UNIVERSIDADE FEDERAL DE SANTA CATARINA

Departamento de Automação e Sistemas
Engenharia de Controle e Automação
DAS5103 - Cálculo Numérico para Controle e Automação

Trabalho Final

Alunos:
Julia Gazolla
Nicolas Antero Nunes
Pollyana Gomes Minatel
Victor Bergamin Biazi

Professor:
Eduardo Camponogara

Florianópolis, Brasil
2021

Sumário

1 Introdução **2**

2 Questão 1 **3**

2.1 Problema: 3

2.2 Solução: 4

3 Questão 2 **6**

3.1 Problema: 6

3.2 Solução: 6

4 Questão 3 **9**

4.1 Problema: 9

4.2 Solução: 10

5 Referências **15**

1 Introdução

Este documento apresenta a proposta de resolução do grupo para o trabalho final de Calculo Numérico, cursado no semestre de 2020.2.

A orientação para o trabalho é: "Cada grupo de estudantes deverá identificar de três a quatro problemas relacionados à disciplina de Cálculo Numérico, formalizar o problema, apresentar e ilustrar a solução de cada problema. No caso da solução implicar a síntese de um algoritmo, este deverá ser implementado e aplicado. As iterações do algoritmo deverão ser ilustradas com tabelas, gráficos e outros meios que favorecem o entendimento.

Pelo menos um problema deverá ser definido dentro de cada uma das seguintes áreas:

- Solução de equação não-linear, sistemas de equações não-lineares, ou sistemas de equações lineares.
- Método de mínimos quadrado e aplicações (e.g., identificação de sistemas e suavização de curvas) ou minimização de norma e aplicações (controle de sistemas dinâmicos).
- Otimização matemática, solução numérica de sistemas de equações diferenciais ordinárias, ou integração numérica. "

2 Questão 1

2.1 Problema:

Os índices de crescimento populacional são um importante indicativo da realidade da população em uma região. Com advento da pandemia, estudos já indicam que haverá uma redução da expectativa de vida da população brasileira. Esse índice é obtido em uma relação da média ponderada das idades das pessoas do lugar que morreram naquele ano. Ou ainda, pode ser chamado de expectativa de vida ao nascer, que apenas considera a expectativa de sobrevida da população residente na região.

Sendo assim, utilize os dados fornecidos pelo IBGE sobre a população residente no Brasil entre 1872 (primeiro CENSO nacional) e 2019 para calcular qual seria a população total em 2025, sem a pandemia. Além disso, sugira qual seria a expectativa de vida ao nascer nesse ano. Utilize os dados de 2019 apresentados a seguir.

ANO	POPULAÇÃO
1872	9.930.478
1890	14.333.915
1900	17.400.000
1920	30.635.605
1940	41.200.000
1950	51.944.397
1960	70.992.343
1970	93.000.000
1980	121.150.573
1991	146.800.000
2000	169.590.693
2010	190.755.799
2019	210.147.125

Tabela 1: População Brasileira de 1872 até 2019

IDADE	EXPECTATIVA DE VIDA
0	76.6
1	76.5
5	72.6
10	67.7
15	62.8
20	58.1
25	53.5
30	48.9
35	44.3
40	39.7
45	35.2
50	30.8
55	26.7
60	22.7
65	18.9
70	15.5
75	12.4
80+	9.7

Tabela 2: Expectativas de vida da população brasileira em 2019, por idade

2.2 Solução:

Para melhor visualizar os dados apresentados na tabela, estes foram expostos em um gráfico:

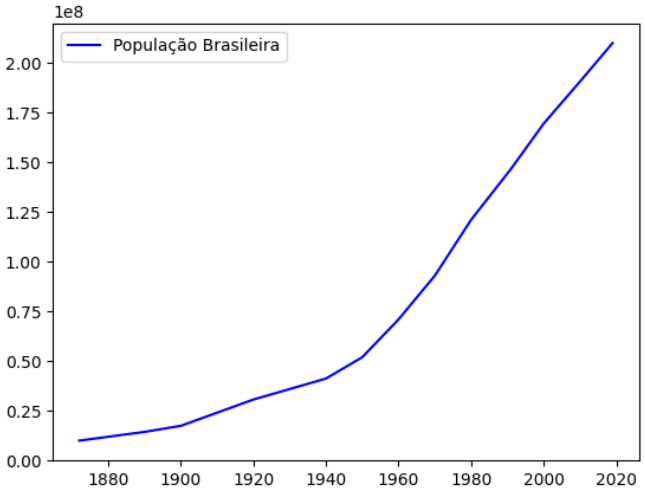

Figura 1: Gráfico dos dados apresentados na Tabela 1

Para obter dados futuros a partir de amostras conhecidas, podemos usar o Método de Mínimos Quadrados. Para isso, considera-se que os dados acima podem ser representados pela seguinte equação exponencial:

$$y = A * e^{Bx} \tag{1}$$

No Método, a equação foi linearizada, tal que:

$$ln(y) = ln(A) + Bx \tag{2}$$

Agora, os parâmetros dos mínimos quadrados estão linearizados e podem ser aplicados no algoritmo com grau 1.

Os resultados apresentados utilizaram o seguinte código em Python:

```
import numpy as np
import math
import matplotlib.pyplot as plt

#Anos com populacao conhecidas - 1872 primeiro censo
anos_br = [1872, 1890, 1900, 1920, 1940, 1950 ,
           1960, 1970, 1980, 1991, 2000, 2010, 2019]
#Numero de populacao conhecida - Fonte: IBGE
brasil = [9930478, 14333915, 17400000, 30635605, 41200000, 51944397,
          70992343, 93000000, 121150573, 146800000, 169590693,
          190755799, 210147125]

#note que o coportamento desses valores    exp, ou seja y = A*exp(x*B)
#ajuste de dados para linearizar: log y = log a + bx
brasil_ln = np.zeros(len(brasil))
for i in range (len(brasil)):
    brasil_ln[i] = np.log(brasil[i])

plt.plot(anos_br, brasil,color='blue', label='Popula  o Brasileira')
plt.legend()
```

```
plt.show()

def projecao(A,B,x):#calculo de projecoes futuras
    y = A * np.exp(B*x)
    print("A projecao para " + str(x) + " = ")
    print(y)

def log(x,y_ln,a_ln, b): #fun   o de verifica  o

    y_teste = np.zeros(len(x))
    for i in range (len(x)):
        y_teste[i] = a_ln + b*x[i]
    print(y_ln)
    print(y_teste)

def min_quadrados(X,Y,pontos, grau): #como foi linearizado, usamos grau = 1
 #organiza  o de dados e calculo do minimos quadrados
    H = np.zeros((grau+1,pontos))
    for i in range(len(H)):
        for j in range(len(H[0])):
            H[i][j] = pow(X[j],i)

    A = np.zeros((grau+1,grau+1))
    b = np.zeros(grau+1)
    for i in range(len(A)):
        for j in range(len(A)):
            A[i][j]=H[i].dot(H[j])
        b[i] = H[i].dot(Y)

 #resolvendo os dados organizados por minimos quadrados
    x = np.linalg.solve(A,b)
    log(X, Y, x[0], x[1]) #testando se os valores estao prox aos dados
    x[0] = np.exp(x[0])#Ajustando logA para A
    print("A = " + str(x[0])) #coef A
    print("b = " + str(x[1])) #coef B

    projecao(x[0],x[1],2025)

min_quadrados(anos_br,brasil_ln,len(brasil_ln),1)
```

Sendo assim, a projeção da população brasileira em 2025 é aproximadamente 282.80 milhões.

Para obter a expectativa de vida em 2025, baseado em dados da Tabela 2, o método de mínimos quadrados também fora aplicado, porém considerando uma equação polinomial de grau 3. Essa equação foi escolhida com base em estudos de expectativa apresentado nas referências.

```
import numpy as np
import math
import matplotlib.pyplot as plt

idades = [0, 1, 5, 10, 15, 20, 25 , 30, 35, 40,
          45,50, 55, 60, 65, 70, 75, 80]

expectativa = [ 76.6,  76.5, 72.6, 67.7, 62.8, 58.1, 53.5, 48.9, 44.3, 39.7,
                35.2, 30.8, 26.7, 22.7, 18.9,15.5, 12.4,9.7]


def projecao(A0,A1,A2,A3,x):
 #equacao de grau 3: A0 + A1x + A2x^2 + A3x^3
    y = A0 + A1*x + A2*(x**2) + A3*(x**3)
    x = 2019 - x
    print("A projecao para " + str(x) + " = ")
    print(y)

def log(x,y_ln,a_ln, b):

    y_teste = np.zeros(len(x))
    for i in range (len(x)):
```

```
        y_teste[i] = a_ln + b*x[i]
    print(y_ln)
    print(y_teste)


def min_quadrados(X,Y,pontos, grau): #para exp vida: POLINOMIO DE GRAU 3
 #organiza  o de dados e calculo do minimos quadrados
    H = np.zeros((grau+1,pontos))
    for i in range(len(H)):
        for j in range(len(H[0])):
            H[i][j] = pow(X[j],i)

    A = np.zeros((grau+1,grau+1))
    b = np.zeros(grau+1)
    for i in range(len(A)):
        for j in range(len(A)):
            A[i][j]=H[i].dot(H[j])
        b[i] = H[i].dot(Y)

 #resolvendo os dados organizados por minimos quadrados
    x = np.linalg.solve(A,b)
    print("A0 = " + str(x[0]))
    print("A1 = " + str(x[1]))
    print("A2 = " + str(x[2]))
    print("A3 = " + str(x[3]))

 #x    negativo pois queremos saber dados anteriores a  2019 (2019+6)
    projecao(x[0],x[1],x[2], x[3], -6)

min_quadrados(idades, expectativa, len(expectativa), 3)
```

Ou seja, pelo método aplicado na linguagem Python, podemos dizer que a expectativa de vida ao nascer em 2025 é de 82 anos.

3 Questão 2

3.1 Problema:

A Decathlon possui dados sobre duas competições realizadas em 2004, Decastar e Jogos Olímpicos. Os dados possuem a pontuação dos participantes em 10 provas e suas classificações. Faça um estudo das componentes principais escolhendo um número adequado de componentes para representar todos os dados e, depois, proponha uma nova classificação.

3.2 Solução:

Como se trata de uma mesma planilha de dados com duas competições, existem participantes repetidos. Apesar disso, o estudo considerou que todas as observações (linhas) são independentes.

Primeiramente importamos as bibliotecas e o dataframe utilizados:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

df = pd.read_csv("dados_decathlon.csv", index_col=0)
```

Para as primeiras 5 linhas temos:

Athlets	100m	Long.jump	Shot.put	High.jump	400m	110m.hurdle	Discus	Pole.vault	Javeline	1500m	Rank	Points	Competition
SEBRLE	11.04	7.58	14.83	2.07	49.81	14.69	43.75	5.02	63.19	291.7	1	8217	Decastar
CLAY	10.76	7.40	14.26	1.86	49.37	14.05	50.72	4.92	60.15	301.5	2	8122	Decastar
KARPOV	11.02	7.30	14.77	2.04	48.37	14.09	48.95	4.92	50.31	300.2	3	8099	Decastar
BERNARD	11.02	7.23	14.25	1.92	48.93	14.99	40.87	5.32	62.77	280.1	4	8067	Decastar
YURKOV	11.34	7.09	15.19	2.10	50.42	15.31	46.26	4.72	63.44	276.4	5	8036	Decastar

Então, calculamos a média de pontos de cada prova:

```
# media por coluna
colunas = ['100m', 'Long.jump', 'Shot.put', 'High.jump', '400m', '110m.hurdle', 'Discus',
    'Pole.vault', 'Javeline', '1500m']
media_colunas = np.mean(df[colunas], axis=0)
print(media_colunas)
```

Resultando em:

Prova	Média de pontos
100m	10.998049
Long.jump	7.260000
Shot.put	14.477073
High.jump	1.976829
400m	49.616341
110m.hurdle	14.605854
Discus	44.325610
Pole.vault	4.762439
Javeline	58.316585
1500m	279.024878

Agora, é necessário uma nova matriz contendo os pontos de cada participante menos a média de pontos por prova:

```
# subtraindo a media da respectiva coluna
M = df[colunas] - media_colunas
```

Resultando em, para as primeiras 5 linhas:

Athlets	100m	Long.jump	Shot.put	High.jump	400m	110m.hurdle	Discus	Pole.vault	Javeline	1500m
SEBRLE	0.041951	0.32	0.352927	0.093171	0.193659	0.084146	-0.57561	0.257561	4.873415	12.675122
CLAY	-0.238049	0.14	-0.217073	-0.116829	-0.246341	-0.555854	6.39439	0.157561	1.833415	22.475122
KARPOV	0.021951	0.04	0.292927	0.063171	-1.246341	-0.515854	4.62439	0.157561	-8.006585	21.175122
BERNARD	0.021951	-0.03	-0.227073	-0.056829	-0.686341	0.384146	-3.45561	0.557561	4.453415	1.075122
YURKOV	0.341951	-0.17	0.712927	0.123171	0.803659	0.704146	1.93439	-0.042439	5.123415	-2.624878

Para calcularmos a matriz de covariância temos $C = \frac{1}{n-1} M^T M$, sendo n o número de linhas do dataframe. Aplicando isso em python:

```
# calculando matriz de covariancia
C = (M.T.dot(M)) / 40
```

Resultando em:

	100m	Long.jump	Shot.put	High.jump	400m	110m.hurdle	Discus	Pole.vault	Javeline	1500m
100m	0.069181	-0.049823	-0.077301	-0.005761	0.157850	0.071959	-0.196976	-0.006035	-0.200269	-0.185898
Long.jump	-0.049823	0.100110	0.047815	0.008292	-0.219725	-0.075445	0.207670	0.017945	0.182898	-0.124417
Shot.put	-0.077301	0.047815	0.679681	0.035875	-0.131641	-0.097867	1.714784	0.014022	1.492085	1.114460
High.jump	-0.005761	0.008292	0.035875	0.007912	-0.019284	-0.011888	0.110936	-0.003862	0.073796	-0.046624
400m	0.157850	-0.219725	-0.131641	-0.019284	1.330449	0.298207	-0.459279	-0.025426	0.023562	5.494956
110m.hurdle	0.071959	-0.075445	-0.097867	-0.011888	0.298207	0.222585	-0.519844	-0.000355	0.019910	0.206746
Discus	-0.196976	0.207670	1.714784	0.110936	-0.459279	-0.519844	11.409835	-0.140924	2.574275	10.179952
Pole.vault	-0.006035	0.017945	0.014022	-0.003862	-0.025426	-0.000355	-0.140924	0.077284	-0.040256	0.803008
Javeline	-0.200269	0.182898	1.492085	0.073796	0.023562	0.019910	2.574275	-0.040256	23.298193	-10.164190
1500m	-0.185898	-0.124417	1.114460	-0.046624	5.494956	0.206746	10.179952	0.803008	-10.164190	136.264701

Podemos notar que a matriz de covariância é uma matriz simétrica, de modo que sua diagonal principal consta com a variância de cada prova. O próximo passo é encontrar os autovalores e autovetores da matriz de covariância C.

Para encontrar os autovalores da matriz podemos calcular as raízes do polinômio característico $p(\lambda) = det(A - \lambda I)$. Já os autovetores são encontrados por $(A - I)v = 0$

Esses cálculos foram feitos em python com a ajuda de uma biblioteca:

```
# determinando auto-valores e auto-vetores
autovalores, autovetores = np.linalg.eig(C)
```

O resultado dos autovetores representam os pesos das 10 componentes principais (por coluna) para cada prova (por linha).

Para fazer uma análise de quantas componentes principais utilizar, podemos ver a variância explicada e a variância explicada cumulativa delas:

```
# calculando a variancia explicada e a variancia explicada cumulativa (influencia de cada
    componente principal)
total = sum(autovalores)
var_exp = [(i / total)*100 for i in sorted(autovalores, reverse=True)]
var_exp_cum = np.cumsum(var_exp)
```

Temos como resultado:

Variância explicada	**Variância explicada cumulativa**
79.65958964118825	79.65958964
13.552956464233063	93.21254611
5.759799777469904	98.97234588
0.6601788302826874	99.63252471
0.20550263725441784	99.83802735
0.07840365299103295	99.916431
0.03668769982762906	99.9531187
0.02918230455360301	99.98230101
0.014798319684874685	99.99709933
0.0029006725145513165	100

Podemos ver que as duas primeiras componentes principais explicam pouco mais de 93% dos dados. Dessa forma, iremos considerar apenas elas para a aplicação aos dados.

```
#pegando apenas as duas componentes principais
pesos_comp_principais = autovetores[:, 0:2]

#score das 2 componentes principais aplicados aos dados
P = pesos_comp_principais.T.dot(M.T)

#adicionando aos dados
PCA1 = P[0].tolist()
PCA2 = P[1].tolist()
df['PCA1'] = PCA1
df['PCA2'] = PCA2
```

Analisando os coeficientes de PCA1 e PCA2,

Coeficientes de PCA1	Coeficientes de PCA2
-1.28080400e-03	-1.11646237e-02
-9.50593693e-04	9.65383661e-03
8.04474909e-03	8.56736836e-02
-3.21942507e-04	4.28001111e-03
3.95670085e-02	9.81124318e-03
1.26099822e-03	-4.91153939e-03
7.79032346e-02	2.67357273e-01
5.70956842e-03	-1.11678946e-03
-8.59485398e-02	9.57662014e-01
9.92409609e-01	6.08751215e-02

podemos ver que o décimo coeficiente da PCA1 corresponde à prova de 1500m, que conta com pontuações de maior magnitude do que outras provas. Por isso, a análise de componente principais pode ter sofrido influência das pontuações não escalonadas. O mesmo ocorre com o nono coeficiente da PCA2, que corresponde à priva de lançamento de dardos (javeline).

Para conseguirmos uma nova classificação utilizaremos o PCA1, responsável por explicar quase 80% dos dados.

```
df.sort_values(by='PCA1',ascending=True)
```

Como resultado, os primeiros 5 classificados são:

Athlets	100m	Long.jump	Shot.put	High.jump	400m	110m.hurdle	Discus	Pole.vault	Javeline	1500m	Rank	Points	Competition	PCA1	PCA2
Lorenzo	11.10	7.03	13.22	1.85	49.34	15.38	40.22	4.50	58.36	263.08	24	7592	OlympicG	-16.168862	-2.144554
MARTINEAU	11.64	6.81	14.57	1.95	50.14	14.93	47.60	4.92	52.33	262.10	9	7802	Decastar	-16.004398	-5.888292
Smirnov	10.89	7.07	13.88	1.94	49.11	14.77	42.47	4.70	60.88	263.31	17	7993	OlympicG	-15.985132	0.944486
Hernu	10.97	7.19	14.65	2.03	48.73	14.25	44.72	4.80	57.76	264.35	7	8237	OlympicG	-14.518756	-1.313223
Macey	10.89	7.47	15.73	2.15	48.97	14.56	48.34	4.40	58.46	265.42	4	8414	OlympicG	-13.218942	0.488027

4 Questão 3

4.1 Problema:

Como escolher o melhor caminho para se chegar em um destino?

Sabendo de início o ponto de partida e de chegada, assim como informações históricas que nos ajudam a determinar as condições dos caminhos, podemos ter uma infinidade de possíveis caminhos para se chegar ao destino final. A decisão a ser feita depende do estado atual das estradas, relativo a condições de tráfego, limite de velocidade, semáforos, etc.

Temos como objetivo minimizar o tempo de trajeto de um aluno saindo de carro de sua casa com destino a Universidade Federal de Santa Catarina, considerando que as condições das estradas não se alteram, não há novas construções, as estradas não são fechadas e não há qualquer outro imprevisto que possa alterar a dinâmica das estradas. Dessa forma, temos dados suficientes para saber o que irá acontecer ao longo do caminho. Ainda, devemos nos atentar ao fato de que, a cada decisão de rota tomada, teremos uma mudança no estado atual, ou seja, o aluno terá novas decisões de rotas a serem tomadas, portanto a dinâmica de estados não é fixa.

A dinâmica do ambiente pode ser vista na figura 3

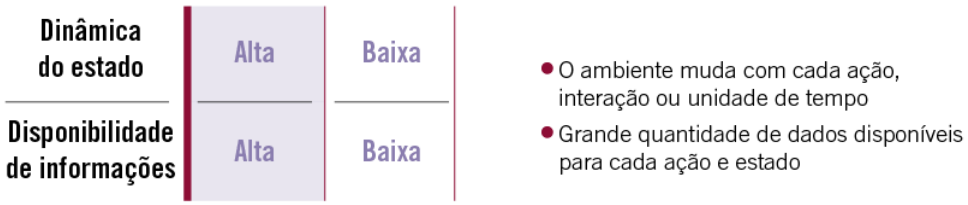

Figura 2: Dinâmica do ambiente.

Para resolução do problema defina os estados e ambiente como apresentado na figura 3. Podemos ver que existem caminhos com buracos, semáforos e faixa de pedestre. Nesses casos o tempo de trajeto pode ser elevado caso tomemos a decisão de seguir por esses caminhos. Dessa forma, consideramos que ir por esse caminho afetará negativamente as nossa decisões.

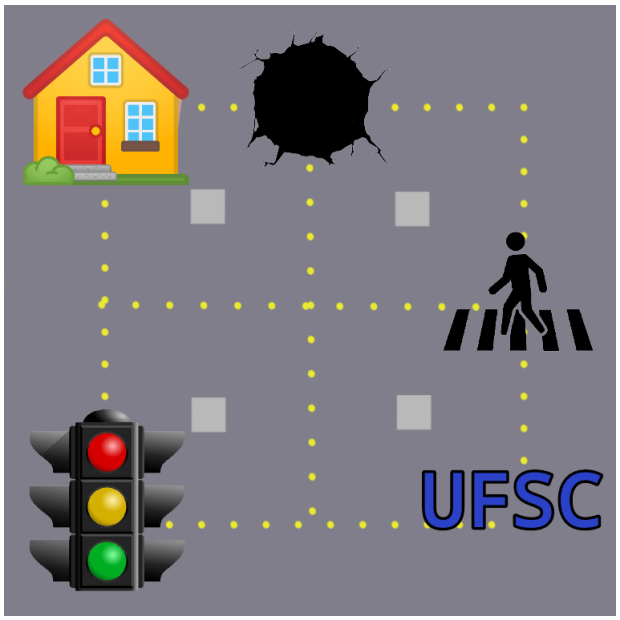

Figura 3: Dinâmica do ambiente.

4.2 Solução:

Para entendermos como funciona a solução apresentada, primeiramente é preciso entender o conceito de agente e ambiente.

O ambiente é o cenário ou situação em que o agente se encontra. O agente usa as informações que ele consegue do ambiente para tomar as decisões necessárias para alcançar um objetivo.

Para o processo de decisão de Markov, utilizado nessa solução, quando é possível tomar várias decisões, o objetivo é obter a maior recompensa possível no final do processo. Portanto, podemos estabelecer uma política que envolva o sacrifício de uma boa decisão a fim de atingir esse objetivo final.

Para isso, ele conta com:

- Conjunto de estados (S): todas as possíveis localizações do agente no ambiente.
- Conjunto de ações (A): todas as ações que podem ser tomadas pelo agente estando em determinado estado.

- Probabilidades de transição: probabilidade de que uma ação seja bem sucedida ao longo prazo.
- Recompensa (R): o valor adquirido ao chegar em algum estado.
- Fator de desconto (γ): diminui o valor de futuras recompensas se comparado com as recompensas atuais.

Sabendo disso, podemos entender que o processo de decisão de Markov (PDM) é independente de ações do passado, ou seja, as ações futuras serão tomadas com base nas variáveis atuais, ou seja, S_t e R_t.

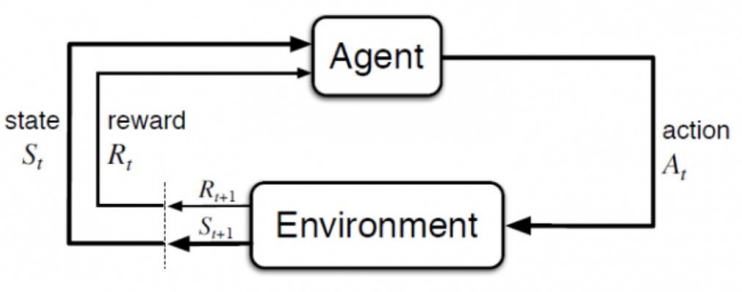

Figura 4: Esquemático de funcionamento do processo de decisão de Markov

A figura 4 ilustra que o ambiente dá ao agente o estado e a recompensa atual, e o agente toma alguma decisão que altera o ambiente, de modo que isso se torna um ciclo.

A forma utilizada para resolver o PDM nesta solução foi a Iteração de Valor (*Value Iteration*). Como o objetivo final é conseguir o maior valor de recompensa acumulado, o processo utiliza a equação de otimização de Bellman:

$$V(s_t) = max_{a_t} \left\{ R(s_t) + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t) V(st+1) \right\} \forall s \tag{3}$$

A *Value Iteration* é uma técnica de aprendizado por reforço. Ela computa o valor resultante para cada iteração e adiciona o valor mais alto à política atual.

O algoritmo utilizado para a implementação dessa técnica converge com base em um erro mínimo aceitável pré definido e exige bastante tempo por iteração, já que sempre examina todo o conjunto de espaços.

Algoritmo:

- Loop infinito até atingir convergência:
  - Define uma variável erro := 0.
  - Para cada estado s no conjunto de estados:
    - v anterior := V(s) (Armazena o valor anterior do estado)
    - v atual := 0
    - Para cada possível ação em um estado:
      - v := valor de recompensa esperado passando por esse estado e ação
      - se o valor de v calculado para a ação e estado for maior que o v atual:
      - v atual := v
      - politica do estado := ação que teve maior valor v
      - V(s) := maior valor de v (v atual) dentre todas as diferentes ações do estado s

- erro := max(erro, —v anterior – V(s)—)
- Se o erro for menor que epsilon o loop infinito se encerra, pois ocorreu a convergência.

A solução desse problema foi desenvolvida utilizando *Python* e aplicando o algoritmo iterativo descrito até a convergência.

Primeiramente vamos definir os parâmetros utilizados para solução do problema e importar a biblioteca:

- $\gamma = 90\%$, fator de desconto
- Noise = 10% , que representa uma probabilidade de fazes uma ação aleatória ao contrário da ação esperada, de forma que o algoritmo consiga explorar ao máximo o conjunto de estados-ações
- épsilon = 0.05 , define o ponto onde podemos considerar a convergência.

```
import numpy as np

#Definicao de parametros
epsilon = 0.05
GAMMA = 0.9
NOISE = 0.1
```

Criamos uma grade 3x3 que representa o conjunto de estados que o aluno pode decidir tomar

```
#Definicao dos estados e tamanho da grade
estados = []
for i in range(3):
    for j in range(3):
            estados.append((i,j))
```

Agora definimos as recompensas, tanto negativas quanto positivas, para o conjunto de estados dado:

- Semáforo : recompensa negativa -1;
- Faixa de pedestre : recompensa negativa -2;
- Buraco : recompensa negativa -1;
- Destino final : recompensa positiva +2;
- Todos outros estados : recompensa 0.

```
#Definicao das recompensas para todos estados
recompensas = {}
for i in estados:
    if i == (1,2):
        recompensas[i] = -2
    elif i == (2,0):
        recompensas[i] = -1
    elif i == (0,1):
        recompensas[i] = -1
    elif i == (2,2):
        recompensas[i] = 2
    else:
        recompensas[i] = 0
```

Criamos um dicionário onde listamos para cada estado que o agente pode se encontrar o conjunto de ações que ele pode decidir tomar.

```
#Definindo o conjunto de acoes.
#Para o estado 2,2 nao teremos nenhuma acao definido, pois e o estado final
acoes = {
    (0,0):('B', 'D'),
    (0,1):('B', 'D', 'E'),
    (0,2):('B', 'E'),

    (1,0):('B', 'C', 'D'),
    (1,1):('B', 'D', 'E', 'C'),
    (1,2):('B', 'E', 'C'),

    (2,0):('C', 'D'),
    (2,1):('C', 'E', 'D')
    }
```

A política inicial do algoritmo é definida de forma aleatória dentre as possíveis ações que o agente pode tomar em cada estado.

```
#Definindo a politica inicial de forma aleatoria
politica={}
for s in acoes.keys():
    politica[s] = np.random.choice(acoes[s])
print(politica)
```

Atribuímos ao valor inicial de cada estado a recompensa definida para aquele estado. Esses valores irão se alterar ao longo das iterações até chegarmos nos valores relativos a política otimizada.

```
#Atribuindo o valor inicial para a funcao de valor como as recompensas de cada estado
V={}
for s in estados:
    if s in acoes.keys():
        V[s] = 0
    if s == (1,2):
        V[s]=-2
    if s == (0,1):
        V[s]=-1
    if s == (2,0):
        V[s]=-1
    if s == (2,2):
        V[s]=2
```

As recompensas de cada estado podem ser representadas pela ilustração:

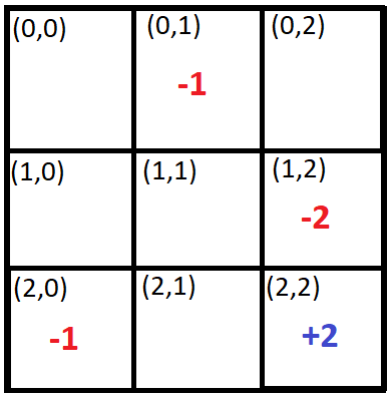

Figura 5: Matriz de recompensas

Com os elementos do processo de decisão de Markov definido no código, podemos passar para a codificação do algoritmo da Iteração de Valor.

```
iteracao = 0
while True:
    erro = 0
    for s in estados:
        # print(politica)
        if s in politica:
            v_anterior = V[s]
            v_atual = 0

            # Itera para cada possivel acao em determinado estado
            for a in acoes[s]:
                if a == 'C':
                    prox = [s[0]-1, s[1]]
                if a == 'B':
                    prox = [s[0]+1, s[1]]
                if a == 'E':
                    prox = [s[0], s[1]-1]
                if a == 'D':
                    prox = [s[0], s[1]+1]

                #Define uma nova acao aleatoria
                random_1=np.random.choice([i for i in acoes[s] if i != a])
                if random_1 == 'C':
                    acao = [s[0]-1, s[1]]
                if random_1 == 'B':
                    acao = [s[0]+1, s[1]]
                if random_1 == 'E':
                    acao = [s[0], s[1]-1]
                if random_1 == 'D':
                    acao = [s[0], s[1]+1]

                #Calcula o valor
                prox = tuple(prox)
                acao = tuple(acao)
                v = recompensas[s] + (GAMMA * ((1-NOISE)* V[prox] + (NOISE * V[acao])))
                # Verifica se o houve uma melhora no valor, se sim redefine a politica
    com
                # a acao tomada e o valor para o estado que est  sendo iterado
                if v > v_atual:
                    v_atual = v
                    politica[s] = a

            # Armazena o melhor valor para todos os estados
            V[s] = v_atual
            # Calcula o erro entre o conjunto de valores anteriores e o valor atual
            erro = max(erro, np.abs(v_anterior - V[s]))

   # Verifica se a iteracao chegou na convergencia, de acordo com o epsilon definido
    if erro < epsilon:
        break
    iteracao += 1
print(politica)
```

É importante notar que, por conta da aleatoriedade inserida no algoritmo, o número de iterações necessárias para chegar em um valor ótimo pode variar drasticamente, de forma que, a convergência pode ser alcançada seja com dezenas de interações ou com dezenas de milhar.

Caso desejássemos inserir maior complexidade ao ambiente, seja aumentando a grade de estados ou inserindo mais recompensas, o tempo e computação necessária para calcular o problema pode acabar ficando inviável.

A política inicial aleatória ficou igual a:

```
(0, 0): 'D', (0, 1): 'E', (0, 2): 'E',
(1, 0): 'B', (1, 1): 'B', (1, 2): 'E',
(2, 0): 'C', (2, 1): 'D'
```

Enquanto a política resultante da convergência do algoritmo ficou igual a:

```
(0, 0): 'B', (0, 1): 'B', (0, 2): 'E',
(1, 0): 'D', (1, 1): 'B', (1, 2): 'E',
(2, 0): 'D', (2, 1): 'D'
```

Podemos ver também os valores ótimos calculados para cada estado, que representam a expectativa de recompensa futura passando por aquele estado. De maneira simplificada, representa o quão bom é passar por aquele estado, de modo que quanto maior o número, melhor.

```
(0, 0): 1.015086114054373, (0, 1): 0.11718200592739692, (0, 2): 0.0949174248011915,
(1, 0): 1.1968282242164916, (1, 1): 1.35638976751537, (1, 2): 0,
(2, 0): 0.45355792716138854, (2, 1): 1.660820213444525, (2, 2): 2
```

Ou seja, segundo a política final e os valores ótimos calculados, o melhor caminho a se seguir quando partimos do estado (0,0) e desejamos chegar ao estado (2,2) é baixo, direita, baixo e direita. Podemos perceber que se trata de um resultado otimizado pois não passa por nenhum estado com recompensa negativa.

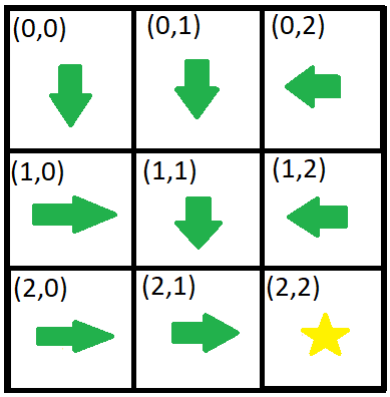

Figura 6: Política ótima

5 Referências

- Tabelas sobre população. Acesso em 10 de maio de 2021. Disponível em: `https://www.ibge.gov.br/`
- Informações de saúde. Acesso em 12 de maio de 2021. Disponível em: `http://tabnet.datasus.gov.br/`
- Tábua completa de mortalidade para o Brasil 2019. IBGE. Acesso em 12 de maio de 2021. Disponível em: `https://biblioteca.ibge.gov.br/visualizacao/periodicos/3097/tcmb_2019.pdf`
- Joice Viviani Birk, Eliete Biasotto Hauser e Vandoir Stormowski. MODELOS MATEMÁTICOS PARA A EXPECTATIVA DE VIDA DOS BRASILEIROS. Canoas RS. 16, 17 e 18 de outubro de 2013. Acesso em 14 de maio de 2021. Disponível em: `https://www.researchgate.net/publication/271706445_MODELOS_MATEMATICOS_PARA_A_EXPECTATIVA_DE_VIDA_DOS_BRASILEIROS`
- Dados Decatlhon. Acesso em 17 de maio de 2021. Disponível em: `https://malouche.github.io/data_in_class/decathlon_data.html`

- PCA na mão e no Python. Acesso em 17 de maio de 2021. Disponível em: `https://leandrocl2005.github.io/pca_na_mao_e_no_python/`
- BARTO, Richard S. Sutton And Andrew G.. Reinforcement Learning: an introduction. Cambridge, Massachusetts: The Mit Press, 1992. Disponível em: http://incompleteideas.net/book/first/ebook/the-book.html. Acesso em: 25 abr. 2021.
- BHANDARKAR, Raghuveer. Policy Iteration in RL: a step by step illustration. A step by step Illustration. Disponível em: https://towardsdatascience.com/policy-iteration-in-rl-an-illustration-6d58bdcb87a7. Acesso em: 10 mai. 2021.
- SILVER, David. UCL Course on RL. 2015. Disponível em: https://www.davidsilver.uk/teaching/. Acesso em: 01 maio 2021.
- POOLE, David; MACKWORTH, Alan. Artificial Intelligence: foundations of computational agents. Vancouver: Cambridge University Press, 2017. Disponível em: https://artint.info/aifca1e.html. Acesso em: 01 maio 2021.