Universidade do Minho

2ºSemestre 2019/20

(MIEI, 3ºAno)

Modelos Estocásticos de Investigação Operacional

Trabalho Prático

Identificação do Grupo

Número:	*Nome completo:*	*Rubrica:*
A83631	Filipa Alves dos Santos	Filipa Santos
A85006	Hugo André Coelho Cardoso	HCardoso
A84475	João da Cunha e Costa	João Costa
A84464	Válter Ferreira Picas Carvalho	Válter Carvalho

Data de entrega: 2020-__05__-__11__

Índice de conteúdos

1. Introdução **2**
2. Desenvolvimento **3**
2.1. Parte 1 3
2.1.1. Descrição do problema 3
2.1.2. Algoritmos 4
2.1.3. Resultados 6
2.2. Parte 2 7
2.2.1. Problemática 7
2.2.2. Tipo de Modelo 7
2.2.3. Principais Características 7
2.2.4. Condições de Aplicação 8
2.2.5. Cálculo da Função de Probabilidade 9
2.2.6. Resultados 10
2.2.6. Referências 11
3. Anexos **12**
3.1. Anexo A1 12
3.2. Anexo A2 12
3.3. Anexo A3 13
3.4. Anexo A4 20

1. Introdução

Neste trabalho prático, era pretendido obter a solução ótima do cenário apresentado no enunciado, aplicando métodos e algoritmos, bem como uma fazer pesquisa sobre a matéria, tudo de modo a desenvolvermos o nosso conhecimento na área de Programação Dinâmica Estocástica e Processos Markovianos.

A primeira parte consiste de um problema de um empresário que gere 2 filiais de aluguer de automóveis e quer maximizar os seus ganhos através de um número específico de transferências. Através do método de iteração de valor, conseguimos determinar esse número. Todo o raciocínio e código envolvido neste processo será abordado no resto deste relatório.

Já na segunda parte, apresentamos o resumo de um artigo científico que aborda a utilização de modelos de Markov de vários estados na análise da progressão da Doença Renal Crónica (DRC) (publicado no Turkiye Klinikleri Journal of Biostatistics), procurando identificar a problemática, as características do modelo desenvolvido e os resultados obtidos.

2. Desenvolvimento

2.1. Parte 1

2.1.1. Descrição do Problema

O problema descrito no enunciado tem como **objetivo** maximizar o lucro total obtido nas 2 filiais. Isto é, determinar a política ótima de transferência diária entre as 2 filiais de modo a ganhar mais a longo prazo. Assim, com a noção de "transferência diária", concluímos que um **estágio** será a passagem de 1 dia e que há um número infinito de estágios, sendo que não há nenhum período de tempo apresentado no enunciado.

Como todas as alterações ao ganho/custo são relativas à transferência, entrega, pedidos e armazenamento de automóveis, é fácil concluir que os nossos **estados** vão ser a quantidade de automóveis em stock em ambas as filiais. Cada filial a qualquer ponto pode ter de 0 a 12 automóveis, ou seja, tem 13 estados individuais distintos. Juntamente com os estados da outra filial, no total teremos 13x13 combinações, isto é, 169 estados possíveis.

Quanto às **decisões** possíveis, existem 7 cenários possíveis:

1. Não transferir nenhum carro entre filiais;
2. Transferir 1 automóvel da filial 1 para a filial 2;
3. Transferir 2 automóveis da filial 1 para a filial 2;
4. Transferir 3 automóveis da filial 1 para a filial 2;
5. Transferir 1 automóvel da filial 2 para a filial 1;
6. Transferir 2 automóveis da filial 2 para a filial 1;
7. Transferir 3 automóveis da filial 2 para a filial 1;

Podemos concluir que, como o período de tempo total não é restringido e como existem várias decisões possíveis, estamos na presença de um **problema de número infinito de estágios com alternativas**.

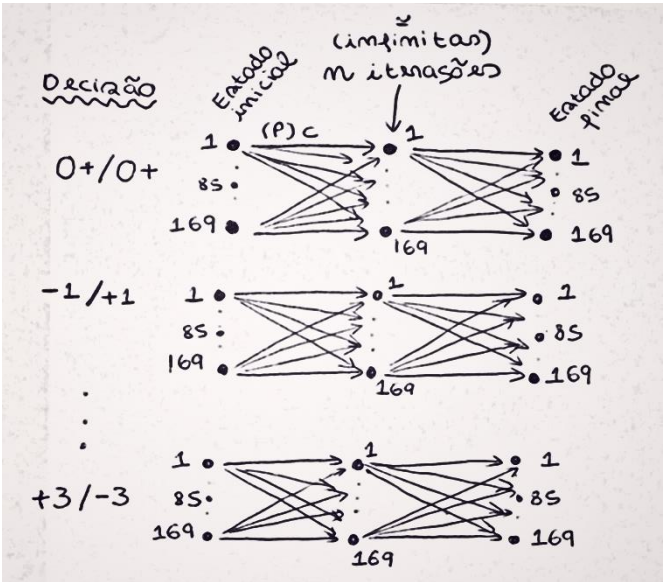

Figura 1 - Esboço parcial da rede de Prog. Dinâmica

2.1.2. Algoritmos

Em termos de algoritmos para gerar as matrizes e para realizar o método de iteração de valor, decidimos criar um **programa em Java** com as funções necessárias. Vamos começar por explicar o código correspondente à criação das matrizes de transição.

Começando pelas **matrizes de probabilidades**, foram necessárias 3 funções distintas:

- **`filialTransicao:`** esta função cria as 2 matrizes de probabilidades 13x13 correspondentes a cada uma das filiais, considerando que não ocorrem quaisquer transferências entre elas. Estas matrizes serão as bases para outras correspondentes às situações em que há transferências entre filiais. No código dividimos o nosso raciocínio em 2 partes: quando `i>=j` e quando `j>i`, sendo `i` e `j` o estado inicial e o estado final respetivamente. Para ambos os casos, determinamos a fórmula geral para calcular as probabilidades detetando padrões nos vários exemplos que fizemos à mão:

  ```
  i>=j : E(0) * P(i-j) + ... + E(j-1) * P(i-1)

  j>i  : E(j-i) * P(0) + ... + E(j-1) * P(i-1)
  ```

  Já para os pedidos e entregas em excesso, tivemos que os tratar em separado das 2 partes descritas acima. Para os pedidos em excesso tivemos o seguinte raciocínio, que depois traduzimos para código:

  ```
  E(j) * (P(i) + ... + P(12))
  ```

  Deste modo estamos a contar com todos os pedidos em excesso que são feitos quando já não há mais automóveis disponíveis nesse dia. Relativamente às entregas em excesso, essas só acontecem quando o estado final é 12 (`j==2`). Decidimos tratar dessa parte com o seguinte raciocínio:

  ```
  P(0)*(E(j-i+1)+...+E(y)) + ... + P(i-1)*(E(11)+E(12))
  ```

- **`transfereEntreFiliais:`** como foi dito acima, esta função transforma as matrizes bases nas outras 12 matrizes 13x13 necessárias, correspondentes às outras 6 decisões em que há transferências entre filiais. Para explicarmos esta transformação, iremos dar o exemplo de quando a filial 1 transfere 1 carro para a filial 2. Neste caso, como a filial 1 enviou um automóvel (durante a noite), esta só poderá começar o dia com no máximo 11 automóveis logo, para obtermos esta matriz fazemos um shift das linhas da matriz base 1 unidade para cima e atribuímos à linha 12 valores impossíveis (como, p.e., 999999). Na matriz correspondente à filial 2, que recebe 1 automóvel, faz-se um shift contrário, de 1 unidade para baixo.

- **`juntaTransicoesFiliais:`** com as matrizes 13x13 feitas, fizemos esta função que junta os pares correspondentes de matrizes (a -1 da filial 1 com +1 da filial 2, a +2 da filial 1 com -2 da filial 2, etc), criando 7 matrizes a partir das 14 originais. Para cada uma delas, tivemos que gerar todas as combinações de estados entre as 2 matrizes 13x13 correspondentes, ou seja, 169 combinações diferentes, o que gera uma matriz 169x169. Cada entrada da matriz é a multiplicação de 2 probabilidades, uma de cada filial, sendo que queremos que as transições nas 2 filiais ocorram simultaneamente.

A criação das **matrizes de contribuições** teve um raciocínio semelhante:

- **`calculaContribuiçoes:`** esta função cria as 2 matrizes de contribuição 13x13 correspondentes a cada uma das filiais, considerando que não ocorrem quaisquer transferências entre elas. O raciocínio vai ser muito similar à `filialTransicao` sendo que queremos o custo/ganho médio numa certa transição. Simplesmente vamos ter que multiplicar em cada parcela pelo ganho, que vai ser o número de pedidos * 30, dado que cada pedido equivale a um lucro de 30 euros para o empresário:

```
i>=j : E(0) * P(i-j) * (i-j) * 30 + ... + E(j-1) * P(i-1) * (i-1) * 30

j>i  : E(j-i) * P(0) * 0 * 30 + ... + E(j-1) * P(i-1) * (i-1) * 30

E(j) * (P(i) + ... + * (12)) * i * 30

P(0) * (E(j-i+1)+...+E(y)) * 0 * 30 + ... + P(i-1)*(E(11)+E(12)) * (i-1) * 30
```

  Além disso, para os dias que acabam com mais de 8 automóveis, temos que considerar a taxa extra (para o estacionamento adicional necessário) de 10 euros e retirá-la do valor.

- **`contribuicoesTransferencia:`** faz os shifts iguais à `transfereEntreFiliais` mas para estas matrizes de contribuição. Nas linhas impossíveis vamos considerar outro valor impossível (-999999).

- **`juntaContribuicoes:`** tem o mesmo objetivo e funciona do mesmo modo que a `juntaTransicoesFiliais,` apenas com algumas alterações. Agora, ao juntarmos as contribuições de duas matrizes, somamos os custos/ganhos para obtermos a contribuição total. Além disso, subtraímos o custo correspondente à(s) transferência(s) feita(s). P.e., se ocorreu a transferência de 2 carros, depois de somarmos os valores das duas matrizes, retiramos 14 e fazemos o mesmo para todas as entradas dessas matrizes. Novamente, esta função vai receber 14 matrizes 13x13, juntar os pares e gerar as 7 matrizes de contribuição necessárias.

Depois de termos todas matrizes feitas, criamos a função `converge` para utilizarmos o **método de iteração de valor** e chegarmos ao resultado pretendido. A função segue a mesma lógica que o método mencionado logo começa por calcular as matrizes de esperança. De seguida, vai iterar o resto do algoritmo um grande número de vezes (nós optamos por 100 vezes), de modo a simular estágios infinitos. Em cada iteração, como é esperado, calcula as 7 matrizes Vns (169x1), correspondentes às 7 decisões, determina a Fn e por fim, Dn. O nosso raciocínio para o cálculo destas 3 matrizes foi inspirado nestas fórmulas:

$$V^k = Q^k + P^k Fn - 1$$

$$Fn[i] = \max\left(V^0[i], \dots, V^6[i]\right)$$

$$Dn = Fn - (Fn - 1)$$

Por fim, também fizemos uma função puramente mecânica chamada `gravaTransicoesExcel` para conseguirmos apresentar as nossas matrizes 169x169 de modo legível.

2.1.3. Resultados

Depois de 100 iterações, o nosso algoritmo fez com que os valores de Dn convergissem para um único valor, 42.568018437459045. Determinamos assim que o maior lucro diário que o empresário fará está compreendido entre 42.565 e 42.575 euros, com a política ótima abaixo:

- 0 - Decisão +0/+0
- 1 - Decisão -3/+3
- 2 - Decisão -2/+2
- 3 - Decisão -1/+1
- 4 - Decisão +1/-1
- 5 - Decisão +2/-2
- 6 - Decisão +3/-3

Filial 2

Filial 1

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	4	4	0
2	0	0	0	0	0	0	0	0	0	0	5	4	0
3	0	0	0	0	0	0	0	0	0	6	5	4	0
4	0	0	0	0	0	0	0	0	0	0	5	4	0
5	0	0	0	0	0	0	0	0	0	0	0	4	0
6	0	0	0	0	0	1	1	0	0	0	0	0	0
7	0	0	2	1	1	1	1	1	1	0	0	0	0
8	0	3	2	1	1	1	1	1	1	1	0	0	0
9	0	3	2	1	1	1	1	1	1	1	1	0	0
10	0	3	2	2	2	2	2	2	2	2	2	0	0
11	0	3	3	3	3	3	3	3	3	3	3	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0

2.2. Parte 2

2.2.1. Problemática

O principal objetivo do artigo científico analisado é desenvolver um modelo estocástico para a progressão de Doença Renal Crónica (DRC).

A Doença Renal Crónica causa um declínio progressivo das capacidades de filtração dos rins ao longo do tempo e tem vários efeitos agravantes como complicações metabólicas, aumento do risco de acidentes cardiovasculares, infeção, etc. Sendo uma doença crónica, a sua progressão pode ser dividida por várias fases e as hipóteses de inibição da doença são diretamente proporcionais ao quão cedo a mesma é detetada e começado o tratamento.

2.2.2. Tipo de Modelo

Foi usado um modelo de Markov de tempo contínuo com vários estados homogéneos, dado que a deterioração da DRC é contínua no tempo e a probabilidade de transição de um estado para outro depende da quantidade de tempo no mesmo e não do instante em que a transição se dá. O modelo foi usado para calcular as probabilidades de transição de estados mais baixos para estados maiores, dado que a DRC é irreversível, bem como o tempo médio de estadia em cada estado.

2.2.3. Principais Características

Os estados do modelo traduzem as fases da DRC, que são indicativas da condição dos rins (quanto maior o número, mais debilitados), e existem 5 estados diferentes – 1, 2, 3, 4 e 5. O estado 5 é um estado absorvente e os restantes são estados transientes. Só é possível um paciente mover-se para estados maiores, pois a doença é irreversível, pelo que é impossível voltar a estados anteriores.

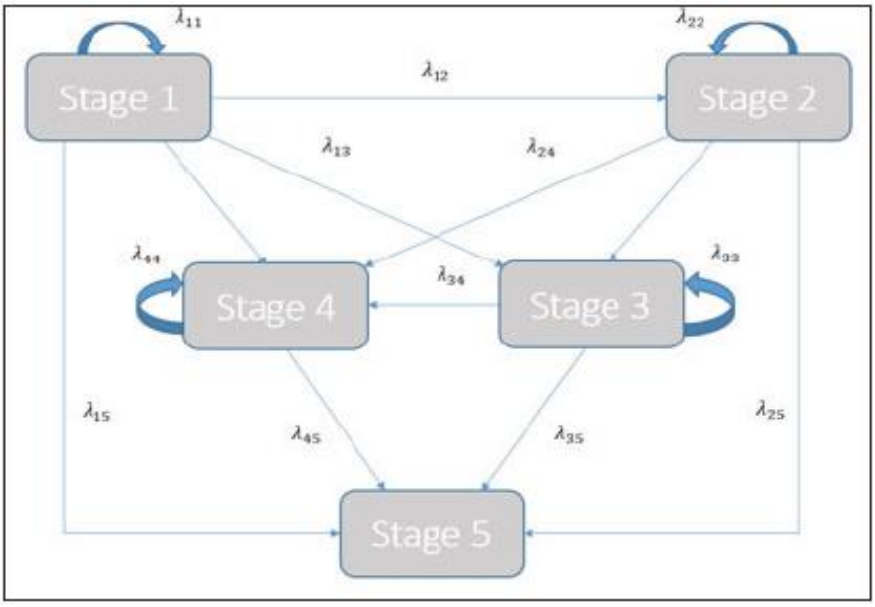

Figura 2 - Diagrama de transição entre estados

A matriz transição que se pretende calcular é a seguinte, sendo $p_{kl}(s,t)$ a probabilidade de um indivíduo num estado k e instante s se encontrar num estado l no instante t.

$$P(s,t)=\begin{pmatrix} p_{11}(s,t) & p_{12}(s,t) & p_{13}(s,t) & p_{14}(s,t) & p_{15}(s,t) \\ p_{21}(s,t) & p_{22}(s,t) & p_{23}(s,t) & p_{24}(s,t) & p_{25}(s,t) \\ p_{31}(s,t) & p_{32}(s,t) & p_{33}(s,t) & p_{34}(s,t) & p_{35}(s,t) \\ p_{41}(s,t) & p_{42}(s,t) & p_{43}(s,t) & p_{44}(s,t) & p_{45}(s,t) \\ p_{51}(s,t) & p_{52}(s,t) & p_{53}(s,t) & p_{54}(s,t) & p_{55}(s,t) \end{pmatrix}$$

Figura 3 - Matriz de transição

Dado que a doença é irreversível, $p_{kl}(s,t) = 0$, para qualquer $k > l$, e $p_{55}(s,t) = 1$, uma vez que o estado 5 é o último (estado absorvente).

2.2.4. Condições de Aplicação

O estudo analisado teve em conta os dados relativos a 117 pacientes sofrendo de Doença Renal Crónica entre Março de 2006 e Outubro de 2016. Foram anotadas informações relevantes como o género, idade, índice de massa corporal e histórico detalhado de outras doenças propulsionadoras da DRC (p.e. diabetes e hipertensão) para cada paciente, e não foram considerados pacientes com outras doenças progressivas como HIV ou doenças cardiovasculares, devido ao número diminuto da amostra obtida.

Variables			Stages				Chi-square	p value
			Stage 1	Stage 2	Stage 3	Stage 4		
Sex	Male	Count	8	13	21	24	1.213	0.750
		percentage	(12.1%)	(19.7%)	(31.8%)	(36.4%)		
	Female	Count	9	7	16	19		
		percentage	(17.6%)	(13.7%)	(31.4%)	(37.3%)		
HTN	No	Count	11	6	14	11	8.483	0.037
		percentage	(26.2%)	(14.3%)	(33.3%)	(26.2%)		
	Yes	Count	6	14	23	32		
		percentage	(8%)	(18.7%)	(30.7%)	(42.7%)		
Diabetes	No	Count	17	5	15	18	24.131	< 0.001
		percentage	(30.9%)	(9.1%)	(27.3%)	(32.7%)		
	Yes	Count	0	15	22	25		
		percentage	(0%)	(24.2%)	(35.5%)	(40.3%)		
Total		Count	17	20	37	43		
		percentage	(14.5%)	(17.1%)	(31.6%)	(36.8%)		

HTN: Hypertension.

Tabela 1 – Tabela de estatísticas relativas aos pacientes

A tabela 2 contabiliza o número de transições entre estados, por parte dos pacientes em questão. As transições foram registadas nas visitas ao médico e a frequência de visitas aumenta com o estado de gravidade da doença.

	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5
Stage 1	75	8	4	3	0
Stage 2	0	116	6	3	1
Stage 3	0	0	293	29	2
Stage 4	0	0	0	145	51

Tabela 2 – Número de transições entre estados

Através destes dados, é possível tirar algumas conclusões: o número de transições para o mesmo estado, ou seja, a permanência no estado atual é muito elevada, uma vez que a progressão da DRC é muito lenta; o número de transições do estado 4 para o 5 é muito superior ao de outros estados para o seguinte, o que sugere que a doença age muito mais depressa nos últimos estados.

2.2.5. Cálculo da Função de Probabilidade

Sendo $S(t) = k$ o estado do paciente num instante t, tem-se que a **intensidade de transição/taxa de transição instantânea** com que um paciente se move para o estado l durante o intervalo de tempo $(t+\delta t)$ é dada por:

$$\lambda_{kl} = \lim_{\delta t \to 0} \frac{P(S(t+\delta t) = l \mid S(t) = k)}{\delta t}$$

Figura 4 - Fórmula da intensidade de transição entre estados

Isto sugere a matriz de intensidades de transição entre estados apresentada abaixo, que é caracterizada por (1) a soma dos elementos de cada linha ser nula e (2) o elemento diagonal de uma linha ser o oposto da soma dos restantes elementos da linha.

$$Q = \begin{pmatrix} -(\lambda_{12}+\lambda_{13}+\lambda_{14}+\lambda_{15}) & \lambda_{12} & \lambda_{13} & \lambda_{14} & \lambda_{15} \\ 0 & -(\lambda_{23}+\lambda_{24}+\lambda_{25}) & \lambda_{23} & \lambda_{24} & \lambda_{23} \\ 0 & 0 & -(\lambda_{34}+\lambda_{35}) & \lambda_{34} & \lambda_{35} \\ 0 & 0 & 0 & -\lambda_{45} & \lambda_{45} \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figura 5 - Matriz de intensidades de transição entre estados

Num modelo de Markov normal, assume-se que a probabilidade de transição para um estado futuro depende unicamente do estado atual. Contudo, no caso de modelos de Markov homogéneos e contínuos no tempo, o tempo de permanência num estado k tem uma distribuição exponencial de taxa $-\lambda_{kk}$. Logo, a probabilidade de um indivíduo passar de um estado k para um estado l é dada por $-\lambda_{kl}/\lambda_{kk}$, sendo os restantes elementos da linha k proporcionais.

Através de equações diferenciais de Kolmogorov, modelou-se a progressão da DRC, obtendo-se os seguintes resultados:

$$p_{11}(s,t) = e^{\lambda_{11}(t-s)}$$

$$p_{12}(s,t) = \frac{\lambda_{12}}{\lambda_{22}-\lambda_{11}}(e^{\lambda_{11}(t-s)} - e^{\lambda_{22}(t-s)})$$

.

.

.

.

.

.

$$p_{44}(s,t) = e^{\lambda_{44}(t-s)}$$

$$p_{45}(s,t) = 1 - e^{\lambda_{44}(t-s)}$$

$$p_{55}(s,t) = 1.$$

Figura 6 - Fórmulas das probabilidades de transições entre estados

A **função de probabilidade** é o produto das probabilidades de transição entre estados de todos os indivíduos e períodos observados, pelo que se obtém:

$$L(Q) = \prod_i L_i = \prod_{i,j} L_{i,j} = \prod_{i,j} P_{S(t_{ij})S(t_{i,j+1})}(t_{i,j+1} - t_{ij})$$

Figura 7 - Fórmula dos elementos da matriz de transição

$L_{i,j}$ é o elemento correspondente à linha $S(t_{ij})$ e coluna $S(t_{i,j+1})$ da matriz de transição $P(t)$, no intervalo de tempo $(t_{i,j+1} - t_{ij})$. A probabilidade $L(Q)$ é maximizada com $log(\lambda_{kl})$, pelo que as estimativas e erro-padrão de λ_{kl} são obtidas por $log(\lambda_{kl})$.

2.2.6. Resultados

Foi calculada a matriz de probabilidades de transição entre estados para um período de 1, 5 e 10 anos no futuro.

	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5
after one year					
Stage 1	0.9067	0.0538	0.0258	0.0106	0.0031
Stage 2	0	0.8876	0.0645	0.0367	0.0112
Stage 3	0	0	0.8915	0.0825	0.0260
Stage 4	0	0	0	0.6689	0.3311
Stage 5	0	0	0	0	1
after five years					
Stage 1	0.6126	0.1741	0.1094	0.0401	0.0638
Stage 2	0	0.5508	0.2019	0.0828	0.1645
Stage 3	0	0	0.5631	0.1288	0.3082
Stage 4	0	0	0	0.0596	0.9404
Stage 5	0	0	0	0	1
after ten years					
Stage 1	0.3753	0.2026	0.1638	0.0554	0.2029
Stage 2	0	0.3033	0.2249	0.0765	0.3952
Stage 3	0	0	0.3170	0.0802	0.6028
Stage 4	0	0	0	0.0036	0.9964
Stage 5	0	0	0	0	1

Tabela 3 - Probabilidades estimadas de transição após 1 ano, 5 anos e 10 anos

Analisando estes resultados, é possível concluir que a probabilidade de um paciente permanecer no mesmo estado após um ano é muito alta, o que confirma que a progressão da doença é bastante lenta, salva a exceção do estádio 4. Contudo, é possível observar que essa probabilidade diminui ao longo do tempo, o que sugere um declínio irreversível e cada vez mais acentuado da doença, caso não comece a ser tratada cedo.

Também foram determinados os tempos médios de permanência em cada estado, apresentados na seguinte tabela.

	Estimates	SE	CI
Stage 1	10.2030	2.6395	(6.1450, 16.9407)
Stage 2	8.3829	2.6501	(4.5112, 15.5772)
Stage 3	8.7051	1.5645	(6.1206, 12.3808)
Stage 4	1.7734	0.2456	(1.3517, 2.3265)

Tabela 4 – Tempos médios de permanência em estados

Os tempos de permanência nos estados 1, 2 e 3 são bastante altos, o que prova a lenta progressão da DRC, enquanto que no estado 4 é 1.7734, que é bastante baixo em comparação, o que corrobora o rápido declínio da condição dos rins nas fases finais.

As conclusões obtidas são consistentes com o consenso existente à volta da Doença Renal Crónica e os dados recolhidos, e reiteram o facto de a hipertensão e os diabetes serem fatores prominentes para o estabelecimento e progressão da DRC.

2.2.7. Referências

GROVER, G., SABHARWAL, A., KUMAR, S., & THAKUR, A. K. (2019). A Multi-State Markov Model for the Progression of Chronic Kidney Disease. Turkiye Klinikleri Journal of Biostatistics, 11(1), 1–14. https://doi.org/10.5336/biostatic.2018-62156

3. Anexos

3.1. Anexo A1

Grupo que inclui o Aluno com o Nº 85006

MEIO-TP1 - Tabelas de probabilidades de pedidos e entregas de automóveis

FILIAL 1

Número de clientes: ; 0 ; 1 ; 2 ; 3 ; 4 ; 5 ; 6 ; 7 ; 8 ; 9 ; 10 ; 11 ; 12

Probabilidade (pedidos): ; 0.0476 ; 0.1388 ; 0.2252 ; 0.2260 ; 0.1644 ; 0.1044 ; 0.0564 ; 0.0240 ; 0.0100 ; 0.0024 ; 0.0004 ; 0.0004 ; 0.0000

Probabilidade (entregas): ; 0.0412 ; 0.0908 ; 0.1092 ; 0.1332 ; 0.1376 ; 0.1180 ; 0.1092 ; 0.0832 ; 0.0612 ; 0.0448 ; 0.0380 ; 0.0244 ; 0.0092

FILIAL 2

Número de clientes: ; 0 ; 1 ; 2 ; 3 ; 4 ; 5 ; 6 ; 7 ; 8 ; 9 ; 10 ; 11 ; 12

Probabilidade (pedidos): ; 0.0444 ; 0.0992 ; 0.1288 ; 0.1444 ; 0.1224 ; 0.1068 ; 0.0968 ; 0.0792 ; 0.0612 ; 0.0488 ; 0.0364 ; 0.0220 ; 0.0096

Probabilidade (entregas): ; 0.0280 ; 0.0656 ; 0.0904 ; 0.1320 ; 0.1508 ; 0.1268 ; 0.1124 ; 0.0928 ; 0.0720 ; 0.0608 ; 0.0356 ; 0.0240 ; 0.0088

3.2. Anexo A2

[As matrizes de transição (de probabilidades e contribuições) estarão no ficheiro excel anexado com o restante trabalho]

3.3. Anexo A3

```
import org.apache.poi.ss.usermodel.*;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

import java.io.File;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class Meio {
    /**
     * Para cada filial, retorna a matriz de transição respetiva (13x13) para o caso
base de não existirem transações entre as filiais.
     * @param pedidos probabilidade de pedidos para 0...12 clientes
     * @param entregas probabilidade de entregas para 0...12 clientes
     * @return matriz de transição 13x13 para uma filial
     */
    private static double[][] filialTransicao(double[] pedidos, double[] entregas){
        int i,j,x,y;
        double prob,sum;
        double[][] probs = new double[13][13];

        for(i = 0; i<13;i++){
            for(j=0; j<13; j++){
                prob = 0;
                probs[i][j] = 0;
                sum=0;

                for(x = i>=j ? 0 : j-i,y = i>=j ? i-j : 0; x <= j-1 && y <= i-1 ; x++,
y++){
                    prob += entregas[x]*pedidos[y];
                }

                for(x=i; x<13; x++){
                    sum+=pedidos[x];
                }
                prob+=sum*entregas[j];

                if(j==12) {
                    for(x=0; x<i ; x++) {
                        sum = 0;
                        for (y = j - i + x + 1; y < 13; y++)
                            sum += entregas[y];
                        prob += pedidos[x] * sum;
                    }
                }

                probs[i][j] = prob;
            }
        }
        return probs;
    }
```

```
    /**
     * Junta as duas matrizes de transição de ambas as filiais numa de 169x169
     * @param filial1 matriz de transição calculada para a primeira filial
     * @param filial2 matriz de transição calculada para a segunda filial
     * @return matriz 169x169 com a junção de ambas as filiais relativas a uma transferência
     */
    private static double[][] juntaTransicoesFiliais(double[][] filial1, double[][] filial2){
        double[][] r = new double[169][169];
        for (int i = 0; i < 169; i++)
            for (int j = 0; j < 169; j++)
                r[i][j] = filial1[i / 13][j / 13] * filial2[i % 13][j % 13];
        return r;
    }

    /**
     * Efetua a transferência entre as duas filiais, que são os seguintes casos: +0/-0, +1/-1, +2/-2, +3/-3, -1/+1, -2/+2, -3/+3
     * @param filial1 matriz 13x13 do caso inicial (+0/-0) da filial 1
     * @param filial2 matriz 13x13 do caso inicial (+0/-0) da filial 2
     * @param troca Transferir/Receber de 1,2 ou 3 carros: se for negativo, transferem-se da filial 1 para a 2, se for positivo é o contrário
     * @return lista com as novas matrizes resultantes das transferências
     */
    private static double[][][] transfereEntreFiliais(double[][] filial1, double[][] filial2, int troca){
        int i,j,x = 0,indice;
        double[][] m1 = new double[13][13];
        double[][] m2 = new double[13][13];
        double[][][] r = {m1,m2};

        if(troca<0){
            indice = Math.abs(troca);
            for(i = 0; i < indice; i++){
                for(j = 0; j < 13; j++){
                    m1[i][j] = m2[12-i][j] = 999999;
                }
            }
            for(i = indice; i < 13; i++){
                for(j = 0; j < 13; j++){
                    m1[i][j] = filial1[x][j];
                    m2[x][j] = filial2[i][j];
                }
                x++;
            }
        }
        else if(troca>0){
            for(i = 0; i < troca; i++){
                for(j = 0; j < 13; j++){
                    m1[12-i][j] = m2[i][j] = 999999;
                }
            }
            for(i = troca; i < 13; i++){
                for(j = 0; j < 13; j++){
                    m1[x][j] = filial1[i][j];
```

```
                    m2[i][j] = filial2[x][j];
                }
                x++;
            }
        }
        return r;
    }

    /**
     * Grava toda a matriz de transição 169x169 resultante da junção das duas matrizes de transição de ambas as filiais num ficheiro Excel
     * @param transicoes são as 7 Matrizes 169x169 de transição resultante da junção das duas matrizes de transição de ambas as filiais
     * @param contribuicoes são as 7 Matrizes 169x169 de contribuições resultante da junção das duas matrizes de transição de ambas as filiais
     * @param ficheiro nome a dar ao ficheiro
     */
    private static void gravaTransicoesExcel(List<double[][]> transicoes, List<double[][]> contribuicoes, String ficheiro) throws Exception {
        File file = new File("./"+ficheiro);
        file.createNewFile();
        FileOutputStream fos;
        XSSFWorkbook wb = new XSSFWorkbook();
        List<XSSFSheet> tsheets = new ArrayList<>();
        List<XSSFSheet> csheets = new ArrayList<>();
        XSSFSheet sh;
        Row row;
        Cell cell;
        tsheets.add(wb.createSheet("Transicao 0|0"));
        tsheets.add(wb.createSheet("Transicao -3|+3"));
        tsheets.add(wb.createSheet("Transicao -2|+2"));
        tsheets.add(wb.createSheet("Transicao -1|+1"));
        tsheets.add(wb.createSheet("Transicao +1|-1"));
        tsheets.add(wb.createSheet("Transicao +2|-2"));
        tsheets.add(wb.createSheet("Transicao +3|-3"));

        csheets.add(wb.createSheet("Contribuicao 0|0"));
        csheets.add(wb.createSheet("Contribuicao -3|+3"));
        csheets.add(wb.createSheet("Contribuicao -2|+2"));
        csheets.add(wb.createSheet("Contribuicao -1|+1"));
        csheets.add(wb.createSheet("Contribuicao +1|-1"));
        csheets.add(wb.createSheet("Contribuicao +2|-2"));
        csheets.add(wb.createSheet("Contribuicao +3|-3"));

        for(int t = 0; t < 7; t++){
            sh = wb.getSheetAt(t);
            for(int i = 0; i < 169; i++){
                row = sh.createRow(i);
                for(int j = 0; j < 169; j++){
                    cell = row.createCell(j);
                    cell.setCellValue(transicoes.get(t)[i][j]);
                }
            }
        }

        for(int t = 0; t < 7; t++){
            sh = wb.getSheetAt(t+7);
```

```
        for(int i = 0; i < 169; i++){
            row = sh.createRow(i);
            for(int j = 0; j < 169; j++){
                cell = row.createCell(j);
                cell.setCellValue(contribuicoes.get(t)[i][j]);
            }
        }
    }

    fos = new FileOutputStream(file);
    wb.write(fos);
    fos.close();
}

/**
 * Calcula as contribuições base para ambas as filiais
 * @param pedidos probabilidade de pedidos para 0...12 clientes
 * @param entregas probabilidade de entregas para 0...12 clientes
 * @return matrix 13x13 com as contribuições para uma filial
 */
private static double[][] calculaContribuiçoes(double[] pedidos, double[] entregas){
    int i,j,x,y;
    double sum;
    double[][] r = new double[13][13];

    for(i = 0; i<13;i++){
        for(j=0; j<13; j++){
            r[i][j] = 0;
            sum=0;
            for(x=i>=j?0:j-i, y=i>=j?i-j:0;x<=j-1 && y <= i-1;x++, y++ ){
                r[i][j] += entregas[x]*pedidos[y]*30*y;
            }
            for(x=i ; x<13; x++ ){
                sum+=pedidos[x];
            }
            r[i][j] += sum*i*30 * entregas[j];
            if(j>8){
                r[i][j]-=10;
            }
            if(j==12) {
                for(x=0; x<i ; x++) {
                    sum = 0;
                    for (y = j - i + x + 1; y < 13; y++)
                        sum += entregas[y];
                    r[i][j] += pedidos[x] * sum * x * 30;
                }
            }
        }
    }
    return r;
}

/**
 * Junta as contribuições de duas filiais numa matriz 169x169
 * @param filial1 contribuições da filial 1
 * @param filial2 contribuições da filial 2
 * @param troca total a retirar por causa do número de trocas
```

```
     * @return matriz 169x169 com as contribuições
     */
    private static double[][] juntaContribuicoes(double[][] filial1, double[][] filial2,
int troca){
        int retira = Math.abs(troca)*7;
        double[][] r = new double[169][169];
        for(int i = 0; i < 169; i++){
            for (int j = 0; j < 169; j++) {
                r[i][j] = filial1[i / 13][j / 13] + filial2[i % 13][j % 13] - retira;
            }
        }
        return r;
    }

    /**
     * Efetua o cálculo das matrizes de contribuição resultantes das transferências en-
tre as duas filiais, que são os seguintes casos: +0/-0, +1/-1, +2/-2, +3/-3, -1/+1, -
2/+2, -3/+3
     * @param filial1 matriz 13x13 do caso inicial (+0/-0) da filial 1
     * @param filial2 matriz 13x13 do caso inicial (+0/-0) da filial 2
     * @param troca Transferir/Receber de 1,2 ou 3 carros: se for negativo, transferem-
se da filial 1 para a 2, se for positivo é o contrário
     * @return lista com as novas matrizes de contribuição resultantes das transferên-
cias
     */
    private static double[][][] contribuicoesTransferencia(double[][] filial1, dou-
ble[][] filial2, int troca){
        int i,j,x = 0,indice;
        double[][] m1 = new double[13][13];
        double[][] m2 = new double[13][13];
        double[][][] r = {m1,m2};
        if(troca<0){
            indice = Math.abs(troca);
            for(i = 0; i < indice; i++){
                for(j = 0; j < 13; j++){
                    m1[i][j] = m2[12-i][j] = -999999;
                }
            }
            for(i = indice; i < 13; i++){
                for(j = 0; j < 13; j++){
                    m1[i][j] = filial1[x][j];
                    m2[x][j] = filial2[i][j];
                }
                x++;
            }
        }
        else if(troca>0){
            for(i = 0; i < troca; i++){
                for(j = 0; j < 13; j++){
                    m1[12-i][j] = m2[i][j] = -999999;
                }
            }
            for(i = troca; i < 13; i++){
                for(j = 0; j < 13; j++){
                    m1[x][j] = filial1[i][j];
                    m2[i][j] = filial2[x][j];
                }
```

```
                x++;
            }
        }
        return r;
    }

    /**
     * Realiza o processo iterativo para obter o resultado final
     * @param n Número de iterações a realizar
     * @param k Número de transições possíveis
     * @param transicoes As K matrizes de transição 169x169
     * @param contribuicoes as K matrizes de contribuições 169x169
     */
    private static void converge(int n, int k, List<double[][]> transicoes, List<dou-
ble[][]> contribuicoes){
        int f = 0, indice;
        double sum, max;
        List<double[]> q = new ArrayList<>();
        List<double[]> v;
        double[] aux;
        double[] f_ant;
        double[] f_atual = new double[169];
        double[] d = new double[169];
        double[] aux2 = new double[k];
        int[] decisao = new int[169];

        for(int x = 0; x < k; x++){ // calculo das esperanças Q %% NAO MEXER %%
            aux = new double[169];
            for(int i = 0; i < 169; i++){
                for (int j = 0; j < 169; j++) {
                    aux[i] += contribuicoes.get(x)[i][j] * transicoes.get(x)[i][j];
                }
            }
            q.add(aux);
        }

        for(int it = 0; it < n; it++){ // para as 50 iteraçoes
            f_ant = Arrays.copyOf(f_atual,169);
            v = new ArrayList<>();
            for(int x = 0; x < k; x++){ // para os k = 7
                aux = new double[169];
                for(int i = 0; i < 169; i++){ // para a matriz P^K
                    sum=0;
                    for(int j = 0; j < 169; j++){
                        sum+= transicoes.get(x)[i][j] * f_ant[j];
                    }
                    aux[i] = sum + q.get(x)[i];
                }
                v.add(aux);
            }
            for(int y = 0; y < 169; y++){
                for(int x = 0; x < k; x++){
                    aux2[x] = v.get(x)[y];
                }
                indice = 0;
                max = -999999;
                for(int i = 0; i < k; i++){
```

```
                    if(aux2[i] > max){
                        max = aux2[i];
                        indice = i;
                    }
                }
                f_atual[y] = max;
                if(it==n-1) decisao[f++]=indice;
            }
            for(int i = 0; i < 169; i++){
                d[i] = f_atual[i] - f_ant[i];
            }
        }
        for(int i = 0; i < 169; i++){
            System.out.print(d[i] + ","); // para mostrar a convergência final
        }
        System.out.println();
        for(int i = 0; i < 13; i++){
            System.out.print("Linha " + i + ": ");
            for(int j = 0; j < 13; j++){
                System.out.print(decisao[13*i+j] + ", ");
            }
            System.out.println();
        }

    }

    public static void main(String... args) throws Exception{
        double[] pedidos1 = {0.0476 , 0.1388 , 0.2252 , 0.2260 , 0.1644 , 0.1044 ,
0.0564 , 0.0240 , 0.0100 , 0.0024 , 0.0004 , 0.0004 , 0.0000};
        double[] entregas1 = {0.0412 , 0.0908 , 0.1092 , 0.1332 , 0.1376 , 0.1180 ,
0.1092 , 0.0832 , 0.0612 , 0.0448 , 0.0380 , 0.0244 , 0.0092};
        double[] pedidos2 = {0.0444 , 0.0992 , 0.1288 , 0.1444 , 0.1224 , 0.1068 ,
0.0968 , 0.0792 , 0.0612 , 0.0488 , 0.0364 , 0.0220 , 0.0096};
        double[] entregas2 = {0.0280 , 0.0656 , 0.0904 , 0.1320 , 0.1508 , 0.1268 ,
0.1124 , 0.0928 , 0.0720 , 0.0608 , 0.0356 , 0.0240 , 0.0088};

        double[][] transicao1 = filialTransicao(pedidos1,entregas1);
        double[][] transicao2 = filialTransicao(pedidos2,entregas2);
        double[][] cont1 = calculaContribuiçoes(pedidos1,entregas1);
        double[][] cont2 = calculaContribuiçoes(pedidos2,entregas2);
        double[][] contJuntas;
        double[][] transJuntas;
        double[][][] transita;
        double[][][] contribui;
        List<double[][]> gravarTransicoes = new ArrayList<>();
        List<double[][]> gravarContribuicoes = new ArrayList<>();
        gravarTransicoes.add(juntaTransicoesFiliais(transicao1,transicao2));
        gravarContribuicoes.add(juntaContribuicoes(cont1,cont2,0));
        for(int troca = 3; troca > -4; troca--){
            if(troca!=0){
                transita = transfereEntreFiliais(transicao1,transicao2,troca);
                contribui = contribuicoesTransferencia(cont1,cont2,troca);

                transJuntas = juntaTransicoesFiliais(transita[0],transita[1]);
                contJuntas = juntaContribuicoes(contribui[0], contribui[1],troca);

                gravarTransicoes.add(transJuntas);
```

```
                gravarContribuicoes.add(contJuntas);
            }
        }
        gravaTransicoesExcel(gravarTransicoes,gravarContribuicoes,"matrizes.xlsx");
        converge(100,7,gravarTransicoes,gravarContribuicoes);
    }
}
```

3.4. Anexo A4

Como a nossa função `main` junta todas as outras funções, não precisamos de dar nenhum input ao nosso programa. O único output que pusemos foi para determinar o resultado final (o maior lucro e a política ótima):

```
42.56827974643875,42.568279746443295,42.56827974644329
Linha 0: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
Linha 1: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0,
Linha 2: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 4, 0,
Linha 3: 0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 5, 4, 0,
Linha 4: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 4, 0,
Linha 5: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0,
Linha 6: 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
Linha 7: 0, 0, 2, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
Linha 8: 0, 3, 2, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
Linha 9: 0, 3, 2, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
Linha 10: 0, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0,
Linha 11: 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 0,
Linha 12: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```