Relatório - Trabalho de Impelementação de um modelo M/M/1

Jorge Luis Murakami Chami, Pedro Diogo Machado

14 de Outubro de 2019

Este trabalho cobre a implementação de um programa Python que permite a modelagem e simulação de um sistema usando um modelo M/M/1, como visto em sala de aula.

1 Arquivos

Os seguintes arquivos estão relacionados ao trabalho estão inclusos no arquivo compactado final:

- **Relatório.pdf** - Cópia deste relatório
- **README.md** - Arquivo texto contendo instruções para instalação dos requisitos e execução do código
- **requirements.txt** - Arquivo texto contendo pacotes Python necessários para o funcionamento correto do sistema
- **configs.ev** - Arquivo de configuração contendo parâmetros que serão usados na execução do programa principal
- **Cliente.py** - Implementação de uma classe representando clientes que passarão pelo sistema
- **Fila.py** - Arquivo principal a ser executado, contém a implementação da classe Fila, que engloba todo o sistema.

2 Código

Nesta seção serão explicados os três arquivos importantes para a execução do sistema, *configs.env*, *Cliente.py* e *Fila.py*

2.1 configs.env

Esse arquivo define uma série de parâmetros que serão utilizados pelo sistema em tempo de execução, tal que mudando os valores contidos nesse arquivo, o comportamento da simulação será diferente. Os parâmetros são:

- Tempo total de simulação em minutos (número inteiro)
- Tempo médio entre chegadas de novos clientes em minutos (número inteiro)
- Tempo médio de atendimento para um cliente em minutos (número inteiro)
- Tipo das distribuições aleatórias de tempos entre chegadas e tempo de serviço (categóricos, com valores possíveis "exponencial", "normal"e "deterministica")

```
[MM1]
# Tempo total de simulacao em minutos
tTotaldeSimulacao=100
# Tempo medio entre chegadas em minutos
# O valor 10, por exemplo, significa uma chegada a cada 10
    minutos
tMedioEntreChegadas=10
# Tempo medio de atendimento em minutos
# Ex: 9.5 minutos de atendimento por cliente
tMedioAtendimento=9.5
# Tipo da distribuicao aleatoria dos TEC e tempo de servico
# os tipos aceitos sao "exponencial", "normal" e "
    deterministica"
# qualquer valor alem desses vai ser tratado como a
    distribuicao deterministica
distTEC=exponencial
distServico=normal
```

2.2 Cliente.py

Este arquivo define a classe Cliente, que guarda informações sobre o tempo de chegada e o tempo de serviço de cada cliente no sistema.

```
class Cliente:
    def __init__(self, t_chegada, t_servico):
        self.t_chegada = t_chegada
        self.t_servico = t_servico

    def __repr__(self):
        return "Cliente␣t_cheg=" + str(self.t_chegada) + "␣
            t_serv=" + str(self.t_servico)
```

2.3 Fila.py

O arquivo principal do projeto, implementa a classe Fila, que define o funcionamento do sistema e além disso, implementa a função *main*, utilizada para executar o código todo. A classe Fila, ao ser instanciada, chama três funções:

- *configure()*, que importa as configurações do arquivo *configs.env*, apresentado anteriormente.
- *gera_entrada()*, que define o estado inicial do sistema.
- *processa()*, que inicia a simulação de fato.

A função *gera_entrada()* utiliza o módulo *random* do pacote de computação numérica *NumPy* para gerar números aleatórios segundo as distribuições escolhidas no arquivo de configuração. Em seguida, cria uma lista de clientes, com tempos de chegada de acordo com a distribuição escolhida, até que o tempo total de simulação seja atingido.

A função *processa()* percorre a lista de clientes, calculando os tempos em que o cliente fica no sistema, é atendido e em que o sistema fica ocioso, e depois calcula a média para esses tempos entre todos os clientes.

```
from numpy import random, round
from queue import Queue
import configparser
import os
from Cliente import Cliente
import numpy.random as r
from statistics import mean


class Fila:
    def __init__(self):
        folder = "/".join( os.path.realpath(__file__).split(
            '/')[:-1] )
        self.configure( folder + '/configs.env' )
        self.gera_entradas()
        self.processa()


    def configure(self, configFilename):
        self.config = configparser.ConfigParser()
        self.config.read(configFilename)
        self.config = self.config['MM1']


    def gera_entradas(self):

        tMedioEntreChegadas = float(self.config['
            tMedioEntreChegadas'])
```

```
    tMedioAtendimento = float(self.config['
        tMedioAtendimento'])

    if self.config['distTEC'] == 'exponencial':
        distTEC = list(map(round, r.exponential(
            tMedioEntreChegadas, size=10000)))
    elif self.config['distTEC'] == 'normal':
        distTEC = list(map(round, r.normal(
            tMedioEntreChegadas, size=10000)))
    else:
        distTEC = [tMedioEntreChegadas for _ in range
            (10000)]

    if self.config['distServico'] == 'exponencial':
        distServico = list(map(round, r.exponential(
            tMedioAtendimento, size=10000)))
    elif self.config['distServico'] == 'normal':
        distServico = list(map(round, r.normal(
            tMedioAtendimento, size=10000)))
    else:
        distServico = [tMedioAtendimento for _ in range
            (100000)]



    tempoAtual = 0
    self.clientes = []
    self.clientes.append(Cliente(0, distServico.pop(0)))
    while tempoAtual + distTEC[0] <= int( self.config['
        tTotaldeSimulacao'] ):
        tempoAtual += distTEC.pop(0)
        novoCliente = Cliente( int(tempoAtual), int(
            distServico.pop(0)) )
        self.clientes.append(novoCliente)


def processa(self):
    tempoAtual = 0
    tempoOcioso = 0
    i = 0
    for c in self.clientes:
        if tempoAtual < c.t_chegada:
            print(f"sistema␣espera␣por␣{c.t_chegada␣-␣
                tempoAtual}␣minutos\n")
            tempoOcioso+= c.t_chegada - tempoAtual
            tempoAtual = c.t_chegada
        c.t_fila = tempoAtual - c.t_chegada
        c.t_atendimento = tempoAtual
        c.t_noSistema = c.t_atendimento+c.t_servico-c.
            t_chegada
```

```
            tempoAtual+= c.t_servico

            i+=1
            print(f"Cliente num.{i}")
            print(c)
            print(f"tempoAtendimento: {c.t_atendimento}")
            print(f"tempoSaida: {c.t_atendimento+c.t_servico
                }")
            print(f"tempo no sistema: {c.t_noSistema}\n")

        print(f"Tempo ocioso do sistema: {tempoOcioso}")
        print(f"Tempo medio no sistema por cliente: { mean([
            c.t_noSistema for c in self.clientes])}")
        print(f"Tempo medio na fila por cliente: { mean([c.
            t_fila for c in self.clientes])}")
        print(f"Tempo medio de servico por cliente: {mean([c
            .t_servico for c in self.clientes])}")


def main():
    Fila()


if __name__ == "__main__":
    main()
```

2.4 Execução

O arquivo *README.md* explica detalhadamente os passos necessários para execução correta do programa, que se resumem a:

- Instalação da linguagem Python no sistema;
- Instalação do pacote *virtualenv* para criação de ambientes virtuais;
- Instalação dos pacotes necessários descritos no arquivo *requirements.txt*;
- Execução do arquivo *Fila.py*.