

PMR3201 - Computação para Mecatrônica

Prof. Thiago de Castro Martins

Prof. Newton Maruyama

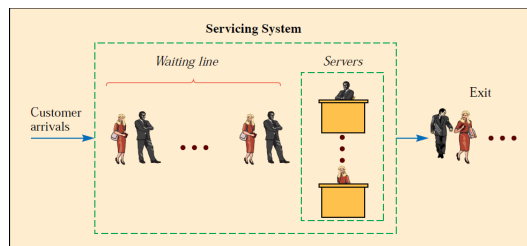
Prof. Marcos de Sales Guerra Tsuzuki

Prof. Rafael Traldi

Prof. André Kubagawa Sato

Monitor: Luiz Fernando Ferreira da Silva

Exercício Programa 2 - Sistema de Simulação de Filas - Versão 2020



Sistemas de fila de atendimento a clientes.

- **DATA FINAL DE ENTREGA: Final do semestre (Data precisa será definida a posteriori).**
- O exercício deve ser feito individualmente.
- Submeta através do sistema MOODLE um único arquivo compactado com o seguinte conteúdo:
 - Os arquivos com os códigos fontes da sua solução,
 - Um relatório de 1 página A4 em formato PDF com a descrição dos experimentos e conclusões relacionadas aos exercícios de projeto propostos.

1 Introdução

Nesse exercício programa deve ser implementado um sistema de simulação de filas. Esses sistemas podem ser caracterizados como sistemas de eventos discretos estocásticos, onde a ocorrência de eventos determina a evolução do sistema.

Na primeira parte esses sistemas são caracterizados. Uma especial atenção é dada para o caráter estocástico do sistema e como realizar amostragem de funções densidade de probabilidades pertinentes a tais sistemas.

Na segunda parte uma especificação do programa é apresentada. Parte do código já é fornecido incluindo algumas classes e funções. Um algoritmo de alto nível do controle da simulação é fornecido para ser utilizado como referência.

Ao final apresenta-se dois exercícios de projeto de dimensionamento do número de filas que devem ser resolvidos utilizando o simulador.

2 Sistemas de simulação de filas

Caracterização do sistema como um Sistema de Eventos Discretos Um sistema discreto possui variáveis que podem assumir apenas um número limitado de valores ou estados (On/Off por exemplo). Um sistema também pode ser determinístico ou estocástico dependendo da relação entre as suas entradas e saídas.

Num **sistema determinístico**, é possível determinar o valor exato do estado final do sistema a partir do seu estado inicial num **sistema estocástico**, ao contrário, não é possível determinar o valor exato de uma ou mais variáveis do sistema porque essas possuem um caráter intrinsecamente aleatório, por exemplo, o tempo de atendimento de um cliente no banco. A simulação de sistemas de eventos discretos é dirigida por eventos. O algoritmo deve examinar todos os eventos para descobrir qual será o próximo evento que vai ocorrer.

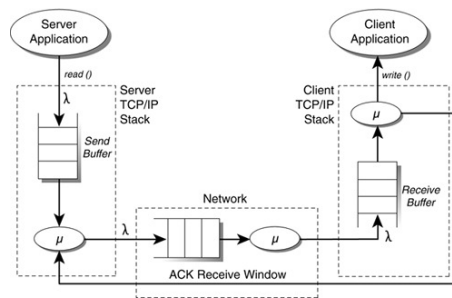


Figura 1: Modelo de filas de um sistema cliente-servidor.

Caracterização do sistema de filas como um Sistema de Eventos Discretos Um dos modelos de sistemas de eventos discretos comumente utilizados são Sistemas de Filas (*Queueing System, Waiting Line*). Para tal, um corpo teórico, denominado Teoria de Filas, foi desenvolvido.

A teoria de filas é utilizada em áreas diversas como: sistemas de atendimento a clientes (agências bancárias, supermercados, etc.), sistemas de manufatura e sistemas de computação (escalonamento e transmissão de dados em redes, escalonamento de processos em sistemas operacionais).

Na Figura 1 apresenta-se um diagrama representando um modelo de filas de um sistema cliente-servidor. O fluxo de dados é regido por processos estocásticos.

Sistema de filas em agências bancárias

- O sistema de filas na agência bancária definido aqui possui um número de caixas dado por n_{caixas} ,
- O sistema é classificado como múltiplas filas múltiplos servidores (MFMS),
- Clientes chegam em intervalos de tempo que são caracterizados como uma **variável aleatória**,
- Uma vez escolhida a fila o cliente não pode mudar,
- o tempo de atendimento do caixa também é uma **variável aleatória** (OBS: na prática esse tempo depende do número de documentos que o cliente necessita processar e da eficiência do caixa).
- A figura abaixo ilustra um sistema desse tipo:

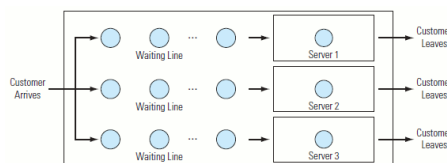


Figura 2: Sistema de fila múltiplas servidores múltiplos.

- A simulação é realizada por eventos. Os seguintes eventos são considerados: **Evento 1** - Chegada do cliente na agência, **Evento 2** - o caixa começa o atendimento do cliente, **Evento 3** - termina o atendimento do cliente.

Sequência de ocorrência de eventos para um cliente

- O cliente chega na agência bancária no instante de tempo dado por $t_{evento1}$ (Evento1),
- O cliente verifica se existe algum caixa livre,
- Caso exista, ele escolhe um caixa e se dirige ao atendimento,

- Caso não exista caixa livre o cliente escolhe uma fila que tenha o menor comprimento e espera o atendimento,
- Quando começa o atendimento pelo caixa esse instante de tempo é denominado t_{evento_2} (Evento 2),
- Quando termina o atendimento esse instante de tempo é dado por t_{evento_3} (Evento 3).

Cálculo das grandezas que caracterizam o desempenho Através de simulações de sistemas de filas objetiva-se a estimativa de grandezas que caracterizam o desempenho do sistema.

Em sistemas de atendimento ao cliente, por exemplo, procura-se estabelecer sistemas de filas que possam aumentar a eficiência do sistema (taxa de atendimento alta) o que consequentemente promove a satisfação dos clientes (tempo de espera na fila pequeno)

Para cada cliente com atendimento finalizado podemos calcular:

- **Tempo de espera na fila:** $\Delta t_{fila} = t_{evento_2} - t_{evento_1}$,
- **Tempo de atendimento:** $\Delta t_{atend} = t_{evento_3} - t_{evento_2}$,
- **Tempo total na agência bancária:** $\Delta t_{total} = t_{evento_3} - t_{evento_1}$

Para um número de clientes $n_{clientes}$ podemos calcular a média e variância de cada uma dessas grandezas:

- **tempo de espera na fila:** $\overline{\Delta t_{fila}}, Var_{\Delta t_{fila}}$
- **tempo de atendimento:** $\overline{\Delta t_{atend}}, Var_{\Delta t_{atend}}$
- **tempo total:** $\overline{\Delta t_{total}}, Var_{\Delta t_{total}}$

Uma outra grandeza importante é o número médio de clientes na fila que pode ser calculado da seguinte forma:

$$\bar{n}_{CF} = \frac{\int_0^{T_{final}} n_{CF}(t) dt}{T_{final}}, \quad (1)$$

onde T_{final} é o tempo quando o último cliente terminam de ser atendido.

Como nessa simulação são utilizadas variáveis aleatórias sabemos que os resultados obtidos de uma única simulação não tem relevância estatística, dessa forma, devemos realizar um **número significativo de repetições** n_{rep} .

Distribuições de probabilidades em simulação de filas

- Devemos determinar o instante de tempo absoluto $t_{evento_1}(i)$ de chegada dos clientes $i = 1 \dots n_{clientes}$,
- Para tal inicialmente amostraremos uma função densidade de probabilidade $f(\lambda)$ que nos fornecerá valores aleatórios para o intervalos de tempo entre chagadas de clientes $\Delta t_{evento_1}(i - 1)$,
- A partir de amostras de $\Delta t_{evento_1}(i - 1)$ podemos facilmente obter os instantes de tempo absolutos.
- A figura abaixo ilustra o processo:

$$\begin{array}{ccccccc} 0 & \xrightarrow{\Delta t_{evento_1}(0)} & \mathbf{t_{evento_1}(1)} & \xrightarrow{\Delta t_{evento_1}(1)} & \mathbf{t_{evento_1}(2)} & \xrightarrow{\Delta t_{evento_1}(2)} & \dots \\ & & & & & & \\ & & \dots & \xrightarrow{\Delta t_{evento_1}(n_{clientes}-1)} & \mathbf{t_{evento_1}(n_{clientes})} & & \end{array}$$

Chegada de clientes: Processos de Poisson

- Filas de eventos usualmente obedecem a Processos de Poisson.
- A chegada de clientes numa agência bancária por exemplo pode ser expressa através de uma distribuição de Poisson.
- A distribuição de Poisson descreve a probabilidade da chegada de n clientes durante um período de tempo T e é expressa por:

$$P_T(n) = \frac{(\lambda T)^n \exp^{-\lambda T}}{n!} \quad (2)$$

onde λ é a média da taxa de chegada de clientes no tempo T .

- Por exemplo, se a taxa média de chegada de clientes é de 3 clientes por minuto ($\lambda = 3$) e deseja-se saber qual a probabilidade da chegada de 5 clientes por minuto ($n = 5, T = 1$), utilizando a equação acima obtemos:

$$P_1(5) = 0.101 \quad (3)$$

- A figura abaixo ilustra uma distribuição de Poisson para $\lambda = 3$. Tanto a média como a variância da distribuição de Poisson são iguais a λ .

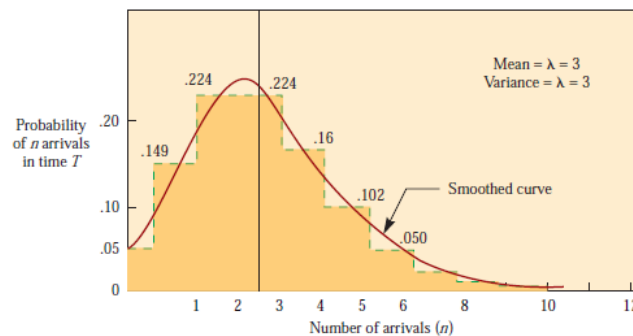


Figura 3: Distribuição de Poisson para $\lambda = 3$ (Fonte [2])

- Se a chegada de clientes segue um processo de Poisson então a função densidade de probabilidade exponencial modela a distribuição de intervalos de tempo entre eventos:

$$f(t) = \lambda \exp^{-\lambda t}, \quad (4)$$

onde λ é o número médio de chegadas por unidade de tempo.

- Obviamente, λ da Equação 4 é o mesmo da Equação 2. A média da distribuição exponencial é $1/\lambda$ e sua variância $1/\lambda^2$. Se existe uma média de λ chegadas por unidade de tempo é natural imaginar que na média uma chegada demora $1/\lambda$.
- Obviamente o mesmo raciocínio vale para o modelo para a função densidade de probabilidade do intervalo de tempo que o servidor demora para finalizar uma tarefa (Taxa de serviço). Por exemplo, quanto tempo o caixa demora para atender um cliente.
- A Figura 4 ilustra uma função densidade de probabilidade exponencial.

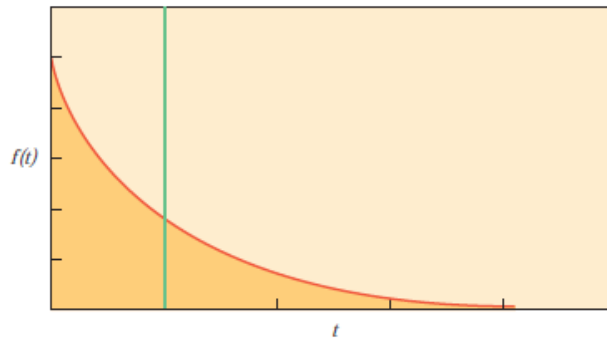


Figura 4: função densidade de probabilidade exponencial para $\lambda = 3$ (Fonte [2]).

- Uma função denominada `fexp()` foi desenvolvida e se encontra no arquivo `AleatorioAlunos.py`.
- A função recebe como parâmetro de entrada a taxa de chegada de clientes em segundos e a duração do expediente em segundos.
- A função retorna uma lista contendo os intervalos de tempo em segundos Δt_{evento_i} entre chegadas de clientes consecutivos.
- O número de amostras gerados é compatível com a duração do expediente. O último cliente não pode chegar após o término do horário de expediente.

```
def fexp(lambdacliente,duracaoexpediente):
    x = 1000*[None] # x e' alocado com 1000 posicoes
    somatoriadotempo = 0 # Somatoria dos intervalos de tempo Dtev1
    k = 0
    # Amostras sao geradas ate' que a somatoria dos intervalos de tempo
    # ultrapasse a duracao do expediente
    while somatoriadotempo <= duracaoexpediente:
        x[k] = rand.expovariate(lambdacliente)
        x[k] = int(x[k])
        somatoriadotempo = somatoriadotempo + x[k]
        k = k+1
    # retira-se o ultimo elemento que esta' alem do horario do expediente
    return x[0:k-1]
```

- Um programa teste para ilustrar a utilização da função se encontra no arquivo `teste_fexp.py`.
- O código do programa é ilustrado a seguir:

```
import numpy as np
import random as rand
from matplotlib import pyplot as plt
from AleatorioAlunos import *

if __name__ == "__main__":
    lambdacliente = 40/3600 # Taxa de chegadas por seg
    duracaoexpediente = 6*3600; # Duracao do expediente em seg
    Dtev1 = fexp(lambdacliente,duracaoexpediente)
    print("Somatoria Dtev1 = ",sum(Dtev1)," Expediente 6h = ",\
    duracaoexpediente)
    # Histograma com 20 setores
```

```
plt.hist(Dtev1,20)
plt.show()
```

- A função `TempoDeAtendimentoCaixa()` gera a duração do tempo de atendimento no caixa utilizando a taxa de atendimento no caixa μ_{caixa} e a imposição de um tempo mínimo de atendimento.

```
def TempoDeAtendimentoCaixa(mucaixa,tempominimoatendimento):
    x = int(rand.expovariate(mucaixa))
    if x >= tempominimoatendimento:
        return(x)
    else:
        return(tempominimoatendimento)
```

3 Especificação do programa EP2

O programa executar uma simulação estocástica de um sistema com múltiplas filas e múltiplos servidores mais especificamente um sistema de filas de atendimento numa agência bancária.

O programa deve atender aos seguintes requisitos:

- taxa de chegada de clientes no sistema,
- taxa de atendimento individualizada para cada caixa,
- tempo mínimo para um atendimento do caixa,
- especificação do número de repetições da simulação,
- análise desempenho através de cálculo de média e variância,
- display gráfico do estado da fila para uma simulação específica (iteração).

Código a ser implementado no EP2: classes, funções e programa principal Uma parte do código do sistema de simulação de filas é fornecida pronta.

A estrutura de arquivos proposta e as funções e classes que os compõem são apresentados a seguir. **As partes a serem desenvolvidas pelos alunos estão evidenciadas por retângulos.**

Os seguintes arquivos devem compor o seu sistema:

- `SimuladorDeFilas.py`:
 - Variáveis globais
 - Função `simulacao`
 - Programa principal `main()`
- `FilasAlunos.py`:
 - Classe `Cliente`
 - Classe `FilaCaixa`
 - Classe `FilaEventos` (Nova implementação com estrutura dinâmica)
 - Função `verificaCaixaLivre(sinalCaixaLivre)`
 - Função `achaMenorFila(vetorFilaCaixa)`
 - Função `TamanhoDasFilas(vetorFilaCaixa)`
 - Classe `ListaClientesSaida`

- AleatorioAlunos.py
 - Função `fexp(lambdacliente, duracaoexpediente)`
 - Função `TempoDeAtendimentoCaixa(mucaixa, tempominimoatendimento)`
 - Função `calculaEstatisticas(listasaida, logfilas)`

Variáveis globais O seguinte conjunto de variáveis globais são propostas para o desenvolvimento do programa:

- `lambdacliente_hour`: taxa de chegada de clientes na agência bancária por hora,
- `duracaoexpediente_hour`: duração do expediente em horas,
- `nCaixas`: número de caixas,
- `vetorFilaCaixa[]`: array onde cada posição armazena um objeto da classe `FilaCaixa`,
- `vetorTaxaDeAtendimentoCaixa[]`: array onde a posição i armazena a taxa de atendimento μ_i do caixa i . Obs: a taxa μ_i pode ser individualizada.
- `tempominimodeatendimento`: tempo mínimo que um atendimento é realizado em segundos,
- `sinalCaixaLivre[]`: array booleano (`True`, `False`) que indica se o caixa i está livre para atendimento,
- `FE`: objeto da classe `FilaEventos`,
- `saida`: objeto da classe `ListaClientesSaida`
- `logfilas`: lista onde sub-listas se referem a um instante de tempo e o número de clientes em cada fila.
- `numeroderepeticoes`: número de repetições que uma simulação deve ser realizada.

OBS: Esse conjunto de variáveis é apenas uma proposta, o aluno não precisa necessariamente utilizar como especificado.

Classe Cliente

- A classe denominada `Cliente` define objetos que armazenam todas as informações de ocorrências dos eventos para um cliente específico.
- Utiliza-se a terminologia *Inserir cliente na fila* ou *Inserir um Evento 1 na Fila de Eventos* se referindo ao mesmo objeto.
- As variáveis internas da classe são ilustradas a seguir:

```
class Cliente():
    def __init__(self, clienteid, tipoEvento, timeEvento1):
        self.__clienteid = clienteid
        self.__tipoEvento = tipoEvento # tipo do evento que sera' o
        # proximo a ocorrer
        self.__timeEvento1 = timeEvento1
        # instante de tempo de ocorrencia
        # do Evento 1
        self.__timeEvento2 = 0 # idem Evento 2
        self.__timeEvento3 = 0 # idem Evento 3
        self.__caixaid = 0 # identificacao do caixa e da fila
        # em que foi colocado
```

- `__clienteid`: número inteiro sequencial que identifica o cliente.
- `__tipoEvento`: **identifica o tipo do evento que será o PRÓXIMO a a ocorrer**. Pode assumir os valores 1, 2 e 3

- `__timeEvento1`, `__timeEvento2`, `__timeEvento3`: instantes de tempo absolutos em que ocorreram ou ocorrerão os eventos 1, 2 e 3. Unidade de tempo em segundos.
- `__caixaid`: número inteiro (0, 1, 2, ...) que identifica o caixa e a fila associada.
- Setter e getter para `__clienteid`:

```
def set_clienteid(self, clienteid):
    self.__clienteid = clienteid

def get_clienteid(self):
    return(self.__clienteid)
```

- Setter e getter para `__tipoEvento`:

```
def set_tipoEvento(self, tipo):
    if tipo in range(1, 4):
        self.__tipoEvento = tipo
    else:
        print('Tipo de evento nao existente')

def get_tipoEvento(self):
    return(self.__tipoEvento)
```

- Setter e getter para `__timeEvento1`, `__timeEvento2`, `__timeEvento3`:

```
def set_timeEvento(self, time, tipoevento):
    if tipoevento == 1:
        self.__timeEvento1 = time
    else:
        if tipoevento == 2:
            self.__timeEvento2 = time
        else:
            if tipoevento == 3:
                self.__timeEvento3 = time
    def get_timeEvento(self, tipoevento):
        if tipoevento == 1:
            return(self.__timeEvento1)
        else:
            if tipoevento == 2:
                return(self.__timeEvento2)
            else:
                if tipoevento == 3:
                    return(self.__timeEvento3)
```

Classe FilaEventos: variáveis e operações

- A única variável é a fila `__filae`

```
class FilaEventos():
    def __init__(self):
        self.__filae = []
```

- Insere e retira eventos da fila de eventos:

```

# x e' um dicionario (time,cliente)
# time: funciona como a chave de ordenacao da fila de
# prioridade
# cliente: objeto da classe Cliente()
def insereFilaEventos(self,x):
    heapq.heappush(self.__filae,x)

# Retira da fila de eventos
# retorna o objeto da classe Cliente()
def retiraFilaEventos(self):
    x = heapq.heappop(self.__filae)
    return(x[1])

```

- Verifica se a fila de eventos está vazia:

```

def FilaEventosVazia(self):
    if self.__filae == []:
        return(True)
    else:
        return(False)

```

- Essa implementação é colocada aqui como referência, o aluno deve implementar uma solução com lista ligada.

Classe ListaClientesSaida: variáveis e operações

- Essa classe contém apenas a lista, __filasaida, onde objetos da classe Cliente são armazenados quando o atendimento é terminado.

```

class ListaClientesSaida():
    def __init__(self):
        self.__filasaida=[]

```

- Insere e retira da lista:

```

def insereListaSaida(self,cliente):
    self.__filasaida.append(cliente)

def retiraListaSaida(self):
    return(self.__filasaida.pop(0))

```

- Verifica se a lista está vazia:

```

def ListaSaidaVazia(self):
    if len(self.__filasaida) == 0:
        return(True)
    else:
        return(False)

```

- Calcula o tamanho da lista:

```

def tamListaSaida(self):
    x = len(self.__filasaida)
    return(x)

```

Classe FilaCaixa

- A classe FilaCaixa implementa uma fila com comportamento FIFO (First-In First-Out) e deve ser implementado utilizando uma estrutura do tipo lista. Essa fila tem número de posições ilimitadas.
- As seguintes operações devem ser implementadas:
 - `insereFilaCaixa(cliente)`: insere na fila do caixa um objeto da classe `Cliente`,
 - `retiraFilaCaixa()`: retira um objeto da fila do caixa,
 - `FilaCaixaEstaVazia()`: verifica se a fila está vazia,
 - `tamFila()`: calcula o tamanho da fila (número de clientes).

Classe FilaEventos: implementação dinâmica com lista ligada

- O aluno deve desenvolver essa classe de maneira que possa ter comportamento equivalente ao apresentado com `heapq`.
- A classe FilaEventos implementa uma fila com prioridades o conteúdo da fila são objetos da classe `Cliente` e a chave de ordenação (*key*) corresponde ao instante de tempo absoluto em que ocorrerá o próximo evento.
- As seguintes operações devem ser implementadas:
 - `insereFilaEventos(x)`: `x` é um dicionário `x=(timeEvento,cliente)` onde `timeEvento` é o instante de tempo absoluto da ocorrência do próximo evento e `cliente` é um objeto da classe `Cliente`.
 - `retiraFilaEventos()`: devolve apenas o objeto da classe `Cliente`
 - `FilaEventosVazia()`: verifica se a fila de eventos está vazia.

Outras funções do arquivo `FilasAlunos.py`

- `verificaCaixaLivre(sinalCaixaLivre)`: verifica se existe um caixa livre. Devolve o dicionário (`sinal,caixaid`) onde `sinal` é uma variável booleana. Se `sinal=True` então `caixaid` deve conter um índice (0,1,2,...) correspondente a um caixa livre. Função pode devolver índice referente a qualquer caixa livre.
- `achaMenorFila(vetorFilaCaixa)`: devolve o índice correspondente a menor fila de caixa. `vetorFilaCaixa` é um array que contém em cada posição um objeto da classe `FilaCaixa`. Se houver mais de uma fila com o menor tamanho a função pode devolver o índice de qualquer uma dessas filas.
- `TamanhoDasFilas(vetorFilaCaixa)`: devolve uma lista contendo o tamanho de cada uma das filas.

Função Simulacao

- A função `Simulacao` deve implementar um ciclo completo de simulação.
- As variáveis de retorno dessa função devem ser `saida` objeto da classe `ListaClientesSaida` que deve conter todos os objetos da classe `Cliente` na ordem em que o atendimento foi terminado e `logfilas` uma lista contendo instantes de tempo t em que ocorreram aumento ou diminuição do tamanho de alguma fila e o tamanho de cada uma das filas:
- `logfilas` pode ser usado para plotar o número de clientes na fila a cada instante.
- Um possível algoritmo para referência de implementação é apresentado a seguir.

```
1 Criar <nc> eventos <Evento1> # criar objetos da classe <Cliente>
2                               # todos os eventos <Evento1> são criados no inicio
3 Inserir eventos <Evento1> na fila de eventos <FE>
4 Enquanto houver eventos em <FE> Faça
5 {
6     <evento> <-- retira evento de <FE>
7     <clock> <-- instante absoluto de ocorrencia de <evento>
```

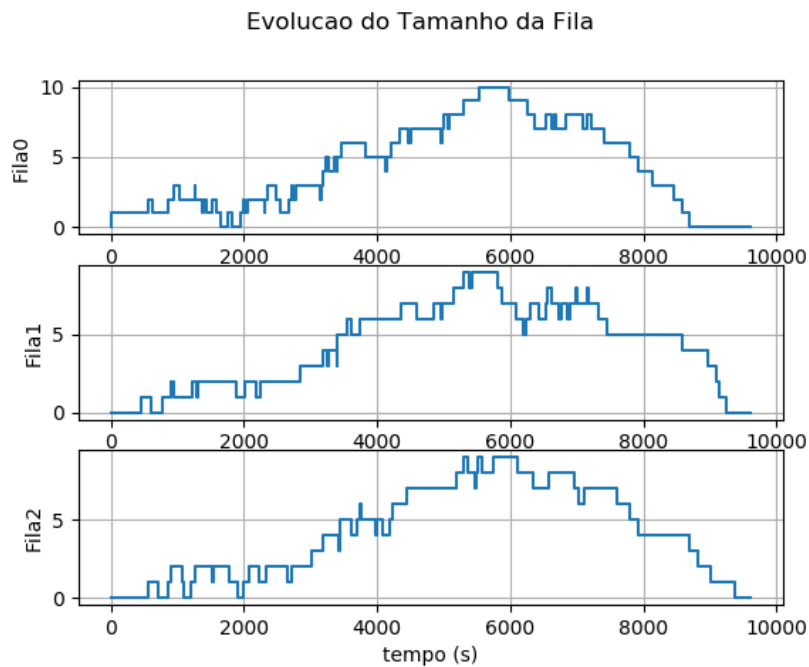
```

8      Se <evento> for <Evento1> então
9      {
10     Se existe(m) caixa(s) livre(s) Então
11     {
12     escolhe um caixa livre identificado como <caixaid>
13     cria um Evento2 para o instante de tempo <clock> associado a <evento>
14     Sinaliza o caixa identificado por <caixaid> como ocupado
15     }
16     Senão
17     {
18     Escolha uma fila de caixa mais curta
19     Insere <evento> na fila de caixa
20     Loga o estado das filas em <logfilas>
21     }
22     Senão Se <evento> for do tipo 2 Então
23     {
24     Dt_atend <-- amostra a funcao que gera o tempo de atendimento desse caixa
25     Gera um <Evento3> para <evento> associado ao instante de tempo <clock>+<Dt_atend>
26     Insere <evento> na fila de eventos <FE>
27     }
28     Senão # <Evento 3>
29     {
30     Coloca <evento> na lista de saida de clientes <saida>      # <Classe ListaSaidaCientes>
31     O caixa identificado como <caixaid> associado a <evento> é sinalizado como livre
32     Se existem clientes nessa fila Então
33     {
34     <proximoevento> <-- retira evento da fila do caixa
35     Loga o estado das filas em <logfilas>
36     Cria um <Evento2> aassociado a <proximoevento> associado ao instante de tempo <clock>
37     Insere <proximoevento> na fila de ventos <FE>
38     }
39
40     }
41 }

```

Programa principal main

- O programa principal deve controlar a realização das várias iterações da função Simulacao().
- O número de iterações é dado pela variável numeroderepeticoes
- Cálculos de médias e variâncias que caracterizam o desempenho do sistema devem ser realizados considerando todas as iterações.
- Deve ser implementado um gráfico tipo degrau (step()) que visualiza o estado da fila do caixa a cada instante.
- Deve ser permitido ao usuário escolher qual iteração específica ($i = 0, \dots, n_{iteracoes}-1$) deve ser visualizada.



- Uma possível solução para o gráfico tipo degrau é ilustrado abaixo:

```
# Plot do estado das filas ao longo do tempo
# numero de subplots e' dependente de nCaixas
# cada subplot e' incorporado sequencialmente
# x = tempo da simulacao em segundos
# y = tamanho de uma das filas
#
fig, ax = plt.subplots(nCaixas)
fig.suptitle('Evolucao do Tamanho da Fila')
nlines = len(ltemporaltamfila)
# x recebe a coluna 0 da lista
x = [row[0] for row in ltemporaltamfila]
for k in range(1,nCaixas+1):
    # y recebe a coluna k da lista
    y = [row[k] for row in ltemporaltamfila]
    ax[k-1].step(x, y)
    ax[k-1].set_xlabel('tempo (s)')
    ax[k-1].set_ylabel('Fila'+str(k-1))
    ax[k-1].grid()
fig.canvas.draw()
```

4 Exercícios de projeto

Verifique experimentalmente qual o número de caixas adequados dentro das especificações abaixo:

1. Qual o número mínimo de caixas dentro das especificações propostas abaixo para que na média o número de clientes na fila seja menor ou igual a 5 ? Dado as seguintes especificações de um sistema de filas:
 - (a) Taxa de chegada de clientes $\lambda = 50$ clientes/hora,
 - (b) Duração do expediente $t_{exped} = 6$ horas,

- (c) Taxa de atendimento do caixa (igual para todos os caixas): $\mu = 15$ clientes/hora,
 - (d) Tempo mínimo de atendimento (igual para todos os caixas): $t_{min} = 120$ seg,
 - (e) Número de repetições: $n_{rep} = 50$
2. Idem para as mesmas condições mas com Taxa de chegada de clientes $\lambda = 70$ clientes/hora,

Escreva um relatório de 1 página A4 detalhando os seus experimentos e as suas conclusões.

5 Bibliografia

1. Christos G. Cassandras and Stephane Lafortune, *Introduction to Discrete Event Systems*, Springer, 2nd Edition, 2009.
2. F. Robert Jacobs and Richard B. Chase, *Operations and supply management: the core*, McGraw-Hill, 2012.