0, M=2**(32-1)-1):
        '''
            Generates a random number, between 0 and 1, where X(0) is the seed used.
            a is a multiplier between 1 and M
            c is an increment
            M is the greatest machine number
            X(n) = (a X(n-1) + c) % M
        '''
        global prev
        prev = ((a*prev+c) % M)

        return prev/M

Congrencial.seed()
```

4.2.2 Distribuição

Os números gerados aleatoriamente passam por uma transformada inversa para ficar dentro da distribuição desejada.

```
#Distributions.py

from utils.Random import Congrencial as crand
from math import log, exp, factorial, cos, sin, pi, sqrt, floor

class Exponencial:
    def __init__(self, a=1):
        self.a = a

    def x(self, rand=crand.rand):
        return -log(1-rand())/self.a
```

```
    def f(self, x):
        return self.a*exp(-x*self.a)

class Erlang:
    def __init__(self, a=1, k=5):
        self.a = a
        self.k = k

    def x(self, rand=crand.rand):
        def prod(rand):
            prod = 1
            for _ in range(self.k):
                prod *= rand()
            return prod

        return - log(prod(rand))/self.a

    def f(self, x):
        return (self.a*exp(-self.a*x)*((self.a*x)**(self.k-1)))/factorial(self.k-1)
```

4.2.3 Servidores e Clientes

Os servidores e clientes foram implementados em forma de classe para que fosse possível uma abstração maior durante a simulação.

```
#Atoms.py

from collections import deque

class Server:
    '''
        Server class:
            Each server has it own queue, for 1 queue models the clients end_time
            must be appended to the shortest Server.queue.
    '''
    isFree: bool
    last_end_time: float

    def __init__(self):
        self.queue = deque()
        self.isFree = True
        self.last_end_time = 0
```

```
    def append(self, end_time):
        '''
            Append the client end_time to its own server queue
        '''
        self.isFree = False
        self.queue.append(end_time)

    def getAwaitTime(self):
        '''
            Returns the Await Time in the server queue
        '''
        return self.queue[len(self.queue)-1] if len(self.queue) > 0 else 0

    def updateQueue(self, real_time):
        '''
            Update the server queue using the real time
        '''
        while len(self.queue) > 0 and self.queue[0] <= real_time:
            self.last_end_time = self.queue.popleft()

        if len(self.queue) == 0:
            isFree = True

    def __len__(self):
        return len(self.queue)

class Client:
    '''
        Client class:
            interarrival_time: Tempo entre clientes (TEC)
            service_time: Tempo de serviço (TS)
            start_time: Tempo do inicio do serviço
            end_time: Tempo do fim do serviço
    '''
    interarrival_time: float
    service_time: float
    start_time: float
    end_time: float
    spent_time: float
    queue_time: float

    def __init__(self, interarrival_time, service_time, arrive_time, queue_time):
        self.interarrival_time = interarrival_time
        self.service_time = service_time
        self.spent_time = service_time + queue_time
```

```
        self.start_time = arrive_time + queue_time
        self.end_time = self.start_time + service_time
        self.queue_time = queue_time
```

4.2.4 Sistema

Na implementação do sistema foram utilizadas duas classes sendo uma o sistema e a outra uma classe auxiliar para manter as informações estatísticas do sistema.

```
#System.py

from utils.Distributions import Erlang, Exponencial
from BasicModels.Atoms import Server, Client
from dataclasses import dataclass

@dataclass
class Stats:
    n_clients = 0
    total_service_time = 0
    total_spent_time = 0
    total_queue_time = 0
    total_free_server_avgtime = 0
    total_clients_inqueue = 0

    def resume(self, time):
        '''
            Returns stats:
                Average service time,
                Average spent time,
                Average queue time,
                Probability of finding a free attendent,
                Probability of await in queue.
        '''
        return  self.total_service_time/self.n_clients,\
                self.total_spent_time/self.n_clients,\
                self.total_queue_time/self.n_clients,\
                self.total_free_server_avgtime/time,\
                self.total_clients_inqueue / self.n_clients

class System:
    '''
            In This Class the simulation of one queue is made by looking into the
            lowest end time on queue, and represents this model:
```

```
                        Server
        Queue         +->  [] (1)
        | | | | | | | +->  [] (2)
                    ...
                      +->  [] (n)
```

```
    '''
    servers: list
    real_time: float
    stats: Stats

    def __init__(self, n_servers):
        self.servers = [Server() for _ in range(n_servers)]
        self.real_time = 0
        self.stats = Stats()

    def client_arrival(self, client: Client):
        '''
                Append the client in the queue with minimal service time
                (getMinimalServiceTime()), and update the stats of the server.
        '''
        # Stats Processing
        self.stats.n_clients += 1
        self.stats.total_service_time += client.service_time
        self.stats.total_spent_time += client.spent_time
        self.stats.total_queue_time += client.queue_time
        self.stats.total_clients_inqueue += 0 if self.hasFreeServer() else 1

        # Client arrival
        self.servers[self.getMinimalServiceTime()].append(client.end_time)

    def getMinimalServiceTime(self):
        '''
            Returns the index of the server with minimal service time
        '''
        min = self.servers[0]
        idx = 0
        for i, a in enumerate(self.servers):
            if min.getAwaitTime() > a.getAwaitTime():
                min = a
                idx = i

        return idx

    def getMinimalAwaitTime(self):
```

```
    '''
        Returns the minimal await time in the server
    '''
    return max(self.servers[self.getMinimalServiceTime()]
     .getAwaitTime() - self.real_time, 0)

def update(self, time_inc):
    '''
        Update the server time using a time increment (TEC)
        and update the queues status.
    '''
    self.real_time += time_inc
    for a in self.servers:
        a.updateQueue(self.real_time)
    self.stats.total_free_server_avgtime += self.getFreeTime()

def getQueuesSize(self):
    '''
        Returns the sum of the sizes of each server queue
    '''
    count = 0
    for a in self.servers:
        count += len(a)

    return count

def hasFreeServer(self) -> bool:
    '''
        Returns True if some server is idle, otherwise returns False
    '''
    for a in self.servers:
        if a.isFree:
            return True

    return False

def getFreeTime(self):
    '''
        Returns the number of free server and the average free time of them.
    '''
    t_free_att = 0
    free_time = 0
    for a in self.servers:
        if(a.isFree):
            t_free_att += 1
```

```
                free_time += self.real_time - a.last_end_time

        return free_time/t_free_att if t_free_att != 0 else 0
```

4.2.5 Modelo

A implementação do modelo foi feita utilizando 2 servidores em série assim como no modelo conceitual, a execução desse código é equivalente a uma simulação de um espaço de tempo de 30 minutos.

```
#QueueModel.py

from utils.Distributions import Exponencial, Erlang
from utils.Random import Congrencial as crand
from BasicModels.System import System
from BasicModels.Atoms import Client

def simulate(n_caixas = 1, n_atendentes = 1, TMAX = 30, tec_g= Exponencial(4),
 tscaixa_g = Erlang(2,3), tsatend_g = Erlang(1.5, 4)):
    caixas = System(n_caixas)
    atendentes = System(n_atendentes)

    prev_tec_att = 0

    while caixas.real_time < TMAX :
        tec_caixa = tec_g.x()
        ts_caixa = tscaixa_g.x()
        caixas.update(tec_caixa)
        cl_c = Client(tec_caixa, ts_caixa, caixas.real_time,
         caixas.getMinimalAwaitTime())
        caixas.client_arrival(cl_c)
        ts_atend = tsatend_g.x()
        atendentes.update(cl_c.end_time - prev_tec_att)
        prev_tec_att = cl_c.end_time
        cl_a = Client(cl_c.spent_time, ts_atend, atendentes.real_time,
         atendentes.getMinimalAwaitTime())
        atendentes.client_arrival(cl_a)

    return caixas.stats.resume(TMAX), atendentes.stats.resume(TMAX)
```

4.2.6 Simulação

Na simulação foram executadas 5000 vezes com diferentes sementes o código em 4.2.5.

```
#main.py

from progress.bar import Bar

from utils.statistics import ci
from QueueModel import simulate
from utils.Random import Congrencial as crand

if __name__ == "__main__":
    n_atendentes = [(4,6), (5,5), (6,4)]

    for c,a in n_atendentes:
        sAvgSt1 = []; sAvgSt2 = []
        sAvgSpt1 = []; sAvgSpt2 = []
        sAvgQt1 = []; sAvgQt2 = []
        sPFA1 = []; sPFA2 = []
        sPAQ1 = []; sPAQ2 = []

        for i in Bar('Simulando para {} caixas e {} atendentes de lanche'.
            format(c, a)).iter(range(5000)):
        # for _ in range(5000):
            crand.seed()

            (avgSt1, avgSpt1, avgQt1, pFA1, pAQ1),\
            (avgSt2, avgSpt2, avgQt2, pFA2, pAQ2) = simulate(c,a,30)

            sAvgSt1.append(avgSt1); sAvgSpt1.append(avgSpt1)
            sAvgQt1.append(avgQt1); sPFA1.append(pFA1); sPAQ1.append(pAQ1)
            sAvgSt2.append(avgSt2); sAvgSpt2.append(avgSpt2)
            sAvgQt2.append(avgQt2); sPFA2.append(pFA2); sPAQ2.append(pAQ2)

        print("Resultados para {} caixas e {} atendentes de lanche".format(c,a))

        print("""Fila 1: \n
            \tTempo médio de serviço: {}\n
            \tTempo médio gasto no sistema: {}\n
            \tTempo médio de fila: {}\n
            \tProbabilidade de atendente livre: {}\n
            \tProbabilidade de esperar na fila  {}\n
            """.format(ci(sAvgSt1), ci(sAvgSpt1), ci(sAvgQt1), ci(sPFA1), ci(sPAQ1)))
```

```
        print("""Fila 2: \n
            \tTempo médio de serviço: {}\n
            \tTempo médio gasto no sistema: {}\n
            \tTempo médio de fila: {}\n
            \tProbabilidade de atendente livre: {}\n
            \tProbabilidade de esperar na fila  {}\n
            """.format(ci(sAvgSt2), ci(sAvgSpt2), ci(sAvgQt2), ci(sPFA2), ci(sPAQ2)))
```

4.2.7 Validação do Modelo

Para verificar a veracidade do modelo foi criada uma tabela para o teste de mesa. A verificação apenas de um modelo com uma única fila já é suficiente para a validação da simulação já que para duas filas será feita apenas uma alteração dos parâmetros na fila que estará em serie, como pode ser visto em 4.2.5.

A validação do modelo foi feita utilizando o código:

```
#TestQueueModel.py

from utils.Distributions import Exponencial, Erlang
from utils.Random import Congrencial as crand
from BasicModels.System import System
from BasicModels.Atoms import Client

def simulate(n_servers = 2, TMAX = 30, tec_g = Exponencial(4), ts_g = Erlang(2,3)):
    '''
            Simulates the queue model of 1 System with n servers,
        Returns the stats from simulation.

        ** See System Documentation
    '''
    print("tec,ts,treal,tinit,tfim,tfila,tsist,tCLivre,n_Fila")
    sv = System(n_servers)

    while sv.real_time < TMAX:
        tec = tec_g.x()
        ts = ts_g.x()
        sv.update(tec)
        cl = Client(tec, ts, sv.real_time, sv.getMinimalAwaitTime())
        sv.client_arrival(cl)
        print("{:.2f}, {:.2f}, {:.2f}, {:.2f}, {:.2f}, {:.2f}, {:.2f}, {:.2f}, {}"
        .format(
            tec, ts, sv.real_time, cl.start_time, cl.end_time,
            cl.queue_time, cl.spent_time, sv.getFreeTime(), sv.getQueuesSize()
        ))
```

```
return sv.stats.resume(TMAX)
```

As tabelas a seguir podem ser utilizadas para fazer o teste de mesa e validar o modelo, sendo a tabela 4.3 para 1 único servidor e a tabela 4.4 para 2 servidores.

Tabela 4.3: Primeiros clientes na fila do caixa com $n = 1$ para teste de mesa

TEC	TS	T Real	T inicio	T final	T fila	T sistema	T servidor livre	Tamanho Fila
0.13	1.55	0.13	0.13	1.68	0.00	1.55	0.00	1
0.12	1.01	0.25	1.68	2.69	1.43	2.44	0.00	2
0.34	2.02	0.59	2.69	4.71	2.10	4.12	0.00	3
0.02	2.03	0.61	4.71	6.75	4.10	6.13	0.00	4
0.42	0.27	1.04	6.75	7.02	5.71	5.98	0.00	5

Tabela 4.4: Primeiros clientes na fila do caixa com $n = 2$ para teste de mesa

TEC	TS	T Real	T inicio	T final	T fila	T sistema	T servidor livre	Tamanho Fila
0.01	0.57	0.01	0.01	0.58	0.00	0.57	0.01	1
0.62	2.81	0.63	0.63	3.44	0.00	2.81	0.63	1
0.53	1.18	1.16	1.16	2.34	0.00	1.18	0.00	2
0.11	0.88	1.26	2.34	3.22	1.08	1.96	0.00	3
0.13	0.44	1.40	3.22	3.66	1.82	2.26	0.00	4

4.2.8 Informações Adicionais

Os experimentos foram realizados em um processador Intel Core i3 de 3ª Geração, utilizando a versão 3.7.4 do Python. Todas as implementações utilizadas nesse trabalho estão disponíveis em: `https://github.com/leogiorgiani/stochastic_sim`.

5. Resultados

Como a analise foi pensada para um problema real o limite total de funcionários trabalhando foi limitado a 12 pessoas. O código gera um relatório com os intervalos de confiança, com grau de confiança de 95%:

```
Resultados para 4 caixas e 8 atendentes de lanche
Fila 1:

    Tempo médio de serviço: (1.4977958937595024, 1.4979213645596743)

    Tempo médio gasto no sistema: (8.977781272886183, 8.981588254012152)

    Tempo médio de fila: (7.479952657916515, 7.4836996106626295)

    Probabilidade de atendente livre: (0.11662298305019846, 0.11679003973484446)

    Probabilidade de esperar na fila  (0.9625249931145609, 0.9625391822117836)


Fila 2:

    Tempo médio de serviço: (2.6639918767364312, 2.6641852614359225)

    Tempo médio gasto no sistema: (2.9851014749164273, 2.985728281854046)

    Tempo médio de fila: (0.32107003929317296, 0.32158257930495765)

    Probabilidade de atendente livre: (2.4915665632561224, 2.4954846370205566)

    Probabilidade de esperar na fila  (0.8740832480804615, 0.8741757832429191)


Resultados para 5 caixas e 7 atendentes de lanche
Fila 1:

    Tempo médio de serviço: (1.4998575864383117, 1.49998255915493)
```

Tempo médio gasto no sistema: (4.878778766155574, 4.881457396846529)

Tempo médio de fila: (3.3788883547107798, 3.3815076626980645)

Probabilidade de atendente livre: (0.21425638457578827, 0.2145951417964876)

Probabilidade de esperar na fila (0.9488523064082518, 0.9488779245145843)

Fila 2:

Tempo médio de serviço: (2.6650589207303357, 2.665248951339116)

Tempo médio gasto no sistema: (6.752347954968191, 6.755004315083707)

Tempo médio de fila: (4.08724945989805, 4.089794938084388)

Probabilidade de atendente livre: (1.0586172242511325, 1.0596853483473503)

Probabilidade de esperar na fila (0.9175004307879184, 0.9175433716588401)

Resultados para 6 caixas e 6 atendentes de lanche
Fila 1:

Tempo médio de serviço: (1.4986422183710226, 1.4987645112899235)

Tempo médio gasto no sistema: (2.753171820128043, 2.7547095893778746)

Tempo médio de fila: (1.2544964049662926, 1.2559782748786876)

Probabilidade de atendente livre: (0.39209585221788945, 0.39283113293594063)

Probabilidade de esperar na fila (0.9313842501725561, 0.931426319891432)

Fila 2:

Tempo médio de serviço: (2.6628807539399233, 2.6630737322110747)

Tempo médio gasto no sistema: (12.375469490384075, 12.379223772083973)

Tempo médio de fila: (9.712543132686495, 9.716195643630574)

Probabilidade de atendente livre: (0.688358272246912, 0.6889187615538285)

```
    Probabilidade de esperar na fila  (0.9363198400251942, 0.9363482610850271)


Resultados para 7 caixas e 5 atendentes de lanche
Fila 1:

    Tempo médio de serviço: (1.5016018642696087, 1.501725626225487)

    Tempo médio gasto no sistema: (1.9223807729354616, 1.923049043878897)

    Tempo médio de fila: (0.42074420671036555, 0.4213581196088993)

    Probabilidade de atendente livre: (0.7266925530892541, 0.7281066891331511)

    Probabilidade de esperar na fila  (0.9073785419313068, 0.9074466775088051)


Fila 2:

    Tempo médio de serviço: (2.6649075473720587, 2.6650989092967694)

    Tempo médio gasto no sistema: (18.489703583478796, 18.494805292732526)

    Tempo médio de fila: (15.824744121599869, 15.829758297942414)

    Probabilidade de atendente livre: (0.4804468141267793, 0.4807758012421625)

    Probabilidade de esperar na fila  (0.9503335100842654, 0.9503521785803921)
```

Devido ao grande número de simulações o h do intervalo de confiança é da ordem de 10^{-2} na maior parte dos resultados.

A tabela 5.1 mostra os tempos médios totais gastos no sistema para cada relação entre o número de caixas (n) e atendentes (m).

Tabela 5.1: Tabela de tempo gastos no sistema (em minutos)

n	m	Tempo Gasto no sistema	Tempo Gasto nas Filas
4	8	11.96	7.80
5	7	11.63	7.46
6	6	15.12	10.96
7	5	20.41	16.24

6. Conclusão

Podemos observar que o tempo médio de serviço na segunda fila é maior que na primeira e já é um indicio de que seja necessário um numero maior de servidores na segunda fila do que na primeira.

Podemos observar que o aumento de servidores na primeira fila faz com que a segunda fila cresça de forma indesejada fazendo com que seja preciso o aumento de numero de servidores também na segunda fila. Com o número de funcionários limitado a 12 pessoas os melhores tempos foram atingidos com 4 e 8, 5 e 7 caixas e atendentes respectivamente.

Em relação a probabilidade de um servidor ficar livre as simulações de 5/7 e 6/6 (n/m) tiveram melhores resultados sem prejudicar os tempos gasto no sistema.

Por fim, a melhor opção para 12 funcionários aparenta ser 5 funcionários no caixa e 7 no atendimento.