Projeto Computacional

Mestrado Integrado em Engenharia Física Tecnológica
Matemática Computational
Prof$^{\underline{a}}$ Ana Leonor Silvestre

Ana Sofia Camões de Sousa 96508
Duarte Miguel de Aguiar Pinto e Morais Marques 96523
Isabel Maria Jaló Alexandre 96537
Martim da Costa Graça Marques Ferreira 96554

Ano Letivo 2020/2021
1$^{\underline{o}}$ semestre

Conteúdo

Índice de programas

1 Introdução

Este trabalho tem como objetivo aplicar uma série de métodos de Matemática Computacional a exercícios concretos.

Em primeiro lugar, é implementado um programa que permite a visualização de funções, o qual terá um papel auxiliar ao longo de todo o trabalho. Permite ainda obter alguma familiarização com a linguagem utilizada - *Python* - e as bibliotecas necessárias.

De seguida, é implementado o método quasi-Newton para localização de zeros de uma função. Este método é aplicado no estudo de uma Função de Conectividade entre Neurónios.

Consideram-se ainda quadraturas, em primeira instância simples e depois composta, para aproximação do valor de integrais. É aplicada a última na determinação do espaço percorrido por um ponto em movimento.

Finalmente, é considerado um método de Runge Kutta para aproximação de soluções de equações diferenciais ordinárias. É então aplicado a um modelo *SEIRP*, o qual modela a propagação da COVID-19.

2 Métodos e Ferramentas

Na execução deste trabalho computacional, optou-se por utilizar a linguagem de programação *Python*, aliada à biblioteca numérica Numpy [3] e à ferramenta gráfica Matplotlib [1]. Foi ainda utilizada a biblioteca SciPy [4].

Em primeiro lugar, e como ferramenta auxiliar, foi desenvolvido um programa capaz de, perante a introdução de uma função $f(t)$ pelo utilizador, sob a forma de *string*, a calcular em qualquer t do domínio, bem como apresentar um gráfico desta num dado intervalo, também fornecido pelo utilizador.

O código deste programa é apresentado abaixo, a título de exemplo. Por questões de organização e de facilidade de consulta, os restantes programas desenvolvidos estão presentes no anexo A e quaisquer ficheiros de texto por estes produzidos estão no anexo B. São apresentados no texto principal apenas os exemplos de execução e as figuras produzidas.

Em primeiro lugar, são importadas as bibliotecas necessárias. De seguida, e com recurso à biblioteca numérica, são definidas funções "atalho", cujo utilizador pode inserir como parte de f, em conjunto com algumas constantes, de novo com recurso à biblioteca.

De seguida, é realizado o *output* de algumas instruções, sendo dada a opção de impressão de um "manual"de utilização, que lista as funções elementares disponíveis, perante o *input* "man". É depois pedido ao utilizador que insira um intervalo, que pode conter expressões ou funções elementares (por exemplo 1/2 ou $\sin(\pi)$), sendo realizado um teste à validez do mesmo (se o extremo mínimo indicado é realmente menor que o máximo). É depois definido o cálculo da função inserida e é criado um vetor com 1000 elementos, espaçados igualmente, entre *tmin* e *tmax*. Para cada elemento deste vetor é calculado o valor de $f(t)$, de forma a poder desenhar-se um gráfico da função, o qual é, finalmente, apresentado com legendas nos eixos e título correspondente à expressão de $f(t)$.

O programa só é terminado quando se insere "quit"na linha de escrita da função.

Programa 1: Base.py

```python
# Importação das bibliotecas necessárias
# https://numpy.org/doc/stable/reference/routines.math.html
import numpy as np
import matplotlib.pyplot as plt

# Definição das funções necessárias
def sin(x):
    return np.sin(x)
def cos(x):
    return np.cos(x)
def tan(x):
    return np.tan(x)
def arcsin(x):
    return np.arcsin(x)
def arccos(x):
    return np.arccos(x)
def arctan(x):
    return np.arctan(x)
```

```
def sinh(x):
    return np.sinh(x)
def cosh(x):
    return np.cosh(x)
def tanh(x):
    return np.tanh(x)
def arcsinh(x):
    return np.arcsinh(x)
def arccosh(x):
    return np.arccosh(x)
def arctanh(x):
    return np.arctanh(x)
def exp(x):
    return np.exp(x)
def log(x):
    return np.log(x)
def log10(x):
    return np.log10(x)
def sqrt(x):
    return np.sqrt(x)
def cbrt(x):
    return np.cbrt(x)
def fabs(x):
    return np.fabs(x)
def power(x, y):
    return np.power(x, y)

# Definição das constantes necessárias
e = np.e
pi = np.pi

# Funções disponíveis
func = ["sin", "cos", "tan"]
func2 = ["exp", "log", "log10", "sqrt", "cbrt", "fabs", "power"]
consts = ["pi", "e "]

print("\n\nMatemática Computacional")
while (True):
    print("\n\nA função deve ser inserida em função de t")
    print("<man> para opções ou <quit> para sair")
    val = input("Insira uma função: f(t) = ")

    # Impressão de instruções de utilização, perante o input "man"
    if (val == "man"):
        print("\nLista de funções suportadas:\n")
```

```
        for x in func:
            print(x)
        for x in func:
            print("arc" + x)
        for x in func:
            print(x + "h")
        for x in func:
            print("arc" + x + "h")
        for x in func2:
            print(x)
        print("\nLista de constantes suportadas:\n")
        for x in consts:
            print(x)
        continue
    elif (val == "quit"):
        break

    # Input do intervalo
    tmin = float(eval(input("Insira o t mínimo: ")))
    tmax = float(eval(input("Insira o t máximo: ")))

    # Teste
    if (tmin > tmax):
        print("Erro")
        exit()

    # Definição do cálculo da função
    def fu(t):
        return eval(val)

    # ---- Gráfico ----
    # Cálculo da função entre tmin e tmax
    dt = (tmax-tmin) / 1000
    t = np.arange(tmin, tmax+dt, dt)
    y = []
    for p in t:
        y.append(fu(p))
    # Plot da função
    plt.plot(t, y, 'b--')
    # Definição de legendas
    plt.xlabel('$t$')
    plt.ylabel('$f(t)$')
    plt.title("f(t) = " + val)
    # Desenho do gráfico
    plt.show()
```

De seguida são apresentados dois exemplos da execução do programa. Ao longo de todo o relatório, será utilizada a notação input para denotar interações na forma de texto introduzido pelo utilizador, excetuando caracteres "mudança de linha", ou seja, *enter*.

```
Matemática Computacional



A função deve ser inserida em função de t
<man> para opções ou <quit> para sair
Insira uma função: f(t) = man

Lista de funções suportadas:

sin
cos
tan
arcsin
arccos
arctan
sinh
cosh
tanh
arcsinh
arccosh
arctanh
exp
log
log10
sqrt
cbrt
fabs
power

Lista de constantes suportadas:

pi
e



A função deve ser inserida em função de t
<man> para opções ou <quit> para sair
Insira uma função: f(t) = quit
```

```
Matemática Computacional


A função deve ser inserida em função de t
<man> para opções ou <quit> para sair
Insira uma função: f(t) = arctan(t)
Insira o t mínimo: -pi/2
Insira o t máximo: pi/2


A função deve ser inserida em função de t
<man> para opções ou <quit> para sair
Insira uma função: f(t) = quit
```

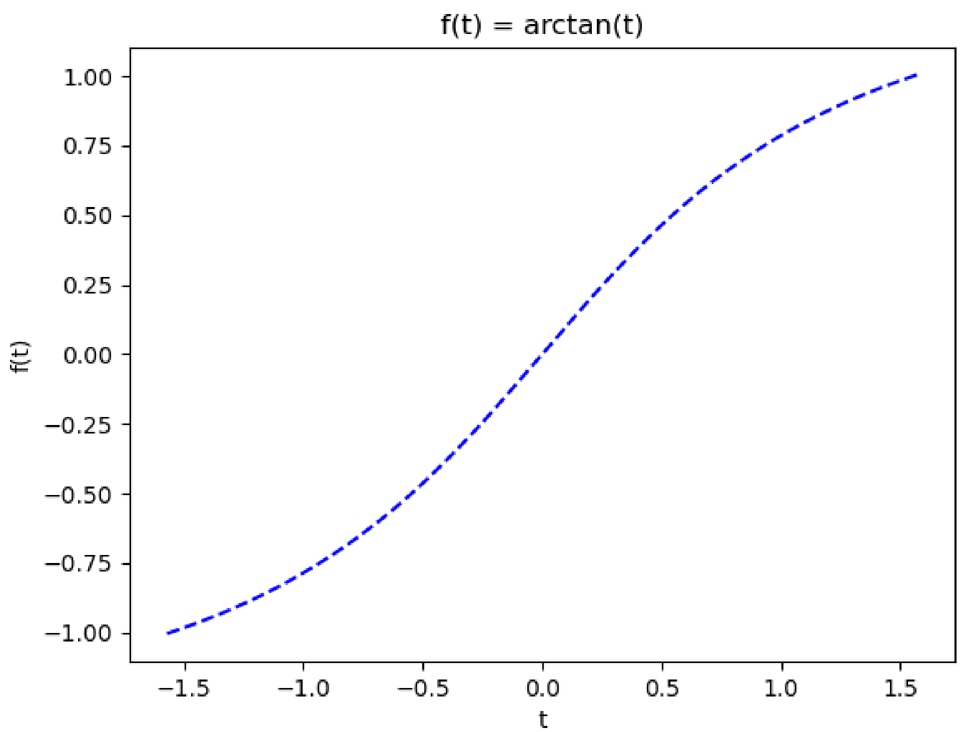

Figura 1: Gráfico da função *arctan*

3 Grupo I

3.1 Pergunta 1. - Método Quasi-Newton

3.1.1 Alínea a

Dada uma função f, o método quasi-Newton para a equação $f(x) = 0$ consiste em:

$$x_{k+1} = x_k - \frac{\delta}{f(x_k + \delta) - f(x_k)} f(x_k),\ k = 0, 1, 2, ... \tag{1}$$

Considera-se δ um valor dado, próximo de zero. Trata-se de uma "aproximação"do método de Newton que evita a necessidade do cálculo de derivadas, o qual seria computacionalmente mais "caro".

Antes de determinar a ordem e o coeficiente assintótico de convergência, é útil verificar que o método de facto converge (para alguma iterada inicial que não o próprio ponto fixo).

Para o método quasi-Newton apresentado, seja z a solução da equação $f(x) = 0$ e $g(x)$ a função iteradora:

$$f(z) = 0 \Leftrightarrow g(z) = z \tag{2}$$

$$g(x) = x - \frac{\delta f(x)}{f(x + \delta) - f(x)} \equiv g_\delta(x). \tag{3}$$

Fazendo a derivada da expressão (3), obtém-se:

$$g'_\delta(x) = 1 - \frac{\delta f'(x)}{f(x+\delta) - f(x)} + \frac{\delta f(x)(f'(x+\delta) - f'(x))}{(f(x+\delta) - f(x))^2} = 1 + \delta \frac{f(x)f'(x+\delta) - f'(x)f(x+\delta)}{(f(x+\delta) - f(x))^2} \tag{4}$$

$$g'_\delta(z) = 1 - \frac{\delta f'(z)}{f(z+\delta)} \Leftrightarrow g'_\delta(z) = 1 - \frac{f'(z)}{\frac{f(z+\delta) - f(z)}{\delta}} \tag{5}$$

Em que a última equivalência é verdadeira porque $f(z) = 0$. Por aplicação do Teorema de Lagrange, temos

$$g'_\delta(z) = 1 - \frac{f'(z)}{f'(\xi)} \tag{6}$$

em que ξ é um número entre z e $z + \delta$. Como δ é muito próximo de zero, temos:

$$\frac{f'(z)}{f'(\xi)} \approx 1 \Rightarrow \left|g'_\delta(z)\right| = \left|1 - \frac{f'(z)}{f'(\xi)}\right| \approx 0 \tag{7}$$

Logo, quanto menor for $|\delta|$, mais atrator será o ponto fixo z, concluindo-se que o método converge localmente para o ponto fixo, pelo que se pode determinar a ordem e o coeficiente assintótico de convergência.

Seja g uma função $C^p(I)$, onde I é uma vizinhança de z, ponto fixo de g, ou seja, tal que $g(z) = z$. Se $g'(z) = ... = g^{(p-1)} = 0$, com $g^{(p)} \neq 0$, então a ordem de convergência de g é p e:

$$\lim_{n\to\infty} \frac{|z - x_{n+1}|}{|z - x_n|^p} = \frac{|g^p(z)|}{p!} = K_\infty$$

Onde K_∞ é o coeficiente assintótico de convergência. Em particular, nas condições do teorema do ponto fixo, a convergência é linear se $g'(z) \neq 0$, sendo que:

$$\lim_{n\to\infty} \frac{|z - x_{n+1}|}{|z - x_n|} = |g'(z)| = K_\infty, \quad K_\infty \in [0, 1]$$

Tendo em conta (2) e que se deverá ter $f(x_k + \delta) \neq f(x_k)$ aquando da aplicação do método quasi-Newton, sabemos que a condição $f(z + \delta) \neq 0$ será sempre verdadeira. Deste modo, ter-se-á $|g'_\delta(z)| \neq 0$ sempre que se verificar a condição

$$f(z + \delta) \neq \delta f'(z) \tag{8}$$

Supondo f uma função C^1, a equação (5) pode ser reescrita, tendo em conta que $f(z) = 0$ e que

$$f(z + \delta) = f(z) + \delta f'(z) + \frac{\delta^2 f''(\beta)}{2}$$

Ficando:

$$g'_\delta(z) = 1 - \frac{f'(z)}{f'(z) + \frac{\delta f''(\beta)}{2}}, \tag{9}$$

Em que β está entre z e $z + \delta$ e em que $g'_\delta(z) \neq 0$ quando:

$$f''(\beta) \neq 0 \tag{10}$$

Assim, verificando-se a condição (8) ou (10), o método quasi-Newton terá ordem de convergência $p = 1$ (convergência linear) e coeficiente assimptótico de convergência $K_\infty \equiv K_\delta = |g'_\delta(z)| = \left|1 - \frac{\delta f'(z)}{f(z+\delta)}\right| = \left|1 - \frac{f'(z)}{f'(z)+\frac{\delta f''(\beta)}{2}}\right|$.

Analisando a equação (9), é também possível estabelecer uma relação de similitude entre o método quasi-Newton e o método de Newton. Facilmente se depreende que $g'_\delta(z) \to 0$ quando $\delta \to 0$. Ora, derivando a expressão (4) e calculando-a em $x = z$, obtém-se:

$$g''(z) = \frac{-f''(z)}{\frac{f(z+\delta)-f(z)}{\delta}} + \frac{f'(z)\frac{f'(z+\delta)-f'(z)}{\delta}}{\frac{(f(z+\delta)-f(z))^2}{\delta^2}} + \frac{f'(z)\frac{f'(z+\delta)-f'(z)}{\delta}}{\frac{(f(z+\delta)-f(z))^2}{\delta^2}} \quad \overset{\delta \to 0}{\equiv}$$

$$\equiv \frac{-f''(z)}{f'(z)} + \frac{f'(z)f''(z)}{f'(z)^2} + \frac{f'(z)f''(z)}{f'(z)^2} = \frac{f''(z)}{f'(z)}$$

Deste modo, obtém-se que $|g''(z)| = \left|\frac{f''(z)}{f'(z)}\right|$. Quando $f'(z), f''(z) \neq 0$, a ordem de convergência será, então, $p = 2$, sendo o coeficiente assimptótico de convergência dado por $K_\infty = \left|\frac{f''(z)}{2f'(z)}\right|$. Conclui-se, portanto, que o método quasi-Newton convergirá de forma semelhante ao método de Newton quando $\delta \to 0$, tal como esperado.

3.1.2 Alínea b

Tendo este método convergência linear, é válido que

$$\lim_{n\to\infty} \frac{|z - x_{n+1}|}{|z - x_n|} := K_\delta{}^{1}, \quad K_\delta \in (0,1) \tag{11}$$

$$\begin{aligned} |z - x_{n+1}| &\leq C|z - x_n| \\ &= C|z - x_{n+1} + x_{n+1} - x_n| \\ &\leq C|z - x_{n+1}| + C|x_{n+1} - x_n| \Rightarrow \\ \Rightarrow (1-C)|z - x_{n+1}| &\leq C|x_{n+1} - x_n| \iff \\ \iff |z - x_{n+1}| &\leq \frac{C}{1-C}|x_{n+1} - x_n| \end{aligned} \tag{12}$$

Para n grande,

$$|z - x_{n+1}| \approx K_\delta |z - x_n| \Rightarrow |z - x_{n+1}| \leq \frac{K'_\delta}{1 - K'_\delta}|x_{n+1} - x_n| \tag{13}$$

onde K'_δ é um majorante de K_δ.

Justifica-se que, para δ suficientemente pequeno, e n suficientemente grande, garantindo-se convergência:

$$K_\delta \leq \frac{1}{2} \Rightarrow |z - x_{n+1}| \leq |x_{n+1} - x_n| \tag{14}$$

Considerou-se então como critério de paragem:

$$|z - x_{n+1}| \leq |x_{n+1} - x_n| \leq \epsilon \tag{15}$$

Considerou-se como erro de cada iterada a diferença, em módulo, entre o seu valor e o valor da iterada final, tomando-se este como o valor real do zero da função.

O programa desenvolvido 2 deriva do anterior, na medida em que permite a introdução de várias funções, listadas através de um "manual". Não permite, todavia, que a função seja trivialmente constante, isto é, a sua expressão tem que depender de t. É pedido ao utilizador um δ, uma iterada inicial x_0, um número máximo de iteradas a realizar n_{max} e ainda uma tolerância de erro ϵ. São definidos os cálculos da função $f(t)$ e $g(t)$ (função iteradora). É definida uma função auxiliar de cálculo de erro, a qual permite arredondá-lo sempre para cima, majorando-o, com apenas um algarismo. São então calculadas as iteradas até o número da iterada ser superior a n_{max}, ser encontrado z (isto é, $f(x_i) = 0$), ou o erro ser inferior a ϵ. São apresentadas na consola como *output*, bem como um gráfico da função com as iteradas calculadas e numeradas, de modo a ser possível avaliar quão rápido converge, se a convergência é monótona, etc.

[1] Utiliza-se o símbolo K_δ para designar o coeficiente assintótico de convergência K_∞ a fim de enfatizar a sua dependência do δ escolhido.

3.2 Pergunta 2. - Função de Conectividade entre Neurónios

Para responder a este exercício, foi necessário proceder a um estudo teórico da seguinte função facultada, referente à conectividade entre neurónios:

$$W(r) = B\exp(-kr)[k\sin(ar) + \cos(ar)] \tag{16}$$

B, k, a são constantes reais positivas e $r > 0$ é a distância entre neurónios.

Para a resolução deste exercício, considere-se a função $W(r)_c = W(r) + c$, em que $c \in \mathbb{R}$. A função original é $W(r)_0$.

Consistindo esta função numa soma de uma constante com um produto entre uma exponencial e a soma de funções trigonométricas, $W(r)_c \in C^\infty(\mathbb{R}^+)$. Ao se ter a soma de um cosseno com um seno, esta é uma função oscilatória. Como está a multiplicar por uma exponencial decrescente $(-kr < 0)$, a amplitude de oscilação diminui à medida que r aumenta. Tem-se que $\lim\limits_{r\to+\infty} W(r) = 0$, como seria de esperar, pois, se dois neurónios estiverem muito longe, a sua conectividade é nula.

$$W'(r)_c = -kB\exp(-kr)[k\sin(ar) + \cos(ar)] + B\exp(-kr)[ak\cos(ar) - a\sin(ar)] \tag{17}$$

Ao longo do exercício, considera-se $B = 2$ e $a = 3$, logo $W(r) = 2\exp(-kr)[k\sin(3r) + \cos(3r)]$ e $W'(r) = 2\exp(-kr)[(-k^2 - 3)\sin(3r) + 2k\cos(3r)]$.

3.2.1 Alínea a

$$W(r) = 2\exp(-r)[\sin(3r) + \cos(3r)] \tag{18}$$

Facilmente se observa que de 0 a pelo menos $\frac{\pi}{6}$, a função (18) é positiva, pois a exponencial é sempre positiva e tanto o $\sin(3r)$ como o $\cos(3r)$ são positivos nesse intervalo. Temos que $W(\frac{\pi}{6}) = 2\exp(-\frac{\pi}{6}) > 0$. Como a função é contínua, para tomar valores negativos tem que existir um zero (consequência do Teorema de Bolzano).

A derivada é:

$$W'(r) = 2\exp(-r)[-4\sin(3r) + 2\cos(3r)] \tag{19}$$

A derivada será nula quando $\cos(3r) = 2\sin(3r)$. Utilizando a fórmula fundamental da trigonometria, tal acontecerá quando $\sin^2(3r) = \frac{1}{5}$ e o $\sin(3r)$ e o $\cos(3r)$ tiverem o mesmo sinal. Entre $\frac{\pi}{6}$ e $\frac{\pi}{3}$, $\sin(3r)$ e $\cos(3r)$ têm sinais opostos, logo a derivada não se anula nesse intervalo. Visto que $W'(\frac{\pi}{6}) = -8\exp(-\frac{\pi}{6}) < 0$ e $W'(\frac{\pi}{3}) = -4\exp(-\frac{\pi}{3}) < 0$, a derivada é negativa no intervalo referido, logo a função é estritamente decrescente. $W(\frac{\pi}{6}) > 0.1$ e $W(\frac{\pi}{3}) < 0$, logo neste intervalo existe um único ponto em que $W(r) = 0.1$. No entanto, é necessário estudar o que acontece de 0 a $\frac{\pi}{6}$. Nesse intervalo inicial, a derivada tem um zero. De forma a estudar o sinal da derivada, pode-se ver que $W'(0) = 4 > 0$, logo a derivada é positiva até ao seu zero, tornando-se então negativa, ou seja, a função $W(r)$ cresce até atingir um máximo e depois decresce. $W(0) = 2 > 0.1$, podendo-se assim concluir que o primeiro ponto em que $W(r) = 0.1$ se encontra no intervalo de $\frac{\pi}{6}$ até $\frac{\pi}{3}$, sendo que a função é maior que 0.1 até ao início do intervalo referido.

Utilizando então $\frac{\pi}{6}$ como iterada inicial, pode-se utilizar o programa descrito na alínea anterior para descobrir o primeiro zero de $W(r)_{-0.1}$.

Após testar diferentes valores de δ, concluiu-se que o valor que garantia maior eficiência do método era 10^{-8}. Obtiveram-se, então, os seguintes resultados:

```
Matemática Computacional


Método Quasi-Newton
A função deve ser inserida em função de t
<man> para opções
Insira uma função: f(t) = 2*exp(-t)*(sin(3*t)+cos(3*t))-0.1
Insira o delta: 0.00000001
Insira a aproximação inicial x0: pi/6
Insira o número máximo de iterações a realizar: 100
Insira a tolerância de erro: 0.00000001

---- Cálculo das iteradas ----

   0: 5.235987755982988e-01 +- 3e-01
   1: 7.524976272652966e-01 +- 8e-03
   2: 7.601200188014523e-01 +- 5e-05
   3: 7.601699723316495e-01 +- 3e-09
   4: 7.601699745191770e-01 (f(z) = 4e-17)
```

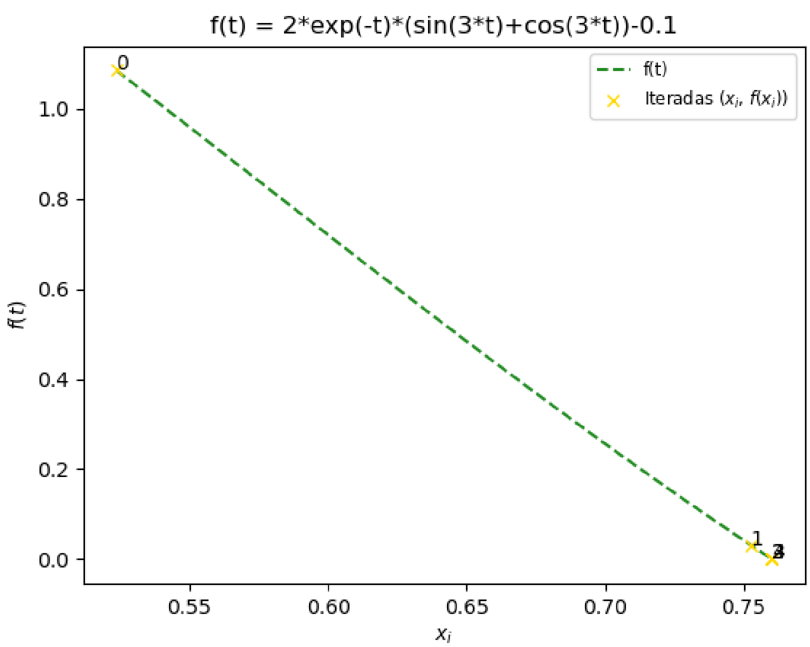

Figura 2: Gráfico produzido com as iteradas

Como se pode observar, o método originou uma iterada final com ordenada muito próxima de zero. Pode-se assim concluir que o primeiro intervalo em que $W(r) \geq 0.1$ é aproximadamente $]0; 0.7601699745191770]$.

Para descobrir o segundo intervalo, é necessário continuar o estudo teórico da função a partir de $\frac{\pi}{3}$.

De $\frac{\pi}{3}$ a $\frac{\pi}{2}$, $\sin(3r)$ e o $\cos(3r)$ têm o mesmo sinal, logo a derivada terá um zero neste intervalo. A função em $\frac{\pi}{3}$ é negativa, decresce até ao zero da derivada e depois cresce até $\frac{\pi}{2}$. $W(\frac{\pi}{2}) = -2\exp(-\frac{\pi}{2}) < 0$, logo de $\frac{\pi}{3}$ a $\frac{\pi}{2}$ a função é negativa. De $\frac{\pi}{2}$ a $\frac{2\pi}{3}$, $\sin(3r)$ e $\cos(3r)$ têm sinais contrários, a derivada é estritamente positiva e $W(r)$ é crescente. $W(\frac{2\pi}{3}) = 2\exp(-\frac{2\pi}{3}) > 0.1$, logo vem do Teorema de Bolzano que existe um ponto entre $\frac{\pi}{2}$ e $\frac{2\pi}{3}$ tal que $W(r) = 0.1$.

Utilizando desta vez $\delta = 10^{-6}$ e $\frac{\pi}{2}$ como iterada inicial, obtém-se:

```
Matemática Computacional



Método Quasi-Newton
A função deve ser inserida em função de t
<man> para opções
Insira uma função: f(t) = 2*exp(-t)*(sin(3*t)+cos(3*t))-0.1
Insira o delta: 0.000001
Insira a aproximação inicial x0: pi/2
Insira o número máximo de iterações a realizar: 100
Insira a tolerância de erro: 0.00000001

---- Cálculo das iteradas ----

   0: 1.570796326794897e+00 +- 4e-01
   1: 1.880927216571553e+00 +- 4e-02
   2: 1.911886287044048e+00 +- 2e-03
   3: 1.913219823800147e+00 +- 3e-06
   4: 1.913222393564791e+00 +- 6e-12
   5: 1.913222393570632e+00 (f(z) = -8e-17)
```

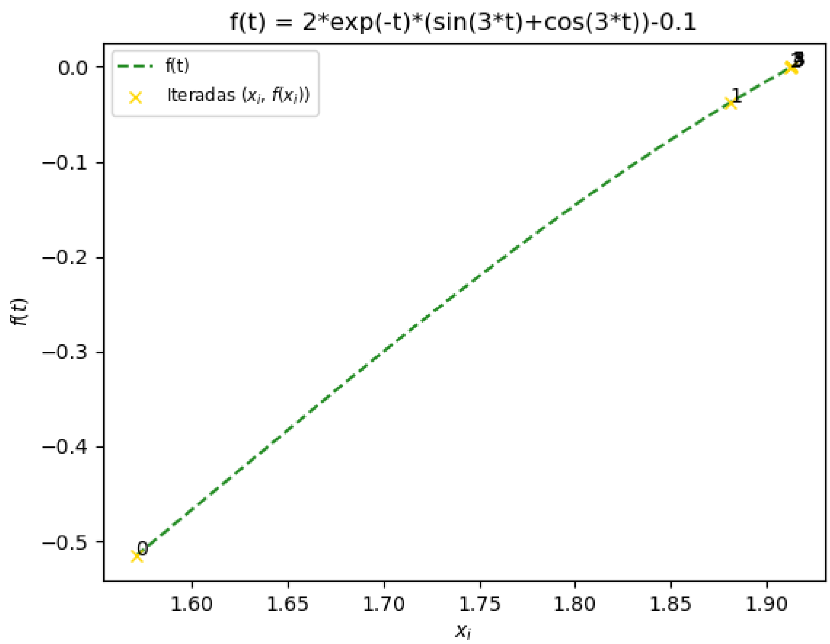

Figura 3: Gráfico produzido com as iteradas

De $\frac{2\pi}{3}$ a $\frac{5\pi}{6}$, a derivada vai ter um novo zero. $W(r)$ será crescente até esse zero, tornando-se depois decrescente. $W(\frac{5\pi}{6}) = 2\exp(-\frac{5\pi}{6}) > 0.1$, logo, de $\frac{2\pi}{3}$ a $\frac{5\pi}{6}$, $W(r) > 0.1$. No intervalo de $\frac{5\pi}{6}$ a π, a função é decrescente. $W(\pi) = -2\exp(-\pi) < 0$, logo $W(r)_{-0.1}$ tem um zero de $\frac{5\pi}{6}$ a π.

Usando agora $\frac{5\pi}{6}$ como iterada inicial e $\delta = 10^{-7}$, obteve-se:

```
Matemática Computacional


Método Quasi-Newton
A função deve ser inserida em função de t
<man> para opções
Insira uma função: f(t) = 2*exp(-t)*(sin(3*t)+cos(3*t))-0.1
Insira o delta: 0.0000001
Insira a aproximação inicial x0: 5*pi/6
Insira o número máximo de iterações a realizar: 100
Insira a tolerância de erro: 0.00000001

---- Cálculo das iteradas ----

   0: 2.617993877991494e+00 +- 8e-02
   1: 2.696641430251942e+00 +- 7e-04
   2: 2.695962645890826e+00 +- 7e-08
   3: 2.695962715622425e+00 +- 5e-16
   4: 2.695962715622424e+00 (f(z) = 3e-17)
```

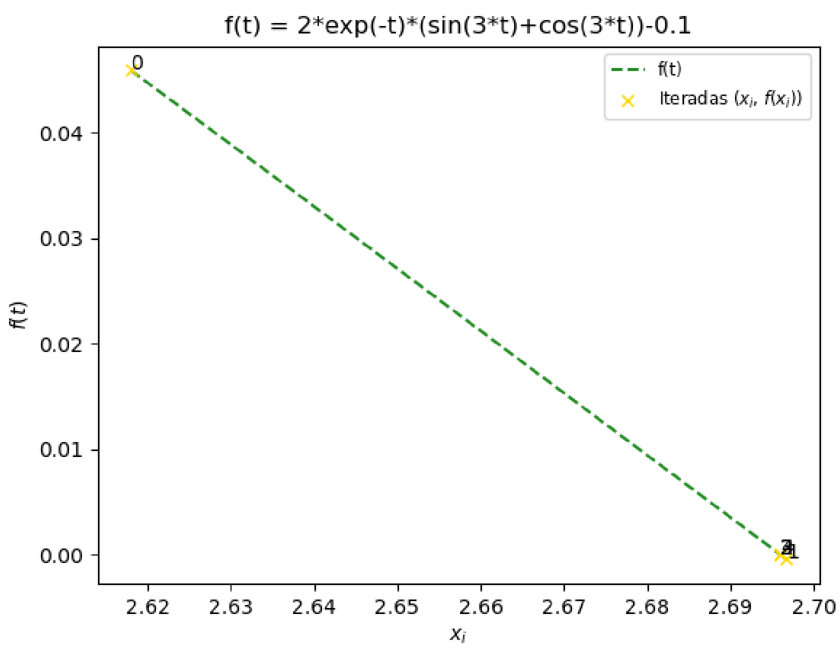

Figura 4: Gráfico produzido com as iteradas

Conclui-se que o segundo intervalo em que $W(r) \geq 0.1$ é $[1.913222393570632; 2.695962715622424]$.

3.2.2 Alínea b

Utilizando os valores obtidos na alínea anterior para o primeiro zero calculado com iterada inicial $r_0 = \frac{\pi}{6}$, que é aproximadamente 0.5235987755982988, e $\delta = 10^{-8}$, obtém-se a seguinte tabela para estudar a convergência do método:

n	r_n	$r_n - r_{n-1}$	$(r_n - r_{n-1})/(r_{n-1} - r_{n-2})$
1	0.7524976272652966	0.2288988516669977	-
2	0.7601200188014523	0.0076223915361557	0.0333002611443626
3	0.7601699723316495	0.0000499535301972	0.0065535245677506
4	0.7601699745191770	0.0000000021875275	0.0000437912492640

Tabela 1: Iteradas do primeiro zero calculado na alínea anterior - potencial convergência linear

Os valores da última coluna da tabela 1 estão a tender para zero, o que permite concluir que o método não apresenta convergência linear para a iterada inicial e para o δ utilizados.

n	r_n	$r_n - r_{n-1}$	$(r_n - r_{n-1})/(r_{n-1} - r_{n-2})^2$
1	0.7524976272652966	0.2288988516669977	-
2	0.7601200188014523	0.0076223915361557	0.1454802455401036
3	0.7601699723316495	0.0000499535301972	0.8597727546092185
4	0.7601699745191770	0.0000000021875275	0.8766397307873951

Tabela 2: Iteradas do primeiro zero calculado na alínea anterior - convergência quadrática

Observando a tabela 2, verifica-se que os valores da última coluna aparentam convergir para um valor próximo de 0.9, concluindo-se que o método apresenta convergência quadrática, como seria de esperar quando se utiliza δ muito próximo de zero (método de Newton).

Utilizando $\delta = 0.1$, obtêm-se as iteradas presentes na tabela 3 e observa-se que o método apresenta convergência linear como tinha sido concluído no estudo teórico no primeiro exercício, uma vez que os valores da última coluna estão a convergir, em módulo, para um valor entre 0 e 1, neste caso, aproximadamente 0.1077.

n	r_n	$r_n - r_{n-1}$	$(r_n - r_{n-1})/(r_{n-1} - r_{n-2})$
1	0.7508085853142543	0.2272098097159555	-
2	0.7610494903265237	0.0102409050122694	0.0450724597897950
3	0.7600740785097427	-0.0009754118167810	-0.0952466423243201
4	0.7601802880982177	0.0001062095884750	-0.1088869200144312
5	0.7601688636371638	-0.0000114244610538	-0.1075652511028926
6	0.7601700941537244	0.0000012305165605	-0.1077089374050849
7	0.7601699616351150	-0.0000001325186094	-0.1076934790390463
8	0.7601699759067256	0.00000001427161067	-0.1076951435996228
9	0.7601699743697450	-0.0000000015369807	-0.1076949679534546

Tabela 3: Iteradas do primeiro zero calculado na alínea anterior - convergência linear ($\delta = 0.1$)

3.2.3 Alínea c

$$W(r) = 2\exp(-kr)[k\sin(3r) + \cos(3r)] \tag{20}$$

$$W'(r) = 2\exp(-kr)[(-k^2 - 3)\sin(3r) + 2k\cos(3r)] \tag{21}$$

Como já foi analisado, devido à exponencial decrescente, a amplitude de oscilação da função $W(r)$ decorrente do seno e do cosseno diminui com o aumento de r. Consequentemente, o primeiro máximo local de $W(r)$ será também máximo absoluto. Sendo r^* a distância para a qual esse máximo é atingido, $W'(r^*) = 0$. Como a exponencial nunca é nula, isso quer dizer que $(-k^2 - 3)\sin(3r^*) + 2k\cos(3r^*) = 0$. Por outro lado, $W(r^*) = 2.4$. Como k é positivo, para $(k^2 + 3)\sin(3r^*) = 2k\cos(3r^*)$, o cos e o sin têm de ter o mesmo sinal. Assim sendo, o primeiro zero da derivada vai ocorrer de 0 a $\frac{\pi}{6}$. Independentemente de k, tanto a derivada como a função são positivas desde 0 até ao primeiro zero da derivada, logo esse zero da derivada será um máximo da função, $r^* \in]0, \frac{\pi}{6}]$. Rearranjando: $(k^2 + 3)\sin(3r^*) = 2k\cos(3r^*) \iff \tan(3r^*) = \frac{2k}{k^2+3} \Rightarrow r^* = \frac{1}{3}\arctan\left(\frac{2k}{k^2+3}\right)$. Como $W(r^*) = 2.4$, $W(r^*)_{-2.4} = 0$, o que permite descobrir k.

Utilizando novamente o programa desenvolvido no exercício 1, calculou-se o k utilizando como iterada inicial o valor utilizado nas duas últimas alíneas (1) e $\delta = 10^{-6}$, e obtiveram-se os seguintes resultados:

```
Matemática Computacional


Método Quasi-Newton
A função deve ser inserida em função de t
<man> para opções
Insira uma função: f(t) =
↪  2*exp(-t*arctan(2*t/(t*t+3))/3)*(t*sin(arctan(2*t/(t*t+3)))+
cos(arctan(2*t/(t*t+3))))-2.4
Insira o delta: 0.000001
Insira a aproximação inicial x0: 1
Insira o número máximo de iterações a realizar: 100
Insira a tolerância de erro: 0.00000001

---- Cálculo das iteradas ----

   0: 1.000000000000000e+00 +- 3e-01
   1: 1.245627701394737e+00 +- 1e-02
   2: 1.254623003805937e+00 +- 2e-05
   3: 1.254640283883012e+00 +- 7e-11
   4: 1.254640283943503e+00 (f(z) = 0e+00)
```

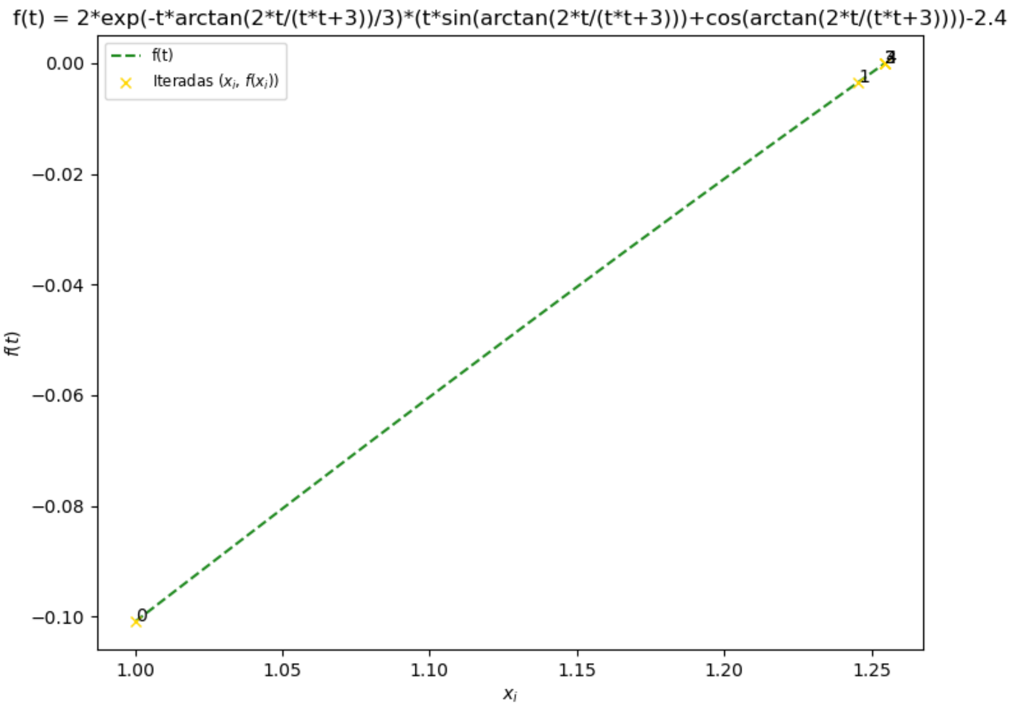

Figura 5: Gráfico produzido com as iteradas

Utilizando o valor de k obtido ($k = 1.254640283943503$), determina-se por fim o valor r^* calculando o zero em $]0, \frac{\pi}{6}]$ de $(-k^2 - 3)\sin(3r^*) + 2k\cos(3r^*)$. Considerando como iterada inicial $\frac{\pi}{6}$ e $\delta = 10^{-4}$:

```
Matemática Computacional


Método Quasi-Newton
A função deve ser inserida em função de t
<man> para opções
Insira uma função: f(t) =
↪  (-1.254640283943503*1.254640283943503-3)*sin(3*t)+2*1.254640283943503*cos(3*t)
Insira o delta: 0.0001
Insira a aproximação inicial x0: pi/6
Insira o número máximo de iterações a realizar: 100
Insira a tolerância de erro: 0.00000001

---- Cálculo das iteradas ----

   0: 5.235987755982988e-01 +- 4e-01
   1: -8.419473888462137e-02 +- 3e-01
```

```
2: 2.290051937639819e-01 +- 7e-02
3: 1.665331651836093e-01 +- 8e-04
4: 1.672512640168913e-01 +- 9e-10
5: 1.672512631272741e-01 (f(z) = 0e+00)
```

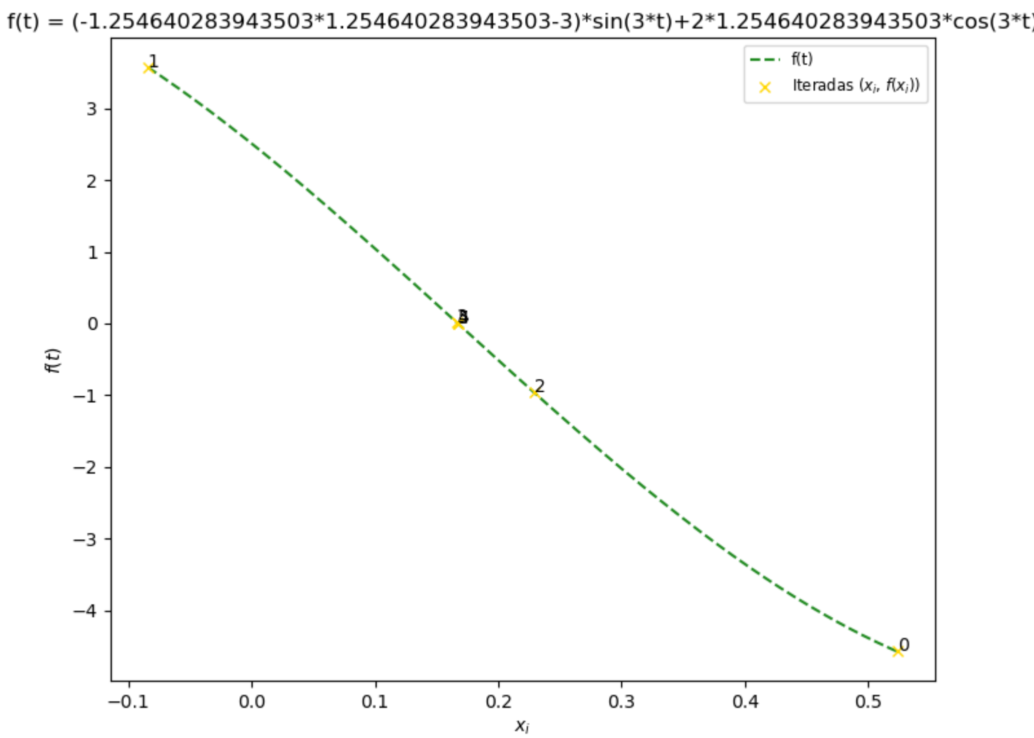

Figura 6: Gráfico produzido com as iteradas

Obtém-se assim $r^* = 0.1672512631272741$. Utilizando as equações 22 e 23 abaixo, é possível verificar que $W'(r^*) = 0$ e $W(r^*) = 2.4$, estando assim confirmado que se determinou k e r^* tal que $\underset{r \in \mathbb{R}^+}{max} W(r) = 2.4$.

$$W(r) = 2\exp(-1.254640283943503r)[1.254640283943503\sin(3r) + \cos(3r)] \tag{22}$$

$$W'(r) = 2\exp(-1.254640283943503r)[(-1.254640283943503^2 - 3)\sin(3r) + 2(1.254640283943503)\cos(3r)] \tag{23}$$

4 Grupo II

4.1 Pergunta 1. - Quadraturas

4.1.1 Alínea a

A fim de aproximar o integral $I(f) = \int_{-1}^{1} f(x)dx$, considera-se a seguinte quadratura:

$$Q(f) = A_1 f\left(-\sqrt{\frac{3}{5}}\right) + A_2 f(0) + A_3 f\left(\sqrt{\frac{3}{5}}\right) \tag{24}$$

Esta terá grau 2 se for exata (isto é, se $Q(f) = I(f)$) para qualquer polinómio de grau ≤ 2. Como tanto $I(f)$ como $Q(f)$ são transformações lineares sobre funções, basta-nos determinar A_1, A_2 e A_3 de modo a que *Q(f)* seja exata para uma base do espaço dos polinómios de grau ≤ 2 (nomeadamente a base canónica $\{1, x, x^2\}$). Logo, deve verificar-se:

$$\begin{cases} Q(1) = I(1) \\ Q(x) = I(x) \\ Q(x^2) = I(x^2) \end{cases} \tag{25}$$

Em que:

$$\begin{aligned} Q(1) &= A_1 + A_2 + A_3 \\ Q(x) &= A_1 \times \left(-\sqrt{\frac{3}{5}}\right) + A_2 \times 0 + A_3 \times \sqrt{\frac{3}{5}} \\ Q(x^2) &= A_1 \times \frac{3}{5} + A_2 \times 0 + A_3 \times \frac{3}{5} \end{aligned} \tag{26}$$

$$\begin{aligned} I(1) &= \int_{-1}^{1} 1\, dx = 2 \\ I(x) &= \int_{-1}^{1} x\, dx = 0 \\ I(x^2) &= \int_{-1}^{1} x^2 dx = \frac{2}{3} \end{aligned} \tag{27}$$

Logo tem-se:

$$\begin{cases} A_1 + A_2 + A_3 = 2 \\ A_1 \times \left(-\sqrt{\frac{3}{5}}\right) + A_2 \times 0 + A_3 \times \sqrt{\frac{3}{5}} = 0 \\ A_1 \times \frac{3}{5} + A_2 \times 0 + A_3 \times \frac{3}{5} = \frac{2}{3} \end{cases} \tag{28}$$

ou, em forma matricial:

$$\begin{bmatrix} 1 & 1 & 1 \\ -\sqrt{\frac{3}{5}} & 0 & \sqrt{\frac{3}{5}} \\ \frac{3}{5} & 0 & \frac{3}{5} \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ A_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ \frac{2}{3} \end{bmatrix} \iff \begin{bmatrix} 1 & 1 & 1 \\ 0 & -\frac{3}{5} & 0 \\ 0 & 0 & \frac{6}{5} \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ A_3 \end{bmatrix} = \begin{bmatrix} 2 \\ -\frac{8}{15} \\ \frac{2}{3} \end{bmatrix} \iff \begin{bmatrix} A_1 \\ A_2 \\ A_3 \end{bmatrix} = \begin{bmatrix} \frac{5}{9} \\ \frac{8}{9} \\ \frac{5}{9} \end{bmatrix} \tag{29}$$

Verificou-se, portanto, que $Q(f)$ só pode ter, pelo menos, grau 2 se $Q(f) = \frac{5}{9} \times f\left(-\sqrt{\frac{3}{5}}\right) + \frac{8}{9} \times f(0) + \frac{5}{9} \times f\left(\sqrt{\frac{3}{5}}\right)$. No entanto, é possível que $Q(f)$ tenha grau superior, pelo que resta verificar o seu grau máximo. Para este efeito, verificar-se-á a exatidão de Q para as funções da base canónica dos polinómios de grau $\leq n$, com n crescente, até que um falhe[2].

$$\begin{aligned} I(x^3) &= \int_{-1}^{1} x^3 \, dx = 0 \\ Q(x^3) &= \frac{5}{9} \times \left(-\sqrt{\frac{3}{5}}\right)^3 + \frac{8}{9} \times 0 + \frac{5}{9} \times \left(\sqrt{\frac{3}{5}}\right)^3 = 0 = I(x^3) \end{aligned} \tag{30}$$

$$\begin{aligned} I(x^4) &= \int_{-1}^{1} x^4 \, dx = \frac{2}{5} \\ Q(x^4) &= \frac{5}{9} \times \left(-\sqrt{\frac{3}{5}}\right)^4 + \frac{8}{9} \times 0 + \frac{5}{9} \times \left(\sqrt{\frac{3}{5}}\right)^4 = \frac{1}{5} + \frac{1}{5} = \frac{2}{5} = I(x^4) \end{aligned} \tag{31}$$

$$\begin{aligned} I(x^5) &= \int_{-1}^{1} x^5 \, dx = 0 \\ Q(x^5) &= \frac{5}{9} \times \left(-\sqrt{\frac{3}{5}}\right)^5 + \frac{8}{9} \times 0 + \frac{5}{9} \times \left(\sqrt{\frac{3}{5}}\right)^5 = 0 = I(x^5) \end{aligned} \tag{32}$$

$$\begin{aligned} I(x^6) &= \int_{-1}^{1} x^6 \, dx = \frac{2}{7} \\ Q(x^6) &= \frac{5}{9} \times \left(-\sqrt{\frac{3}{5}}\right)^6 + \frac{8}{9} \times 0 + \frac{5}{9} \times \left(\sqrt{\frac{3}{5}}\right)^6 = \frac{5 \times 3^3}{9 \times 5^3} \times 2 = \frac{6}{25} \neq I(x^6) \end{aligned} \tag{33}$$

Conclui-se, portanto, que a regra de quadratura Q com os coeficientes determinados acima é exata para os polinómios $\{1, x, x^2, x^3, x^4, x^5\}$ que formam uma base do espaço dos polinómios de grau ≤ 5, mas que não é exata para x^6 (como já se esperava), o que significa que Q tem exatamente grau 5.

[2]Dado o grau máximo de uma quadratura corresponder ao dobro do número de incógnitas consideradas (neste caso, 3: A_1, A_2 e A_3) menos 1, a quadratura não poderá, certamente, ter grau maior que 5.

4.1.2 Alínea b

O programa desenvolvido (programa 3, que se apresenta por inteiro no Anexo A) é muito semelhante ao programa 1 apresentado na secção 2.

Primeiro, é realizado o *input* do natural n - que indica o número de sub-intervalos em que se divide o intervalo para o qual se calcula o integral - e é definido o h, o tamanho de cada sub-intervalo:

```
# Input do número de passos
n = int(input("Insira o número de passos a considerar na aproximação do
↪ integral: "))
if (n < 1):
    print("Erro")
    exit()

# Definição do step size (h)
h = (tmax-tmin)/n
```

Depois, é feito um ciclo para calcular os diversos $x_{i,1}, x_{i,2}$ e $x_{i,3}$ e ir somando as respetivas imagens, sendo finalmente calculado e impresso no ecrã o valor da quadratura.

```
# Cálculo do nodo utilizado
def node1(i): return i + h*((1-sqrt(3/5))/2)
def node3(i): return i + h*((1+sqrt(3/5))/2)
def node2(i): return i + h/2

i = tmin
sum1 = 0
sum2 = 0
sum3 = 0

# Vetores a utilizar caso se queira mostrar as iterações no gráfico
#y1 = []
#y2 = []
#y3 = []
#x1 = []
#x2 = []
#x3 = []
#x4 = []
#y4 = []

while (i < tmax):
    sum1 += fu(node1(i))
    sum2 += fu(node2(i))
    sum3 += fu(node3(i))
    #y1.append(fu(node1(i)))
    #y2.append(fu(node2(i)))
```

```
131             #y3.append(fu(node3(i)))
132             #y4.append(fu(i))
133             #x1.append(i + h*((1-sqrt(3/5))/2))
134             #x2.append(i + h/2)
135             #x3.append(i + h*((1+sqrt(3/5))/2))
136             #x4.append(i)
137             i += h
138
139         Q = h/2 * (a1 * sum1 + a2 * sum2 + a3 * sum3)
140
141         print("Q(f) = ", Q)
```

As linhas comentadas que se podem observar no excerto de código acima (linhas 116-123 e 129-136) foram adicionadas com o intuito de, em conjunto com o código comentado nas linhas 153-156, marcar sobre o gráfico da função os vários pontos $x_i, x_{i,1}, x_{i,2}$ e $x_{i,3}$, tanto para efeitos didáticos (de forma a perceber intuitivamente a frequência com que a função seria "amostrada") como para efeitos de teste. Todavia, no contexto em que o programa foi empregue, revelaram-se pouco úteis e uma complicação desnecessária. A funcionalidade foi então retirada.

4.2 Pergunta 2. - Espaço percorrido por um ponto em movimento

4.2.1 Alínea a

As funções consideradas são:

$$x(t) = t \tag{34}$$

$$y(t) = te^{-\frac{t}{4}} \tag{35}$$

A partir destas, são obtidas as respetivas derivadas:

$$x'(t) = 1 \tag{36}$$

$$y'(t) = e^{-\frac{t}{4}}\left(1 - \frac{t}{4}\right) \tag{37}$$

Deste modo, o integral a calcular é o seguinte:

$$L(\tau) = \int_0^{\tau} \sqrt{1 + e^{-\frac{t}{2}}\left(1 - \frac{t}{4}\right)^2}\, dt \tag{38}$$

Seja f a função integranda em causa. Recorrendo ao programa no qual se implementou a quadratura de Gauss composta (3), é possível obter aproximações $Q_n(f)$ do integral $L(\tau)$ para diferentes valores de n. Assim, para $L(15)$ e para cada valor de n, a execução do programa elaborado (exemplificada abaixo para $n = 10$) faz-se da seguinte forma:

```
Matemática Computacional
Quadratura de Gauss Composta para Aproximação de Integrais

Insira 'man' (em vez de uma função) para ver o manual de funções e constantes
↪ suportadas ou 'quit' para terminar o programa.

Insira uma função (de uma variável t): f(t) = sqrt(1+exp(-t/2)*pow(1-t/4,2))
Insira o mínimo do intervalo de cálculo do integral: 0
Insira o máximo do intervalo de cálculo do integral: 15
Insira o número de passos a considerar na aproximação do integral: 10

Q(f) =  15.449224165648204
```

Executando o código para diferentes valores de n sugeridos no enunciado, obtêm-se as seguintes estimativas do integral $L(15)$:

$$Q_{10}(f) = 15.449224165648204$$
$$Q_{20}(f) = 15.449224225156181$$
$$Q_{40}(f) = 15.449224225721014$$
$$Q_{80}(f) = 15.449224225729189$$
$$Q_{160}(f) = 15.449224225729312$$
$$Q_{320}(f) = 15.449224225729317$$
$$Q_{640}(f) = 15.449224225729317$$
$$...$$

É de notar que os valores $Q_{320}(f)$ e $Q_{640}(f)$ obtidos são iguais, algo que se verificou mesmo considerando um número de casas decimais substancialmente superior. Deste modo, assumindo que a fórmula da quadratura em causa converge para o valor real do integral, pode admitir-se que este é obtido para $n = 320$. Aliás, recorrendo à função **scipy.integrate.quad**, presente em *Python*, a qual devolve o valor de um integral num intervalo $[a, b]$ (e um erro associado), obteve-se este mesmo resultado para o integral $L(15)$.

Todavia, verificou-se que, para valores de n superiores a 640 ($n = 1280, 2560, ...$), obtiveram-se valores de $Q_n(f)$ com oscilações em torno do valor real do integral. É de esperar, porém, que se tratem de instabilidades numéricas, devido a erros de arredondamento. Há que considerar subtrações entre números muito próximos e, deste modo, os efeitos do cancelamento subtrativo começam a manifestar-se. Por esta razão, apenas são apresentados os valores obtidos até $n = 640$.

4.2.2 Alínea b

Recorrendo a um programa auxiliar 4, derivado do programa desenvolvido no exercício anterior, cujo código se encontra no Anexo A, foram também calculadas as diferenças $Q_{2n} - Q_n$ e os quocientes $\frac{Q_{2n}-Q_n}{Q_n-Q_{\frac{n}{2}}}$.

Este programa foi executado da seguinte forma:

```
Matemática Computacional
Determinar espaço percorrido por corpo em movimento com quadratura de Gauss
↪ composta

Valor verdadeiro do integral: 15.449224225729317
Erro associado: 0.000000000292667

Insira uma das seguintes opções:
->'quit' se deseja sair do programa;
-> n, número de passos inicial (inteiro superior ou igual a 1).

10

Quantas linhas da tabela deseja ter? 7
```

```
-=-=-=-=-

Insira uma das seguintes opções:
->'quit' se deseja sair do programa;
-> n, número de passos inicial (inteiro superior ou igual a 1).

quit

A tabela foi escrita no ficheiro 'ProjetoMC_Grupo2_Exercicio2b.txt'.
```

O conteúdo do ficheiro de texto 'ProjetoMC_Grupo2_Exercicio2b.txt', resultante da execução do programa, encontra-se no Anexo B.

Eis, assim, a tabela pedida no enunciado (na qual os valores são apresentados com 15 casas decimais):

n	Q_n	$\|Q_{2n} - Q_n\|$	$\|(Q_{2n} - Q_n)/(Q_n - Q_{n/2})\|$
10	15.449224165648204	5.950797721254730e-08	-
20	15.449224225156181	5.648335132946158e-10	9.491727659926581e-03
40	15.449224225721014	8.174794174919953e-12	1.447292694662754e-02
80	15.449224225729189	1.225686219186173e-13	1.499348109517601e-02
160	15.449224225729312	5.329070518200751e-15	4.347826086956522e-02
320	15.449224225729317	0	0
640	15.449224225729317	...	...

Tabela 4: Valores absolutos de diferenças e quocientes envolvendo valores aproximados do integral $L(15)$.

Pela mesma razão que a indicada na alínea anterior, são apenas apresentados na Tabela 4 os valores obtidos até $n = 640$.

4.2.3 Alínea c

Em primeiro lugar, de forma a que possa aplicar a majoração $|E_n(f)| \leq Ch^\alpha$ dada, é imposta a condição que a função integranda seja "suficientemente regular". Fazendo *plot* da função $f(t)$ entre $t = 0$ e $t = 15$, recorrendo ao programa 4 elaborado para a alínea anterior, é possível verificar que esta condição é satisfeita. Aliás, a função f é dada pela raiz quadrada da soma de 1 com o produto de duas funções C^∞ em $[0, 15]$, e o radicando é sempre positivo, pelo que a função f é C^∞ em $[0, 15]$.

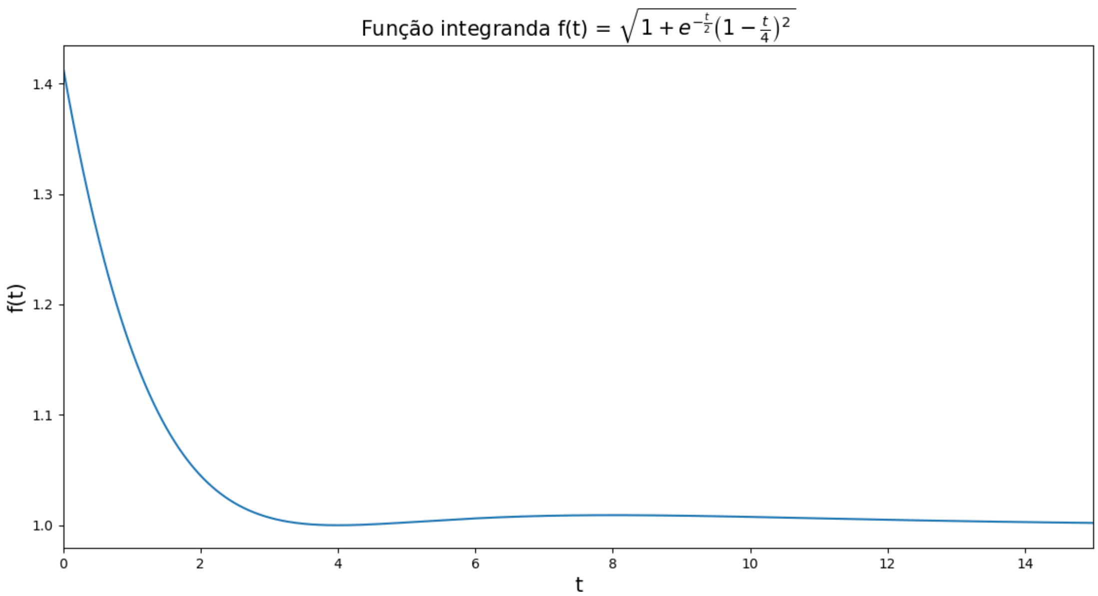

Figura 7: Gráfico da função integranda entre $t = 0$ e $t = 15$.

Procura-se, de seguida, encontrar uma expressão que relacione valores da Tabela 4 com o valor de α. Tem-se, por um lado, que $h = \frac{b-a}{n} = \frac{15}{n}$. Relembre-se também a desigualdade triangular, que, para dois números reais x_1 e x_2, é dada por $|x_1 + x_2| \leq |x_1| + |x_2|$.

A majoração $|E_n(f)| \leq Ch^\alpha$ pode ser aplicada, com as mesmas constantes C e α, para diferentes valores de n, visto que estas não dependem de h. Assim, têm-se as seguintes relações:

$$\begin{aligned} |Q_n - Q_{\frac{n}{2}}| &= |Q_n - I + I - Q_{\frac{n}{2}}| \\ &\leq |I - Q_n| + |I - Q_{\frac{n}{2}}| \\ &\leq C\left(\frac{15}{n}\right)^\alpha + C\left(\frac{15}{\frac{n}{2}}\right)^\alpha \\ &= C\left(\frac{15}{n}\right)^\alpha (1 + 2^\alpha) \end{aligned} \tag{39}$$

$$\begin{aligned} |Q_{2n} - Q_n| &= |Q_{2n} - I + I - Q_n| \\ &\leq |I - Q_{2n}| + |I - Q_n| \\ &\leq C\left(\frac{15}{2n}\right)^\alpha + C\left(\frac{15}{n}\right)^\alpha \\ &= C\left(\frac{15}{n}\right)^\alpha (2^{-\alpha} + 1) \end{aligned} \tag{40}$$

Ora, admitindo que as majorações em (39) e (40) oferecem boas estimativas de erro, pode considerar-se a seguinte relação para os valores apresentados na Tabela 4:

$$\frac{|Q_{2n}-Q_n|}{|Q_n-Q_{\frac{n}{2}}|} \sim \frac{2^{-\alpha}+1}{1+2^{\alpha}} = \frac{\frac{1+2^{\alpha}}{2^{\alpha}}}{1+2^{\alpha}} = 2^{-\alpha}, \quad n = 20, 40, 80, 160 \tag{41}$$

É claro que, para $n = 320$, obtendo-se um erro nulo, não será útil utilizar a estimativa dada em (41).

Assim, para cada valor x_n ($n = 20, 40, 80, 160$) da quarta coluna da tabela, tem-se a estimativa $x_n \sim 2^{-\alpha}$, pelo que $\alpha \sim -log_2 x_n$. Comprova-se, então, que o valor de α não dependerá do valor de n e, consequentemente, do valor de h.

Obtêm-se os seguintes valores aproximados para α:

$$n = 20 : \alpha \approx 6.7$$
$$n = 40 : \alpha \approx 6.1$$
$$n = 80 : \alpha \approx 6.1$$
$$n = 160 : \alpha \approx 4.5$$

Seria de esperar, à partida, um valor de α semelhante para os quatro casos. Contudo, verifica-se uma clara discrepância do último valor face aos restantes. Porém, é também de notar que, com $n = 160$ e valores de n superiores, as aproximações são já muito próximas entre si. Para estimar α, há que considerar subtrações entre números muito próximos; deste modo, já serão mais acentuados os efeitos dos erros computacionais e os efeitos do cancelamento subtrativo começam a manifestar-se. Logo, de forma a se poder deduzir o valor de α, não será útil recorrer a este último valor. Neste caso, a ordem de convergência do método é elevada, e muito rapidamente se calculam aproximações com grande precisão. Portanto, a amostra para estimar a ordem de precisão não é muito numerosa.

Mesmo assim, obtêm-se duas aproximações distintas para o valor de α, sendo estas $\alpha \sim 6$ (para $n = 40$ e $n = 80$) e $\alpha \sim 7$ (para $n = 20$). Porém, sendo os valores $Q_n(f)$ mais precisos para valores de n superiores, será mais apropriado considerar as estimativas obtidas com $n = 40$ e $n = 80$; com $n = 20$, o valor de h é ainda demasiado elevado. Este aspeto será também aparente ao obter os valores de α para diferentes regras de quadratura compostas, a partir das Tabelas 5, 6 e 7 apresentadas abaixo.

Em suma, pode ser deduzido que $\alpha = 6$ para a quadratura de Gauss composta utilizada.

De seguida, esta será comparada com as outras regras de quadratura compostas estudadas:

- regra dos trapézios composta;
- regra de Simpson composta;
- regra do ponto médio composta ou regra dos retângulos.

A fórmula de quadratura para a regra dos trapézios composta, com $h = \frac{b-a}{n}$ e $x_i = a + ih$ ($i = 0, ..., n$), é dada por

$$T_n(f) = \frac{h}{2}\sum_{n=0}^{n-1}[f(x_i) + f(x_{i+1})] = \frac{h}{2}\left[f(x_0) + f(x_n) + 2\sum_{i=1}^{n-1} f(x_i)\right] \tag{42}$$

e a respetiva fórmula de erro, para $f \in C^2([a, b])$, por

$$E_n^T(f) = -\frac{(b-a)h^2}{12} f''(\xi), \quad \xi \in [a,b] \tag{43}$$

Pelo que $|E_n^T(f)| \leq C_T h^2$, sendo C_T uma constante independente de h. Diz-se que a regra dos trapézios composta tem ordem de precisão (ou convergência) igual a 2 ou precisão de segunda ordem em h e representa-se por $|E_n^T(f)| = \mathcal{O}(h^2)$.

Para a regra de Simpson composta, a fórmula de quadratura é

$$S_n(f) = \frac{h}{3}\left[f(x_0) + f(x_n) + 4\sum_{n=1}^{n/2} f(x_{2i-1}) + 2\sum_{n=1}^{n/2-1} f(x_{2i})\right] \tag{44}$$

E a fórmula de erro, para $f \in C^4([a,b])$, é dada por

$$E_n^S(f) = -\frac{(b-a)h^4}{180} f^{(4)}(\xi), \quad \xi \in [a,b] \tag{45}$$

Deste modo, $|E_n^S(f)| \leq C_S h^4$, sendo C_S uma constante independente de h, e a fórmula S_n tem ordem de precisão igual a 4 (ou seja, $|E_n^S(f)| = \mathcal{O}(h^4)$).

Finalmente, sendo $h = \frac{b-a}{n}$ e $x_i = a + \frac{h}{2} + ih$ $(i = 0, ..., n-1)$ a aproximação para o integral pela regra dos retângulos é dada por

$$\int_a^b f(x)dx \approx h\sum_{i=0}^{n-1} f(x_i) \tag{46}$$

Sendo que a regra dos retângulos tem ordem de convergência 2, de acordo com a fórmula de erro

$$\int_a^b f(x)dx - h\sum_{i=0}^{n-1} f(x_i) = \frac{b-a}{24} h^2 f''(\xi), \quad \xi \in [a,b] \tag{47}$$

De acordo com a estimativa $\alpha = 6$ obtida, infere-se que $|E_n(f)| = \mathcal{O}(h^6)$, ou seja, a quadratura de Gauss composta tem ordem de convergência 6. Consequentemente, conclui-se que a quadratura de Gauss apresenta uma eficiência muito superior à das restantes quadraturas utilizadas, pelo que os sucessivos valores aproximados que se calculam à medida que se aumenta o valor de n convergem muito mais rapidamente para o valor real do integral. Mesmo para valores de n relativamente baixos (por exemplo, $n = 10$) já é obtida uma aproximação com 6 casas decimais corretas.

De forma a tornar aparentes as afirmações apresentadas anteriormente, foi realizado outro programa auxiliar (Programa 5), cujo código se encontra no Anexo A. Neste, para além de se calcularem as quadraturas de Gauss, já implementadas anteriormente, recorreu-se às fórmulas (42), (44) e (46), de forma a calcular valores aproximados do integral $L(15)$, para vários valores de n, pelas diferentes fórmulas de quadratura estudadas.

Abaixo, é apresentada a execução do programa necessária para a obtenção das Tabelas 5, 6 e 7.

```
Matemática Computacional
Determinar espaço percorrido por corpo em movimento com diferentes quadraturas

Valor verdadeiro do integral: 15.449224225729317
Erro associado: 0.000000000292667

Insira uma das seguintes opções:
->'quit' se deseja sair do programa;
-> n, número de passos inicial (inteiro superior ou igual a 1).

10

Quantas iteradas deseja realizar? 20

->Seleção de regras de quadratura
Digite 'Enter' para 'Sim', ou qualquer outro input para 'Não'.
Deseja aplicar a regra de quadratura de Gauss composta? n
Deseja aplicar a regra dos trapézios composta?
Deseja aplicar a regra de Simpson composta? n
Deseja aplicar a regra do ponto médio composta? n

Digite:
->'i' se desejar obter gráficos das sucessivas iteradas;
->'e' se desejar obter gráficos dos módulos dos respetivos erros;
-> algo sem 'i' nem 'e', se não desejar obter qualquer gráfico.

n

-=-=-=-=-

Insira uma das seguintes opções:
->'quit' se deseja sair do programa;
-> n, número de passos inicial (inteiro superior ou igual a 1).

10

Quantas iteradas deseja realizar? 16

->Seleção de regras de quadratura
Digite 'Enter' para 'Sim', ou qualquer outro input para 'Não'.
Deseja aplicar a regra de quadratura de Gauss composta? n
Deseja aplicar a regra dos trapézios composta? n
Deseja aplicar a regra de Simpson composta?
```

```
Deseja aplicar a regra do ponto médio composta? n

Digite:
->'i' se desejar obter gráficos das sucessivas iteradas;
->'e' se desejar obter gráficos dos módulos dos respetivos erros;
-> algo sem 'i' nem 'e', se não desejar obter qualquer gráfico.

n

-=-=-=-=-

Insira uma das seguintes opções:
->'quit' se deseja sair do programa;
-> n, número de passos inicial (inteiro superior ou igual a 1).

10

Quantas iteradas deseja realizar? 20

->Seleção de regras de quadratura
Digite 'Enter' para 'Sim', ou qualquer outro input para 'Não'.
Deseja aplicar a regra de quadratura de Gauss composta? n
Deseja aplicar a regra dos trapézios composta? n
Deseja aplicar a regra de Simpson composta? n
Deseja aplicar a regra do ponto médio composta?

Digite:
->'i' se desejar obter gráficos das sucessivas iteradas;
->'e' se desejar obter gráficos dos módulos dos respetivos erros;
-> algo sem 'i' nem 'e', se não desejar obter qualquer gráfico.

n

-=-=-=-=-

Insira uma das seguintes opções:
->'quit' se deseja sair do programa;
-> n, número de passos inicial (inteiro superior ou igual a 1).

quit

As tabelas foram escritas no ficheiro 'ProjetoMC_Grupo2_Exercicio2c.txt'.
```

O conteúdo do ficheiro de texto 'ProjetoMC_Grupo2_Exercicio2c.txt', resultante da execução do programa, encontra-se no Anexo B.

Para a regra dos trapézios composta, obteve-se, assim, a seguinte tabela:

n	Q_n	$\lvert Q_{2n} - Q_n\rvert$	$\lvert(Q_{2n} - Q_n)/(Q_n - Q_{n/2})\rvert$
10	15.514850849804633	4.911887370212931e-02	-
20	15.465731976102504	1.237445121426539e-02	2.519286433420186e-01
40	15.453357524888238	3.099576353031708e-03	2.504819243586726e-01
80	15.450257948535207	7.752672223304558e-04	2.501203822813282e-01
160	15.449482681312876	1.938401324359518e-04	2.500300887908916e-01
320	15.449288841180440	4.846149114534626e-05	2.500075218497841e-01
640	15.449240379689295	1.211546392632101e-05	2.500018806681820e-01
...	...	...	...
2621440	15.449224225730143	2.717825964282383e-13	1.959026888604353e-01
5242880	15.449224225729871	...	...

Tabela 5: Valores absolutos de diferenças e quocientes envolvendo valores aproximados do integral $L(15)$ para a regra dos trapézios composta.

O programa foi executado para o máximo n no qual o tempo de execução seria ainda "aceitável".

Desde já, na Tabela 5, verifica-se uma grande similitude entre os valores da quarta coluna, algo que seria esperado acontecer, assumindo boas estimativas dadas por (41). Aplicando aqui a relação $\alpha \sim -log_2 x_n$, sendo x_n os diferentes valores da quarta coluna, obtém-se $\alpha \approx 2$, que é o valor esperado tendo em conta a fórmula de erro (43). Deste modo, corrobora-se a validade da estimativa em (41) utilizada para determinar α para a quadratura de Gauss. Por outro lado, também na Tabela 5 se verifica uma ligeira discrepância no valor de α obtido para o valor de n da segunda linha e uma discrepância maior no valor de α obtido para a penúltima linha da tabela, de forma análoga ao que sucedeu para a tabela relativa à quadratura de Gauss.

Relativamente à precisão dos resultados, verifica-se que, com $n = 5242880$ (um valor 16384 vezes superior a $n = 320$), apenas se têm 12 casas decimais corretas, enquanto que, para a regra de quadratura de Gauss, para $n = 320$, já se considerara ter chegado ao valor real do integral. Por outro lado, com $n = 10$, a regra dos trapézios fornece um valor aproximado sem qualquer casa decimal correta, enquanto que a quadratura de Gauss produziu um valor aproximado com exatidão de 6 casas decimais.

Para a regra de Simpson composta, a tabela é a seguinte:

n	Q_n	$\lvert Q_{2n} - Q_n \rvert$	$\lvert (Q_{2n} - Q_n)/(Q_n - Q_{n/2}) \rvert$
10	15.451576779535070	2.217761333270118e-03	-
20	15.449359018201800	1.263103849886704e-04	5.695400271156504e-02
40	15.449232707816812	7.951399279448879e-06	6.295127103097731e-02
80	15.449224756417532	4.975120990735604e-07	6.256912545687728e-02
160	15.449224258905433	3.110248414373018e-08	6.251603569369973e-02
320	15.449224227802949	1.944030714184919e-09	6.250403360712935e-02
640	15.449224225858918	1.214974787444589e-10	6.249771562657622e-02
...	...	...	...
163840	15.449224225729273	3.019806626980426e-14	4.473684210526316e-01
327680	15.449224225729242	...	...

Tabela 6: Valores absolutos de diferenças e quocientes envolvendo valores aproximados do integral $L(15)$ para a regra de Simpson composta.

Neste caso, são apresentados os valores que se obtiveram até $n = 327680$, visto que, para valores de n superiores, verificou-se que acontecia algo semelhante ao sucedido com a quadratura de Gauss: os efeitos do cancelamento subtrativo faziam com que os valores de Q_n se afastassem do valor real do integral.

Aplicando aos valores da quarta coluna da Tabela 6 a relação $\alpha \sim -log_2 x_n$, obtêm-se valores $\alpha \approx 4$, tal como seria esperado, tendo em conta a equação 45. Verificam-se também discrepâncias para o primeiro e último valores da quarta coluna.

Neste método, têm-se, para $n = 327680$ (um número $1024 = 2^{10}$ vezes superior a $n = 320$, ou seja, requerendo mais 10 iterações), 12 casas decimais corretas. Por outro lado, com $n = 1$, obtém-se já 1 casa decimal correta. Os valores da terceira coluna da Tabela 6 são mais pequenos do que os da Tabela 5, o que indica que mais rapidamente se chegam a valores mais precisos, logo diferindo dos seguintes em menos casas decimais; porém, estes valores são bastante superiores aos da tabela da quadratura de Gauss composta. Assim, embora esta fórmula de quadratura apresente maior rapidez de convergência do que a regra dos trapézios, é bastante menos eficiente do que a quadratura de Gauss.

Por fim, para a regra dos retângulos:

n	Q_n	$\lvert Q_{2n} - Q_n \rvert$	$\lvert (Q_{2n} - Q_n)/(Q_n - Q_{n/2}) \rvert$
10	15.416613102400387	2.436997127357898e-02	-
20	15.440983073673966	6.175298508216187e-03	2.533978575063491e-01
40	15.447158372182182	1.549041908360138e-03	2.508448630133668e-01
80	15.448707414090542	3.875869574443414e-04	2.502107627640962e-01
160	15.449095001047986	9.691715018078639e-05	2.500526612655793e-01
320	15.449191918198167	2.423056331224416e-05	2.500131634808203e-01
640	15.449216148761479	6.057720572272274e-06	2.500032910589080e-01
...	...	...	...
2621440	15.449224225728702	7.993605777301127e-13	2.675386444708680e-01
5242880	15.449224225729502	...	...

Tabela 7: Valores absolutos de diferenças e quocientes envolvendo valores aproximados do integral $L(15)$ para a regra dos retângulos.

Também aqui foi necessário limitar o valor máximo de n - de modo a permitir um tempo de execução aceitável para o programa. Finalmente, também nesta tabela se obtém o valor esperado para α, recorrendo aos valores da quarta coluna ($\alpha \approx 2$). A rapidez deste método de convergência é, como esperado, análoga à da regra dos trapézios composta (ambas possuem ordem de precisão 2), pelo que bastante inferior à da quadratura de Gauss composta. Só são atingidas, com um valor $n = 5242880$, 12 casas decimais corretas. Mesmo assim, é de apontar que a regra dos retângulos é ligeiramente mais rápida a convergir do que a regra dos trapézios composta. Verifica-se que, na regra dos retângulos, para um dado valor de n, os valores de Q_n são, no geral, mais exatos, e os valores de $|Q_{2n} - Q_n|$ mais baixos. Isto seria de esperar, tendo em conta o denominador 24 na equação 47, face ao denominador 12 da equação 43.

Para além destas tabelas, o programa realizado realiza um *plot* conjunto dos valores $Q_n(f)$ obtidos para as diferentes regras de quadratura. A partir da Figura 8 obtida, é possível comprovar visualmente a rapidez de convergência para os vários métodos considerados. É aparente que a regra dos trapézios composta apresenta a menor rapidez de convergência, seguida da regra dos retângulos; a regra de Simpson composta, embora bastante mais rápida do que as anteriores, não é tão eficiente como a quadratura de Gauss composta.

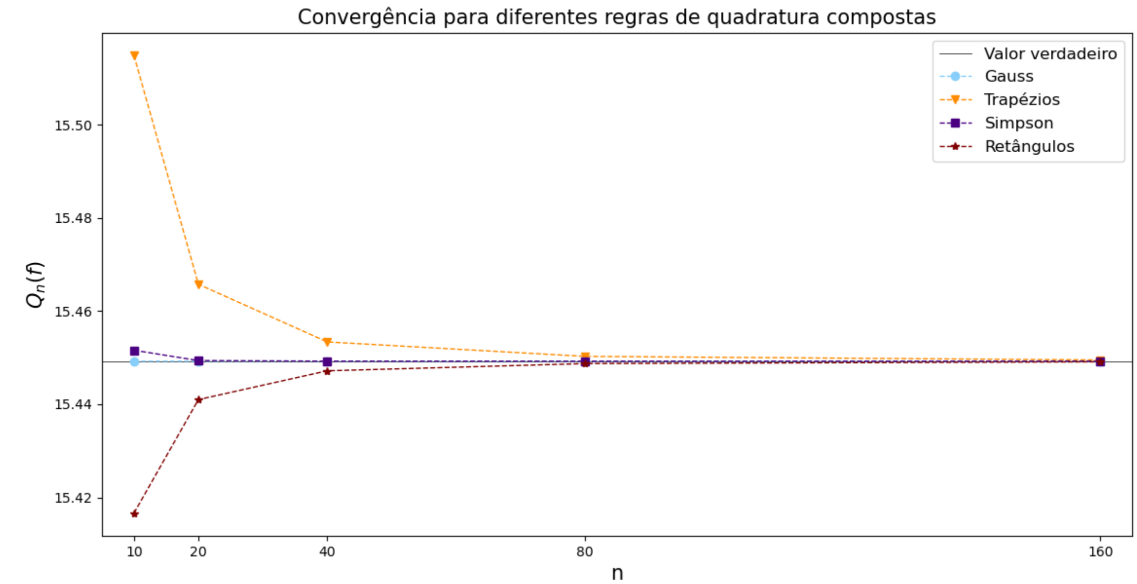

Figura 8: Valores aproximados do integral para diferentes regras de quadratura compostas

De forma a tornar mais evidente a discrepância entre a ordem de convergência para as regras de quadratura de Gauss e de Simpson compostas, eis a seguinte figura:

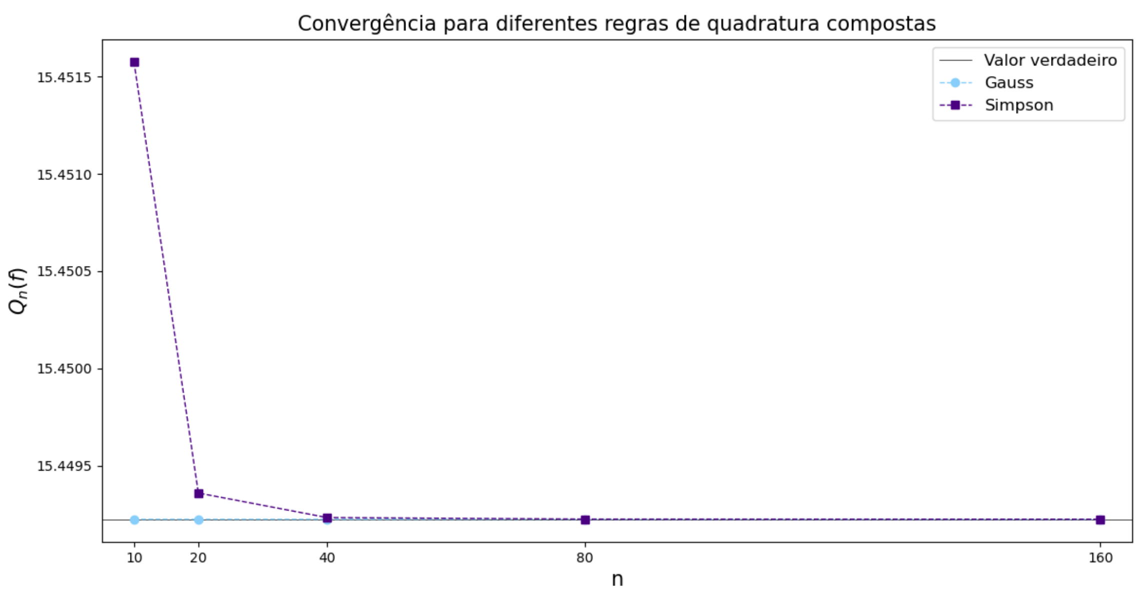

Figura 9: Valores aproximados do integral para a quadratura de Gauss composta e para a regra de Simpson composta

Podem também verificar-se as diferentes ordens de convergência na Figura 10, abaixo apresentada.

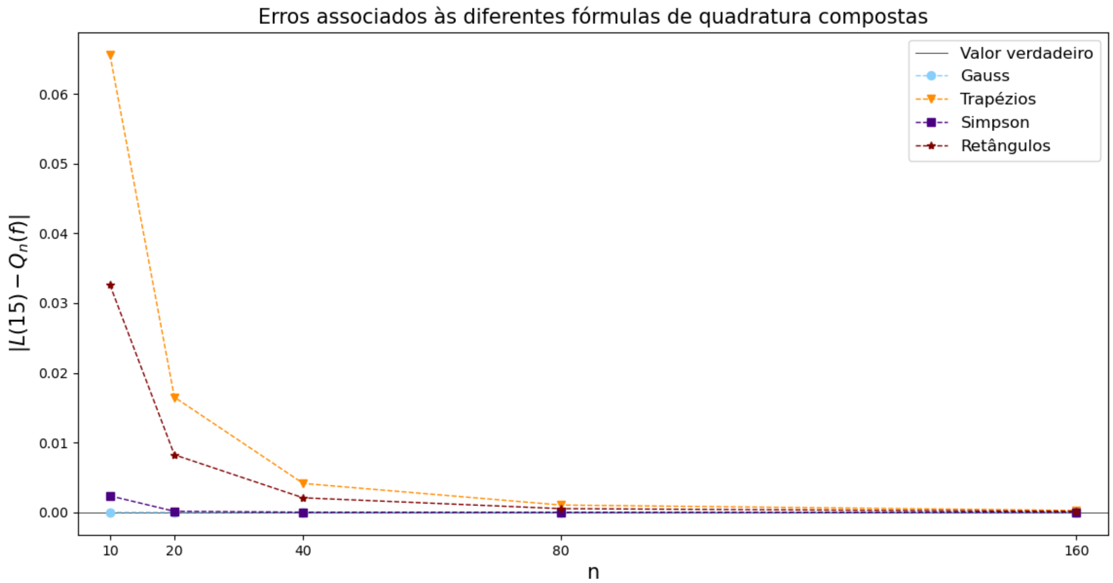

Figura 10: Módulos das diferenças entre o valor real do integral e as diferentes aproximações

Nesta, são aparentes linhas semelhantes a ajustes do tipo $k\left(\frac{1}{n}\right)^p$, $k \in \mathbb{R}^+$, sendo p a ordem de convergência do respetivo método.

A execução do programa 5 necessária para a obtenção das Figuras 8, 9 e 10 é a seguinte:

```
Matemática Computacional
Determinar espaço percorrido por corpo em movimento com diferentes quadraturas

Valor verdadeiro do integral: 15.449224225729317
Erro associado: 0.000000000292667

Insira uma das seguintes opções:
->'quit' se deseja sair do programa;
-> n, número de passos inicial (inteiro superior ou igual a 1).

10

Quantas iteradas deseja realizar? 5

->Seleção de regras de quadratura
Digite 'Enter' para 'Sim', ou qualquer outro input para 'Não'.
Deseja aplicar a regra de quadratura de Gauss composta?
Deseja aplicar a regra dos trapézios composta?
```

```
Deseja aplicar a regra de Simpson composta?
Deseja aplicar a regra do ponto médio composta?

Digite:
->'i' se desejar obter gráficos das sucessivas iteradas;
->'e' se desejar obter gráficos dos módulos dos respetivos erros;
-> algo sem 'i' nem 'e', se não desejar obter qualquer gráfico.

ie

-=-=-=-=-

Insira uma das seguintes opções:
->'quit' se deseja sair do programa;
-> n, número de passos inicial (inteiro superior ou igual a 1).

10

Quantas iteradas deseja realizar? 5

->Seleção de regras de quadratura
Digite 'Enter' para 'Sim', ou qualquer outro input para 'Não'.
Deseja aplicar a regra de quadratura de Gauss composta?
Deseja aplicar a regra dos trapézios composta? n
Deseja aplicar a regra de Simpson composta?
Deseja aplicar a regra do ponto médio composta? n

Digite:
->'i' se desejar obter gráficos das sucessivas iteradas;
->'e' se desejar obter gráficos dos módulos dos respetivos erros;
-> algo sem 'i' nem 'e', se não desejar obter qualquer gráfico.

i

-=-=-=-=-

Insira uma das seguintes opções:
->'quit' se deseja sair do programa;
-> n, número de passos inicial (inteiro superior ou igual a 1).

quit

As tabelas foram escritas no ficheiro 'ProjetoMC_Grupo2_Exercicio2c.txt'.
```

Está incluído no Anexo B o conteúdo do ficheiro de texto resultante desta execução do programa.

4.2.4 Alínea d

Utilizando a equação (40), e considerando $\alpha = 6$, tem-se que $|Q_n - Q_{2n}| \leq Ch^6(2^{-6}+1)$. Usando o valor mais elevado presente na terceira coluna da tabela 4, pode-se estimar um valor de C.

$$|Q_{10} - Q_{20}| \leq C \left(\frac{15}{10}\right)^6 (2^{-6}+1) \iff \left(\frac{10}{15}\right)^6 \frac{5.950797721254730e-08}{2^{-6}+1} \leq C \tag{48}$$

Majorando, e de forma a facilitar os cálculos, pode-se considerar $C = \frac{1}{2^{-6}+1}$. Obtém-se assim $|Q_n - Q_{2n}| \leq h^6$. Neste caso, é necessário escolher h tal que $h^6 \leq 10^{-6}$. Os intervalos considerados têm como extremo mínimo 0 e têm como extremos máximos múltiplos de 0.15, logo, com $i \in 1, ..., 100$, $\left(\frac{0.15i}{n}\right)^6 \leq 10^{-6}$, obtendo-se $n \geq 1.5i$.

Por exemplo, para $i = 1$, tem de se utilizar $n = 2$, obtendo-se o seguinte resultado utilizando o programa 3 da questão 1:

```
Matemática Computacional

Quadratura de Gauss Composta para Aproximação de Integrais


Insira 'man' (em vez de uma função) para ver o manual de funções e constantes
↪ suportadas ou 'quit' para terminar o programa.

Insira uma função (de uma variável t): f(t) = sqrt(1+exp(-t/2)*pow(1-t/4,2))
Insira o mínimo do intervalo de cálculo do integral: 0
Insira o máximo do intervalo de cálculo do integral: 0.15
Insira o número de passos a considerar na aproximação do integral: 2
Q(f) =  0.20827722073773694


Insira 'man' (em vez de uma função) para ver o manual de funções e constantes
↪ suportadas ou 'quit' para terminar o programa.

Insira uma função (de uma variável t): f(t) = quit
```

Desenvolveu-se um programa 6, tendo como base o programa do exercício 1 deste grupo, para calcular os integrais pretendidos e desenhar o gráfico. Utilizando esse programa, constatou-se que para alguns valores de τ, não seria respeitado $|Q_n - Q_{2n}| \leq 10^{-6}$. Utilizando $n \geq 1.5i$, para alguns valores de τ, n é demasiado grande, o que pode afetar os resultados pois, como já foi referido, pode causar instabilidades numéricas devido a erros de arredondamento.

Mesmo considerando diferentes majorações para C, continuou a constatar-se que, para alguns valores de τ, $|Q_n - Q_{2n}| > 10^{-6}$.

Consequentemente, decidiu-se que o programa calcularia o integral para $n = 1$, verificando se $|Q_n - Q_{2n}| \leq 10^{-6}$. Caso a condição não seja verificada, n é incrementado, repetindo-se esta operação até

$|Q_n - Q_{2n}| \leq 10^{-6}$. Com este programa, verificou-se que o n máximo utilizado foi 9, algo que sugere que o n sugerido inicialmente seria demasiado elevado.

Obteve-se, assim, o gráfico pedido:

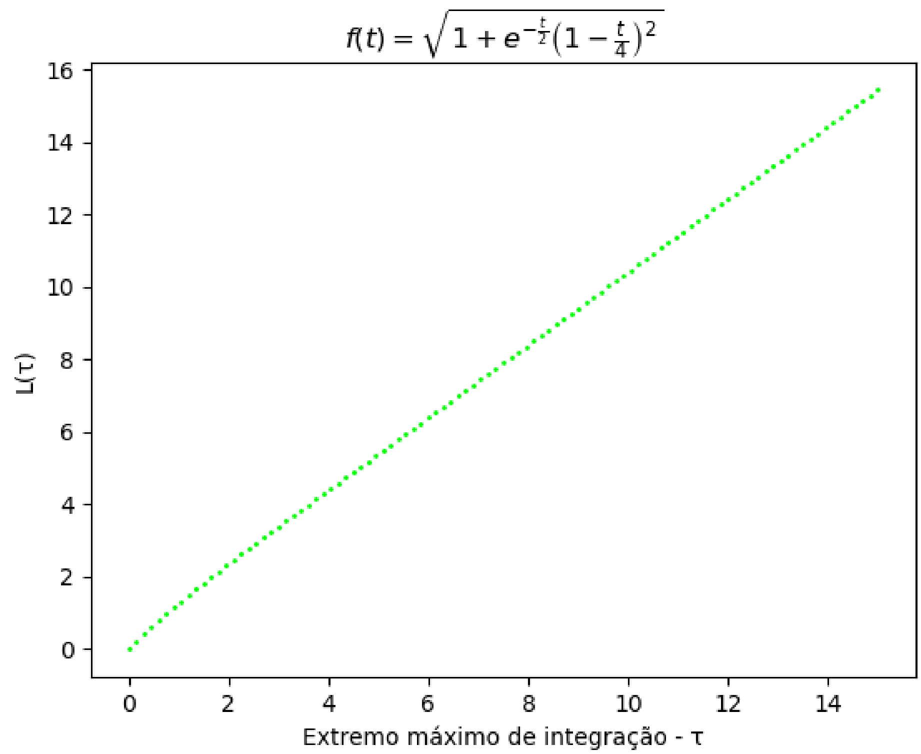

Figura 11: Gráfico de $L(\tau)$ em função de τ

5 Grupo III

5.1 Pergunta 1. - Método Runge Kutta

Considerando um problema de valor inicial, de resolução de Equações Diferenciais Ordinárias:

$$\begin{cases} y'(t) = f(t, y(t)),\ t \in [a, b] \\ y(a) = y_a \end{cases} \tag{49}$$

Um método de Runge Kutta para aproximação da solução desta equação é, considerando n passos:

$$\begin{cases} y_0 = y_a,\ h = \frac{b-a}{n} \\ k_i^{(1)} = f(t_i, y_i),\ k_i^{(2)} = f(t_i + \frac{h}{3},\ y_i + \frac{h}{3}k_i^{(1)}),\ k_i^{(3)} = f(t_i + \frac{2h}{3},\ y_i + \frac{2h}{3}k_i^{(2)}) \\ y_{i+1} = y_i + \frac{h}{4}(k_i^{(1)} + 3k_i^{(3)}),\ i = 0, ..., n-1 \end{cases} \tag{50}$$

Este foi implementado no código 7, presente no anexo A. Analogamente ao código inicialmente apresentado (1), são disponibilizadas uma série de funções e constantes, apresentadas num manual, se pedido. É, de seguida, definido o método de Runge Kutta. Enquanto o utilizador não sair do programa, inserindo quit, é-lhe pedida uma função, um intervalo, o valor da função no extremo inicial do intervalo e o valor de n desejado. É calculado também o valor de h. É definido o cálculo da função $f(t)$ e, finalmente, são calculadas as iteradas, apresentadas tanto como *output*, como num gráfico.

Abaixo é apresentado um exemplo de execução do programa:

```
Matemática Computacional
Método Runge-Kutta



A função deve ser inserida em função de t e de y
<man> para opções ou <quit> para sair
Insira uma função: f(t, y) = 1
Insira o t mínimo: -10
Insira o t máximo: 10
Insira o valor inicial da função: -10
Insira o n pretendido: 20

---- Aproximações (t, y) ----

       -10,        -10
        -9,         -9
        -8,         -8
        -7,         -7
        -6,         -6
        -5,         -5
        -4,         -4
        -3,         -3
        -2,         -2
```

```
     -1,         -1
      0,          0
      1,          1
      2,          2
      3,          3
      4,          4
      5,          5
      6,          6
      7,          7
      8,          8
      9,          9
     10,         10


A função deve ser inserida em função de t e de y
<man> para opções ou <quit> para sair
Insira uma função: f(t, y) = 2*t
Insira o t mínimo: 0
Insira o t máximo: 10
Insira o valor inicial da função: 0
Insira o n pretendido: 20

---- Aproximações (t, y) ----

      0,          0
    0.5,       0.25
      1,          1
    1.5,       2.25
      2,          4
    2.5,       6.25
      3,          9
    3.5,      12.25
      4,         16
    4.5,      20.25
      5,         25
    5.5,      30.25
      6,         36
    6.5,      42.25
      7,         49
    7.5,      56.25
      8,         64
    8.5,      72.25
      9,         81
    9.5,      90.25
     10,        100
```

```
A função deve ser inserida em função de t e de y
<man> para opções ou <quit> para sair
Insira uma função: f(t, y) = quit
```

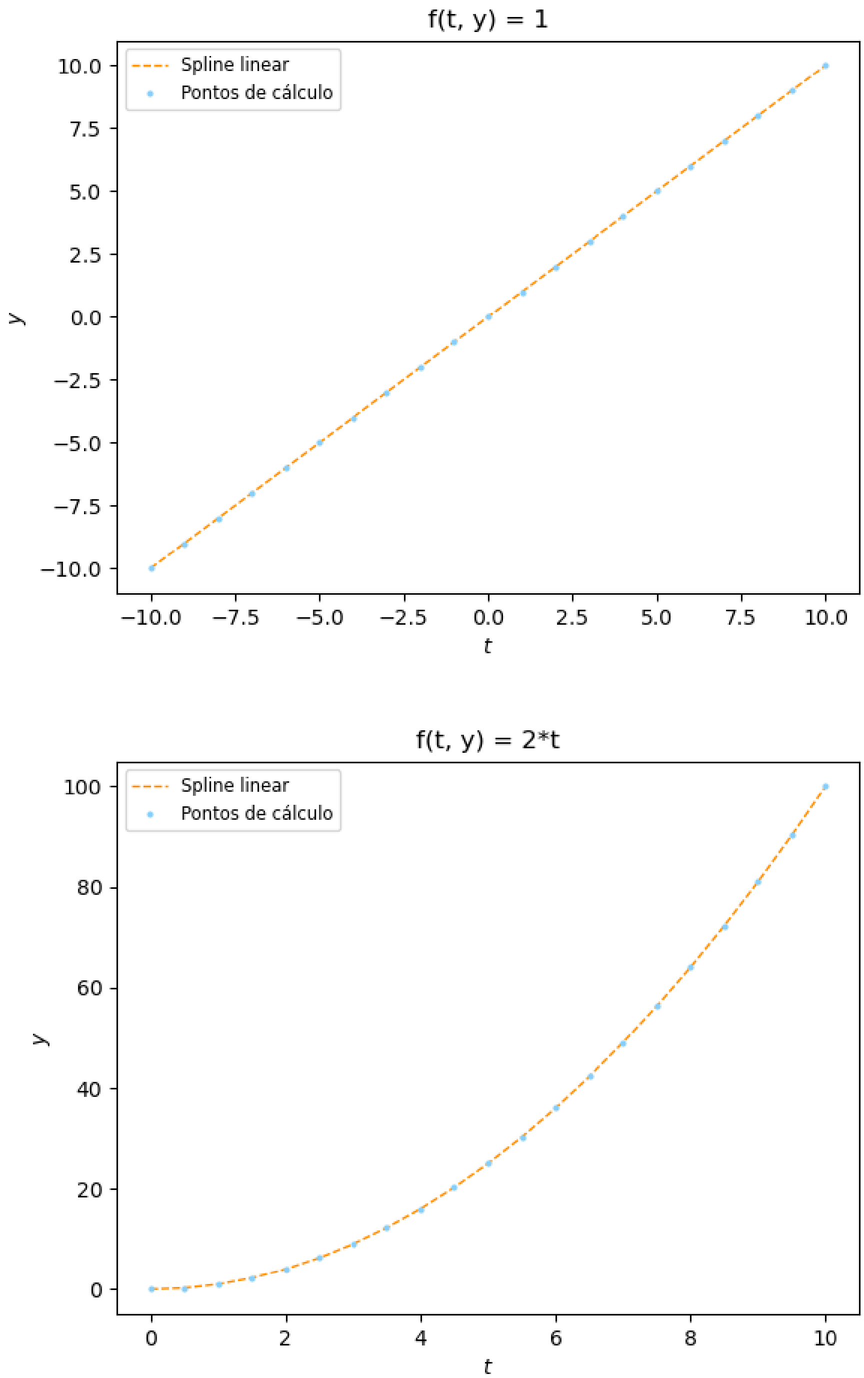

Figura 12: Aplicações do método Runge-Kutta

5.2 Pergunta 2. - Modelo epidémico SEIRP

Através de modelos epidémicos, é possível modelar a evolução de uma pandemia, considerando vários fatores externos. Neste caso, será considerado um modelo *SEIRP* [2] para descrever a evolução da pandemia de COVID-19, ou SARS-CoV-2. Este subdivide a população humana, num dado dia t, em cinco grupos, para além de considerar a população de patógenos (considerando que o vírus pode ser transmitido não só entre humanos, mas também do meio ambiente para o humano). Tem-se:

$$N(t) = S(t) + E(t) + I_A(t) + I_S(t) + R(t) \tag{51}$$

- $N(t)$: número total de indivíduos
- $S(t)$: grupo suscetível
- $E(t)$: grupo exposto
- $I_A(t)$: infecciosos assintomáticos
- $I_S(t)$: infecciosos sintomáticos
- $R(t)$: indivíduos recuperados
- $P(t)$: população de patógenos

A dinâmica da transmissão do vírus é descrita pelo seguinte sistema de equações diferenciais ordinárias:

$$\begin{cases} S'(t) = b - \frac{\beta_1 S(t)P(t)}{1+\alpha_1 P(t)} - \frac{B_2 S(t)(I_A(t)+I_S(t))}{1+\alpha_2(I_A(t)+I_S(t))} + \psi E(t) - \mu S(t) \\ E'(t) = \frac{\beta_1 S(t)P(t)}{1+\alpha_1 P(t)} + \frac{\beta_2 S(t)(I_A(t)+I_S(t))}{1+\alpha_2(I_A(t)+I_S(t))} - \psi E(t) - \mu E(t) - \omega E(t) \\ I'_A(t) = (1-\delta)\omega E(t) - (\mu+\sigma)I_A(t) - \gamma_A I_A(t) \\ I'_S(t) = \delta\omega E(t) - (\mu+\sigma)I_S(t) - \gamma_S I_S(t) \\ R'(t) = \gamma_A I_A(t) + \gamma_S I_S(t) - \mu R(t) \\ P'(t) = \eta_A I_A(t) + \eta_S I_S(t) - \mu_P P(t) \end{cases} \tag{52}$$

O comportamento deste sistema é regido por diversos parâmetros:

- Taxa de natalidade da população humana (por dia): b
- Taxa de mortalidade (natural) humana (por dia): μ
- Taxa de mortalidade natural de patógenos no meio ambiente (por dia): μ_P
- Recíproco da proporção de interação com um ambiente infeccioso: α_1
- Recíproco da proporção de interação com um indivíduo infeccioso: α_2
- Taxa de transmissão de S para E devido ao contacto com P: β_1
- Taxa de transmissão de S para E devido a contacto com I_A e/ou I_S: β_2
- Proporção de indivíduos infecciosos sintomáticos: δ
- Taxa de progressão de E para S devido a sistema imunológico robusto: ψ
- Taxa de progressão de E para I_A ou I_S: ω
- Taxa de mortalidade devido à COVID-19: σ

- Taxa de recuperação da população humana sintomática (por dia): γ_S
- Taxa de recuperação da população humana assintomática (por dia): γ_A
- Taxa de disseminação do vírus para o ambiente por individuos infecciosos sintomáticos (por dia): η_S
- Taxa de disseminação do vírus para o ambiente por indivíduos infecciosos assintomáticos (por dia): η_A
- Estado inicial da epidemia: $S(0)$, $E(0)$, $I_A(0)$, $I_S(0)$, $R(0)$, $P(0)$

Para os quais serão considerados os seguintes valores (considerados no programa como *default*):

- $b = 2.30137 \times 10^{-5}$
- $\mu = 3.38238 \times 10^{-5}$
- $\mu_P = 0.1724$ [2]
- $\alpha_1 = 0.10$ [2]
- $\alpha_2 = 0.10$ [2]
- $\beta_1 = 0.00414$ [2]
- $\beta_2 = 0.0115$ [2]
- $\delta = 0.7$ [2]
- $\psi = 0.0051$ [2]
- $\omega = 0.09$ [2]
- $\sigma = 0.005$
- $\gamma_S = 0.05$ [2]
- $\gamma_A = 0.0714$ [2]
- $\eta_S = 0.1$ [2]
- $\eta_A = 0.05$ [2]
- Estado inicial da epidemia: $S(0) = 50000$, $E(0) = 500$, $I_A(0) = 30$, $I_S(0) = 20$, $R(0) = 0$, $P(0) = 500$

O código desenvolvido 8 está presente no Anexo A, utilizando o método de Runge-Kutta considerado na alínea anterior, no programa 7 (com passo $h = 1$ dia), para aproximar as soluções das equações diferenciais. São, em primeiro lugar, definidas as equações diferenciais 52. É definido o método de Runge Kutta. É definida uma função auxiliar de leitura, a qual permite ao utilizador sair do programa em qualquer altura, durante a introdução de parâmetros, perante quit. Enquanto o utilizador não desejar terminar o programa (novamente, ao inserir quit), são definidos valores *default* para os parâmetros, são lidos os parâmetros, o estado inicial e o número de dias a considerar. Caso o utilizador deseje considerar os *default*, deve clicar em *enter*; se não, pode introduzir o valor desejado, até sob a forma de expressão. São calculados os valores seguintes para cada grupo, até ao número de dias pedido, e é realizado um gráfico.

Abaixo encontra-se um exemplo de execução do código, considerando estes parâmetros, bem como o gráfico produzido.

```
Matemática Computacional
Simulação COVID-19


(enter para parâmetro default, <quit> para sair)
---- Parâmetros ----
Taxa de natalidade:
2.30137e-05
Taxa de mortalidade:
3.38238e-05
Taxa de mortalidade natural de patógenos no meio ambiente:
0.1724
Proporção de interação com um ambiente infeccioso:
0.1
Proporção de interação com um indivíduo infeccioso:
0.1
Taxa de transmissão de S para E devido ao contacto com P:
0.00414
Taxa de transmissão de S para E devido ao contacto com IA e/ou IS:
0.0115
Proporção de indivíduos infecciosos sintomáticos:
0.7
Taxa de progressão de E para S devido a sistema imunológico robusto:
0.0051
Taxa de progressão de E para IA ou IS:
0.09
Taxa de mortalidade devido à COVID-19:
0.005
Taxa de recuperação da população humana sintomática:
0.05
Taxa de recuperação da população humana assintomática:
0.0714
Taxa de disseminação do vírus para o ambiente por indivíduos infecciosos
↪ sintomáticos:
0.1
Taxa de disseminação do vírus para o ambiente por indivíduos infecciosos
↪ assintomáticos:
0.05

---- Estado inicial da epidemia ----
Grupo suscetível(0):
50000
```

```
Grupo exposto(0):
500
Infecciosos assintomáticos(0):
30
Infecciosos sintomáticos(0):
20
Recuperados(0):
0
Patógenos(0):
500

Número de dias da simulação:
90

Sair? (<quit> para terminar, enter para proceder) quit
```

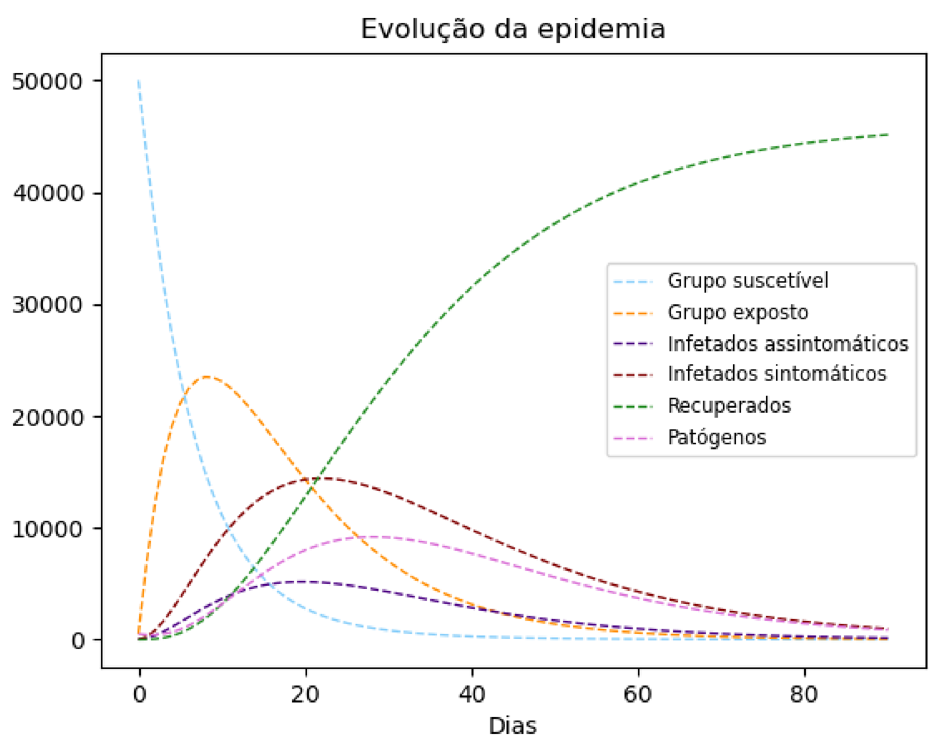

Figura 13: Evolução da epidemia, com os parâmetros por defeito

De modo a ser possível avaliar o efeito dos parâmetros α_1[3] e α_2[4], foi ainda desenvolvido o programa 9, o qual considera quatro combinações destes parâmetros, mantendo os restantes constantes. Tem uma estrutura análoga ao programa anterior 8, sendo a principal diferença o facto de não requerer *input*. Considera apenas os parâmetros *default*, bem como os quatro cenários mencionados. Produz seis gráficos, cinco com os grupos em que se subdivide a população e o sexto referente aos patógenos, sobrepondo em cada um os quatro cenários. São utilizadas as mesmas cores do que no programa 8, para fácil comparação.

Executando o código, obtém-se:

```
Matemática Computacional
Simulação COVID-19

---- Combinações consideradas ----
alpha_1 = 0.100000, alpha_2 = 0.100000
alpha_1 = 0.050000, alpha_2 = 0.100000
alpha_1 = 0.100000, alpha_2 = 0.050000
alpha_1 = 0.050000, alpha_2 = 0.050000
```

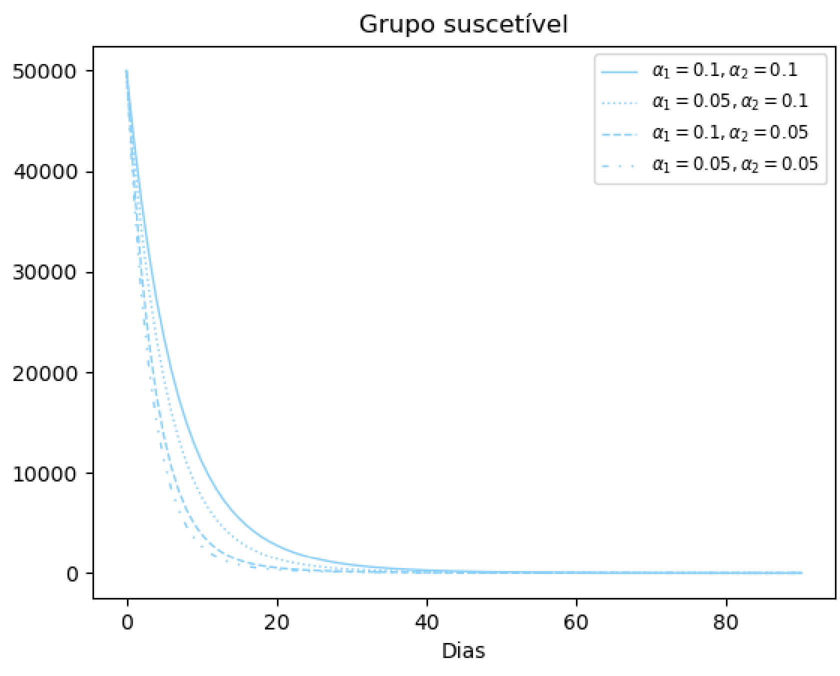

Figura 14: Grupo suscetível

[3]Recíproco da proporção de interação com um ambiente infeccioso

[4]Recíproco da proporção de interação com um indivíduo infeccioso

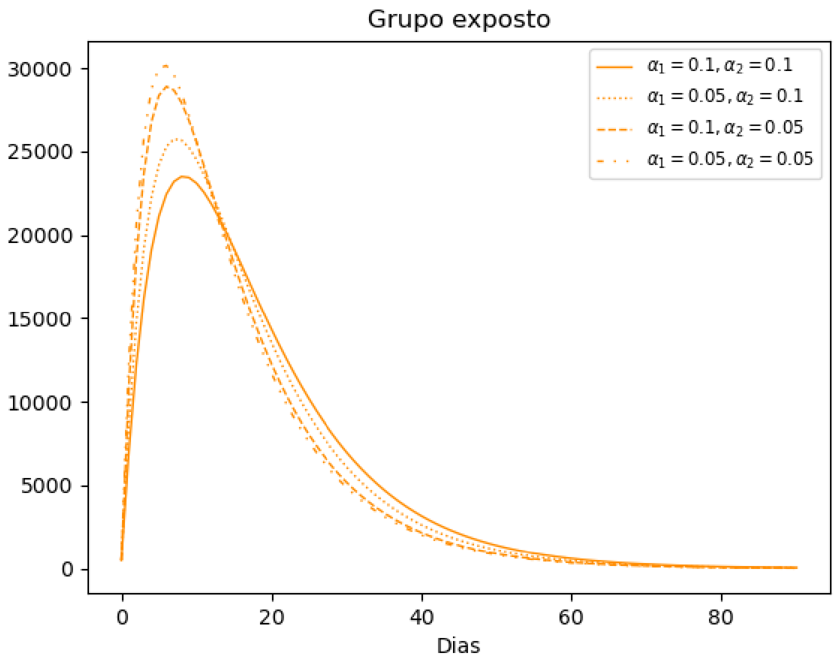

Figura 15: Grupo exposto

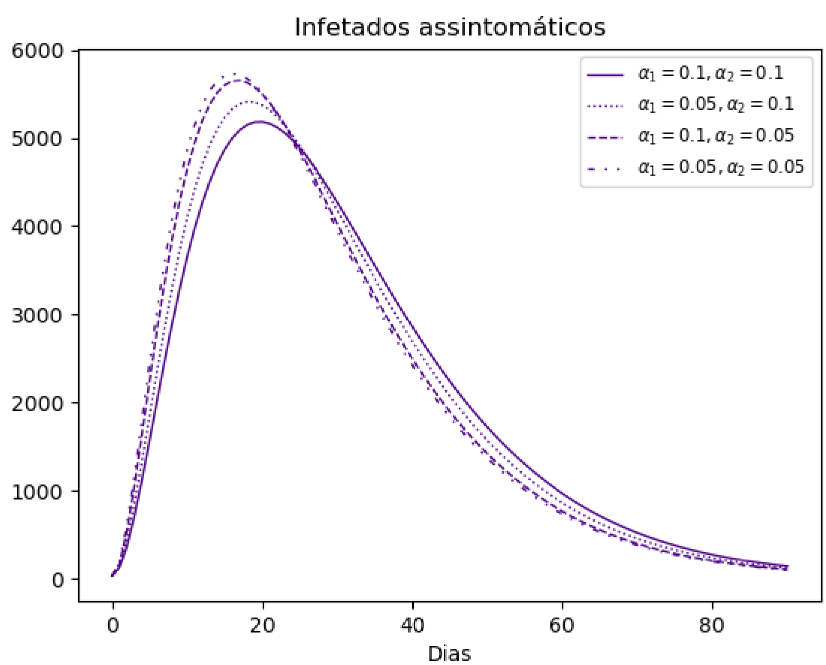

Figura 16: Infetados assintomáticos

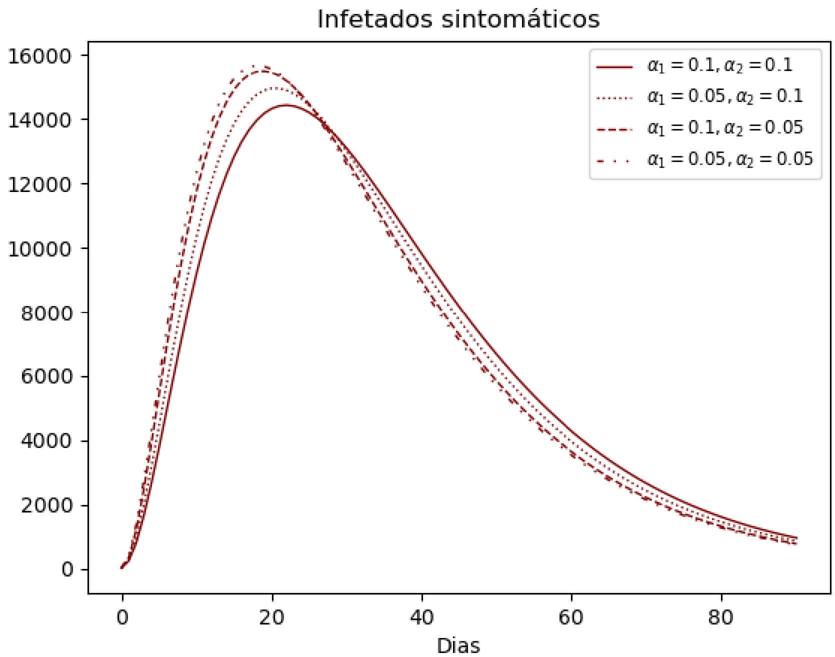

Figura 17: Infetados sintomáticos

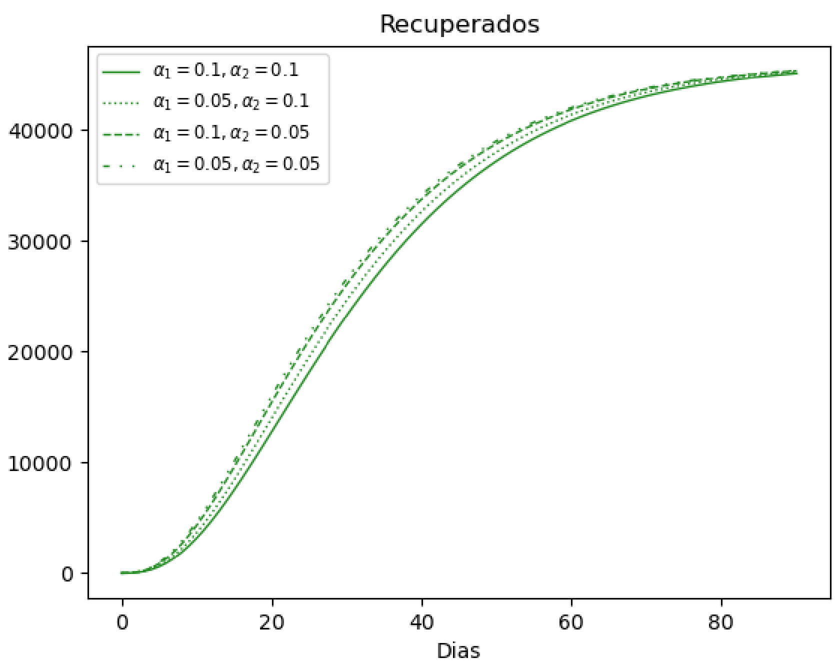

Figura 18: Recuperados

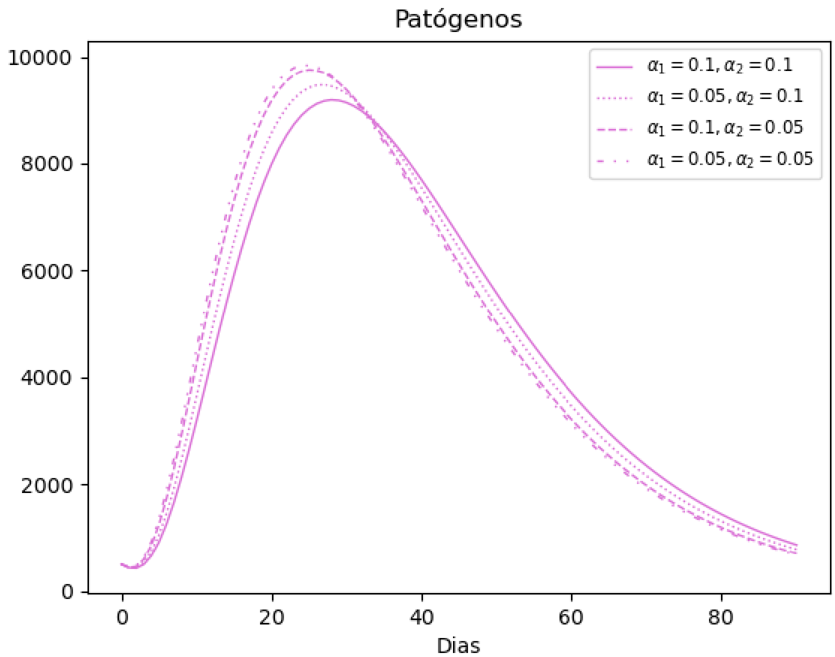

Figura 19: Patógenos

Ao observar as curvas referentes ao grupo exposto (15), infetados assintomáticos (16), infetados sintomáticos (17) e patógenos (19), está patente que o cenário mais perigoso é o último, já que resulta nas curvas que ascendem mais rapidamente, ao longo do tempo, e com picos mais elevados. O *best case scenario* parece ser o primeiro, resultando em curvas mais achatadas, e com uma subida ligeiramente mais suave. Estes resultados estão dentro do esperado, visto o primeiro cenário corresponder ao α_1 e α_2 mais elevados, ou seja, às proporções mais baixas de interação com ambientes e indivíduos infecciosos, respetivamente. O quarto cenário representa exatamente o oposto.

O segundo e terceiro cenários diferenciam-se ainda, se bem de que de forma menos notável, pelos mesmos motivos: o segundo apresenta-se como sendo o menos grave. Este corresponde a elevada proporção de interação com ambientes infecciosos, mas reduzida interação com indivíduos infecciosos, enquanto que o terceiro representa o recíproco. Este resultado é também facilmente justificado quando analisados os parâmetros β_1 e β_2 que caracterizam este vírus: β_2, ou seja a taxa de infeção devido a contacto com infetados, é mais elevado do que β_1, ou seja a taxa de infeção devido a contacto com patógenos.

Confirma-se então a importância de mantar tanto α_1 como α_2 o mais elevados possível, restringido o comportamento da população. Ao ser instituído o isolamento em caso de infeção, α_2 aumenta, dado ser menor o contacto entre indivíduos saudáveis e indivíduos infetados. α_1 aumenta por conseguinte, dado ser contida a dispersão de patógenos por indivíduos infetados. O distanciamento social terá efeitos similares, ao limitar o contacto entre indivíduos infetados e suscetíveis. O dever de recolhimento diminuirá o contacto entre indivíduos suscetíveis e patógenos.

6 Conclusão

Através da realização deste trabalho, pudemos aprofundar não só a nossa compreensão da disciplina, como compreender as metodologias e dificuldades associadas à implementação computacional de vários métodos abordados ao longo do semestre. Permitiu-nos aplicar aquilo que aprendemos num contexto que parecia mais real e não tão estritamente académico.

Neste sentido, foi de caráter particularmente interessante e elucidativo a última questão, relativa à modelação da propagação da epidemia que atravessamos neste momento.

Referências Bibliográficas

[1] *Matplotlib Version 3.3.3*. URL: https://matplotlib.org.

[2] Samuel Mwalili, Mark Kimathi, Viona Ojiambo, Duncan Gathungu e Rachel Mbogo. "SEIR model for COVID-19 dynamics incorporating the environment and social distancing". Em: *BMC Research Notes* 13:352 (2020). URL: https://bmcresnotes.biomedcentral.com/articles/10.1186/s13104-020-05192-1#citeas.

[3] *NumPy v1.19.0*. URL: https://numpy.org.

[4] *SciPy*. URL: https://www.scipy.org/.

Foi também consultado o material de estudo disponibilizado na página da cadeira.

A Código

Programa 2: Quasi_Newton.py

```
# Importação das bibliotecas necessárias
# https://numpy.org/doc/stable/reference/routines.math.html
import numpy as np
import matplotlib.pyplot as plt

# Definição das funções necessárias
def sin(x):
    return np.sin(x)
def cos(x):
    return np.cos(x)
def tan(x):
    return np.tan(x)
def arcsin(x):
    return np.arcsin(x)
def arccos(x):
    return np.arccos(x)
def arctan(x):
    return np.arctan(x)
def sinh(x):
    return np.sinh(x)
def cosh(x):
    return np.cosh(x)
def tanh(x):
    return np.tanh(x)
def arcsinh(x):
    return np.arcsinh(x)
def arccosh(x):
    return np.arccosh(x)
def arctanh(x):
    return np.arctanh(x)
def exp(x):
    return np.exp(x)
def log(x):
    return np.log(x)
def log10(x):
    return np.log10(x)
def sqrt(x):
    return np.sqrt(x)
def cbrt(x):
    return np.cbrt(x)
```

```python
def fabs(x):
    return np.fabs(x)
def power(x, y):
    return np.power(x, y)

# Definição das constantes necessárias
e = np.e
pi = np.pi

# Funções disponíveis
func = ["sin", "cos", "tan"]
func2 = ["exp", "log", "log10", "sqrt", "cbrt", "fabs", "power"]
consts = ["pi", "e "]

print("\n\nMatemática Computacional\n\n")
print("Método Quasi-Newton")

print("A função deve ser inserida em função de t")
print("<man> para opções")
val = input("Insira uma função: f(t) = ")

# Impressão de instruções de utilização, pertante o input "man"
if (val == "man"):
    print("\nLista de funções suportadas:\n")
    for x in func:
        print(x)
    for x in func:
        print("arc" + x)
    for x in func:
        print(x + "h")
    for x in func:
        print("arc" + x + "h")
    for x in func2:
        print(x)
    print("\nLista de constantes suportadas:\n")
    for x in consts:
        print(x)
    exit()

# Teste
# Neste caso a função não pode ser constante (a expressão tem que depender de t)
if ("t" not in val):
    print("Erro, função inválida")
    exit()

```

```
# Input do delta
delta = float(eval(input("Insira o delta: ")))
if delta <= 0:
    print("Erro, delta inválido")
    exit()

# Input do x0
x0 = float(eval(input("Insira a aproximação inicial x0: ")))

# Input do número máximo de iterações (mínimo de 1)
nmax = int(eval(input("Insira o número máximo de iterações a realizar: ")))
if nmax <= 1:
    print("Erro, número máximo de iterações inválido")
    exit()

# Input da tolerância
E = float(eval(input("Insira a tolerância de erro: ")))
if E <= 0:
    print("Erro, tolerância de erro inválida")
    exit()

# Definição do cálculo da função
def fu(t):
    return eval(val)

# Definição do cálculo do método de quasi-Newton
def calc(x):
    return x - delta / (fu(x+delta)-fu(x)) * fu(x)

# Definição do cálculo do majorante do erro, com um algarismo e arredondado para cima
# (Presume que x é positivo)
def err(x):
    num = float("{:e}".format(x).split('e')[0])
    pow = float("{:e}".format(x).split('e')[1])
    num = np.ceil(num)
    if (x >= num*(10**pow)):
        num += 1
        if (num == 10):
            num = 0
            pow += 1
    return num*(10**pow)

# Inicialização do vetor para as iteradas
x = []
# Introdução da iterada 0
```

```python
x.append(x0)
# Introdução da iterada 1
x.append(calc(x0))
it = 1
# Cálculo das iteradas seguintes enquanto
# - O número de iteradas é inferior ao número máximo de iteradas permitido
# - Não é encontrado um zero
# - O erro é superior à tolerância
while ((it < nmax) and (fu(x[it]) != 0) and (abs(x[it]-x[it-1]) > E)):
    x.append(calc(x[it]))
    it = it + 1

# Impressão dos erros (tomando como valor real a última iterada)
# Cálculo da função nos pontos das iteradas
y = []
i = 0
print("\n---- Cálculo das iteradas ----\n")
for n in x:
    if (i != it):
        print("%4d: %16.15e +- %.0e" % (i, n, err(abs(n-x[it]))))
    else:
        print("%4d: %16.15e (f(z) = %.0e)" % (i, n, fu(n)))
    y.append(fu(n))
    i = i + 1

# ---- Gráfico ----
# Cálculo de uma escala para o gráfico
# Cálculo da função no intervalo entre a iterada mais pequena e a iterada maior, em 1000 pontos igualmente espaçados
t = np.arange(np.amin(x), np.amax(x)+(np.amax(x) - np.amin(x))/1000, (np.amax(x) - np.amin(x))/1000)
# Formatação do texto com o número da iterada
i = 0
while (i <= it):
    plt.annotate(str(i), (x[i], y[i]))
    i += 1
# Plot da função
plt.plot(t, fu(t), color='forestgreen', linestyle='dashed', label='f(t)')
# Plot dos pontos das iteradas
plt.plot(x, y, marker='x', color='gold', label='Iteradas ($x_i$, $f(x_i)$)', linewidth=0)
# Definição de legendas
plt.xlabel('$x_i$')
plt.ylabel('$f(t)$')
plt.title("f(t) = " + val)
```

```
plt.legend(loc='best', shadow=False, fontsize='small')
# Desenho do gráfico
plt.show()
```

Programa 3: GrupoII_Ex1b.py

```
# Importação das bibliotecas necessárias
# https://numpy.org/doc/stable/reference/routines.math.html
import numpy as np
import matplotlib.pyplot as plt

# Definição das funções necessárias
def sin(x):
    return np.sin(x)
def cos(x):
    return np.cos(x)
def tan(x):
    return np.tan(x)
def arcsin(x):
    return np.arcsin(x)
def arccos(x):
    return np.arccos(x)
def arctan(x):
    return np.arctan(x)
def sinh(x):
    return np.sinh(x)
def cosh(x):
    return np.cosh(x)
def tanh(x):
    return np.tanh(x)
def arcsinh(x):
    return np.arcsinh(x)
def arccosh(x):
    return np.arccosh(x)
def arctanh(x):
    return np.arctanh(x)
def exp(x):
    return np.exp(x)
def log(x):
    return np.log(x)
def log10(x):
    return np.log10(x)
def sqrt(x):
    return np.sqrt(x)
def cbrt(x):
    return np.cbrt(x)
def fabs(x):
    return np.fabs(x)
def power(x, y):
```

```
    return np.power(x, y)

# Definição das constantes necessárias
e = np.e
pi = np.pi

# Definição das constantes específicas do exercício
a1 = 5/9
a2 = 8/9
a3 = 5/9

print("\n\nMatemática Computacional\n")
print("Quadratura de Gauss Composta para Aproximação de Integrais\n\n")
print("Insira 'man' (em vez de uma função) para ver o manual de funções e constantes suportadas ou 'quit' para terminar o programa.\n")

val = input("Insira uma função (de uma variável t): f(t) = ")

func = ["sin", "cos", "tan"]
func2 = ["exp", "log", "log10", "sqrt", "cbrt", "fabs", "power(base, exponent)"]
consts = ["pi", "e "]

while (val != "quit"):
    # Impressão de instruções de utilização, pertante o input "man"
    if (val == "man"):
        print("\nLista de funções suportadas:\n")
        for x in func:
            print(x)
        for x in func:
            print("arc" + x)
        for x in func:
            print(x + "h")
        for x in func:
            print("arc" + x + "h")
        for x in func2:
            print(x)
        print("\nLista de constantes suportadas:\n")
        for x in consts:
            print(x)
    else:
        # Input do intervalo
        tmin = float(eval(input("Insira o mínimo do intervalo de cálculo do integral: ")))
        tmax = float(eval(input("Insira o máximo do intervalo de cálculo do integral: ")))
```

```
        # Teste
        if (tmin > tmax):
            print("Erro")
            exit()

        # Input do número de passos
        n = int(input("Insira o número de passos a considerar na aproximação do integral: "))
        if (n < 1):
            print("Erro")
            exit()

        # Definição do step size (h)
        h = (tmax-tmin)/n

        # Definição do cálculo da função
        def fu(t):
            return eval(val)

        # Cálculo do nodo utilizado
        def node1(i): return i + h*((1-sqrt(3/5))/2)
        def node3(i): return i + h*((1+sqrt(3/5))/2)
        def node2(i): return i + h/2

        i = tmin
        sum1 = 0
        sum2 = 0
        sum3 = 0

        # Vetores a utilizar caso se queira mostrar as iterações no gráfico
        #y1 = []
        #y2 = []
        #y3 = []
        #x1 = []
        #x2 = []
        #x3 = []
        #x4 = []
        #y4 = []

        while (i < tmax):
            sum1 += fu(node1(i))
            sum2 += fu(node2(i))
            sum3 += fu(node3(i))
            #y1.append(fu(node1(i)))
```

```
            #y2.append(fu(node2(i)))
            #y3.append(fu(node3(i)))
            #y4.append(fu(i))
            #x1.append(i + h*((1-sqrt(3/5))/2))
            #x2.append(i + h/2)
            #x3.append(i + h*((1+sqrt(3/5))/2))
            #x4.append(i)
            i += h

        Q = h/2 * (a1 * sum1 + a2 * sum2 + a3 * sum3)

        print("Q(f) = ", Q)

        # ---- Gráfico ----
        # Cálculo da função entre tmin e tmax
        dt = (tmax-tmin) / 1000
        t = np.arange(tmin, tmax+dt, dt)
        # Cálculo da função
        y = []
        for p in t:
            y.append(fu(p))
        # Plot da função
        plt.plot(t, y, 'r--')
        #plt.plot(x1, y1, 'yo')
        #plt.plot(x2, y2, 'go')
        #plt.plot(x3, y3, 'bo')
        #plt.plot(x4, y4, 'ko')
        # Definição de legendas
        plt.xlabel('$t$')
        plt.ylabel('$f(t)$')
        plt.title("$f(t) = $" + val)
        # Desenho do gráfico
        plt.show()

    print("\n\nInsira 'man' (em vez de uma função) para ver o manual de funções e
    ↪ constantes suportadas ou 'quit' para terminar o programa.\n")
    val = input("Insira uma função (de uma variável t): f(t) = ")
```

Programa 4: GrupoII_Ex2b.py

```
# Importação das bibliotecas necessárias
# https://numpy.org/doc/stable/reference/routines.math.html
import numpy as np
import scipy.integrate
import matplotlib.pyplot as plt

# Definição das funções necessárias
def exp(x):
    return np.exp(x)
def sqrt(x):
    return np.sqrt(x)
def power(x, y):
    return np.power(x, y)

# Definição das constantes específicas do exercício
a1 = 5/9
a2 = 8/9
a3 = 5/9

# Definição da função
def fu(t):
    return sqrt(1+exp(-t/2)*(1-t/4)*(1-t/4))

# Cálculo do nodo utilizado
def node1(i): return i + h*((1-sqrt(3/5))/2)
def node3(i): return i + h*((1+sqrt(3/5))/2)
def node2(i): return i + h/2

print("\n\nMatemática Computacional")
print("Determinar espaço percorrido por corpo em movimento com quadratura de Gauss
↪  composta")

# Valores específicos para estas alíneas
tmin=0
tmax=15

# Função que fornece valor do integral (i[0]) e erro associado(i[1])
i = scipy.integrate.quad(fu, 0, 15)
print ("\nValor verdadeiro do integral: %0.15f" % i[0])
print ("Erro associado: %0.15f" % i[1])

# Número de ciclos
contador = 0
```

```

while (True):

    val = input("\nInsira uma das seguintes opções:\n->'quit' se deseja sair do
    ↪  programa;\n-> n, número de passos inicial (inteiro superior ou igual a 1).\n\n")

    if val.isdigit() :

        # Obtenção do valor de n
        n = int(val)
        if (n < 1):
            print("Erro: O número de passos inicial deve ser um número inteiro superior
            ↪  ou igual a 1\n")
            exit()

        # Obtenção do número de iteradas (linhas da tabela)
        iter = input("\nQuantas linhas da tabela deseja ter? ")
        if (iter.isdigit()==False or int(iter) < 1):
            print("\nErro: Deve digitar um número inteiro igual ou superior a 1.\n")
            exit()
        else:
            iter = int(iter)

        # Guardar o valor inicial de n
        n_copia = n

        # Vetores para valores n e Q, respetivamente
        a=[]
        b=[]

        # Cálculo das várias iteradas
        for cont in range(0, iter):

            # Adicionar valor de n ao vetor a
            a.append(n)

            # Definição do step size (h)
            h = (tmax-tmin)/n

            i = tmin
            sum1 = 0
            sum2 = 0
            sum3 = 0
            while (i < tmax):
                sum1 += fu(node1(i))
```

```
                sum2 += fu(node2(i))
                sum3 += fu(node3(i))
                i += h

            Q = h/2 * (a1 * sum1 + a2 * sum2 + a3 * sum3)

            # Guardar o valor da quadratura no vetor respetivo
            b.append(Q)

            # Alterar o valor de n
            n *= 2

        # Vetores com a terceira e quarta colunas da tabela, respetivamente
        c=[]
        d=[]

        # Primeira linha
        if iter>1:
            col3 = abs(b[1] - b[0])
            c.append(col3)
            d.append(-1)

        # Linhas intermédias
        if iter>2:
            for cont in range (1, iter-1):
                col3 = abs(b[cont+1] - b[cont])
                col4 = col3 / (abs(b[cont] - b[cont-1]))
                c.append(col3)
                d.append(col4)

        # Última linha
        c.append(-1)
        d.append(-1)

        # Escrita da tabela num ficheiro
        if contador==0:
            ficheiro=open(r"ProjetoMC_Grupo2_Exercicio2b.txt", "w")
        ficheiro.write ("Eis a tabela final para número de passos inicial n=%d e %d
        ↪  iteradas:\n" % (n_copia, iter))
        ficheiro.write ("{} {} {} {} {} {} {}".format(" n",11*' ', "Q_n", 14*' ', "|Q_2n
        ↪  - Q_n|", 4*' ', "|(Q_2n - Q_n)/(Q_n-Q_n/2)|"))

        for cont in range(iter):
            ficheiro.write ('\n')
            ficheiro.write ("{} {}".format("%3.0f" % a[cont], '    '))
```

```
            ficheiro.write ("{} {}".format("%.15f" % b[cont], '   '))
            if c[cont]==-1:
                ficheiro.write ("{} {} {} {}".format(9*' ', '-', 9*' ', '    '))
            else:
                ficheiro.write ("{} {}".format("%.15e" % c[cont], '    '))
            if d[cont]==-1:
                ficheiro.write ("{} {} {}".format(9*' ', '-', '    '))
            else:
                ficheiro.write ("{} {}".format("%.15e" % d[cont], '   '))

        ficheiro.write ("\n\n")

        # Gráfico da função integranda
        # Cálculo da função entre tmin e tmax
        dt = (tmax-tmin) / 1000
        t = np.arange(tmin, tmax+dt, dt)
        # Cálculo da função
        y = []
        for p in t:
            y.append(fu(p))
        # Plot da função
        plt.plot(t, y, linestyle='solid')
        # Definição de legendas
        plt.xlabel('t', fontsize=15)
        axes = plt.gca()
        axes.set_xlim([0, 15])
        plt.ylabel('f(t)', fontsize=15)
        plt.title("Função integranda " r'f(t) =
        ↪ $\sqrt{1+e^{-\frac{t}{2}}{\left(1-\frac{t}{4}\right)}^2}$', fontsize=15)
        # Desenho do gráfico
        plt.show()

        # Contabilizar novo ciclo de cálculo realizado
        contador+=1

        print ("\n-=-=-=-=-=-")

    elif val=='quit':
        break

    else:
        print("\nDigitou algo inválido.")

if contador>0:
    # Fechar o ficheiro
```

```
    ficheiro.close()

    if contador==1:
        print ("\nA tabela foi escrita no ficheiro 'ProjetoMC_Grupo2_Exercicio2b.txt'.")
    if contador>1:
        print ("\nAs tabelas foram escritas no ficheiro 'ProjetoMC_Grupo2_Exercicio2b.txt'.")

print('\n')
```

Programa 5: GrupoII_Ex2c.py

```python
# Importação das bibliotecas necessárias
# https://numpy.org/doc/stable/reference/routines.math.html
import numpy as np
import scipy.integrate
import matplotlib.pyplot as plt

# Definição das funções necessárias
def exp(x):
    return np.exp(x)
def sqrt(x):
    return np.sqrt(x)
def power(x, y):
    return np.power(x, y)

# Definição das constantes específicas para a quadratura de Gauss
a1 = 5/9
a2 = 8/9
a3 = 5/9

# Definição da função
def fu(t):
    return sqrt(16+exp(-t/2)*(16+t*(t-8)))/4

# Cálculo dos nodos utilizados para Gauss
def node1(i): return i + h*((1-sqrt(3/5))/2)
def node3(i): return i + h*((1+sqrt(3/5))/2)
def node2(i): return i + h/2

print("\n\nMatemática Computacional")
print("Determinar espaço percorrido por corpo em movimento com diferentes quadraturas")

# Valores específicos para estas alíneas
tmin=0
tmax=15

# Função que fornece valor do integral (TrueValue[0]) e erro associado(TrueValue[1])
TrueValue = scipy.integrate.quad(fu, 0, 15)
print ("\nValor verdadeiro do integral: %0.15f" % TrueValue[0])
print ("Erro associado: %0.15f" % TrueValue[1])

# Número de tabelas obtidas
contador = 0

```

```
while (True):

    val = input("\nInsira uma das seguintes opções:\n->'quit' se deseja sair do programa;\n-> n, número de passos inicial (inteiro superior ou igual a 1).\n\n")

    if val.isdigit() :

        # Obtenção do valor de n
        n = int(val)
        if (n < 1):
            print("Erro: O número de passos inicial deve ser um número inteiro superior ou igual a 1.\n")
            exit()

        # Obtenção do número de iteradas (linhas da tabela)
        iter = input("\nQuantas iteradas deseja realizar? ")
        if (iter.isdigit()==False or int(iter) < 1):
            print("\nErro: Deve digitar um número inteiro igual ou superior a 1.\n")
            exit()
        else:
            iter = int(iter)

        # Guardar o valor inicial de n
        n_copia = n

        # Perguntar quais as regras de quadratura a aplicar
        print("\n->Seleção de regras de quadratura")
        print("Digite 'Enter' para 'Sim', ou qualquer outro input para 'Não'.")
        val1 = input("Deseja aplicar a regra de quadratura de Gauss composta? ")
        if val1 == "":
            Gauss = 1
            b=[]
        else:
            Gauss = 0
        val2 = input("Deseja aplicar a regra dos trapézios composta? ")
        if val2 == "":
            Trapezios = 1
            c=[]
        else:
            Trapezios = 0
        val3 = input("Deseja aplicar a regra de Simpson composta? ")
        if val3 == "":
            Simpson = 1
            d=[]
        else:
```

```
            Simpson = 0
        val4 = input("Deseja aplicar a regra do ponto médio composta? ")
        if val4 == "":
            Retangulos = 1
            e=[]
        else:
            Retangulos = 0

        a=[]

        # Cálculo das várias iteradas
        for cont in range(0, iter):

            # Adicionar valor de n ao vetor a
            a.append(n)

            # Definição do step size (h)
            h = (tmax-tmin)/n

            #Gauss
            if Gauss==1:
                i = tmin
                sum1 = 0
                sum2 = 0
                sum3 = 0
                while (i < tmax):
                    sum1 += fu(node1(i))
                    sum2 += fu(node2(i))
                    sum3 += fu(node3(i))
                    i += h
                Q = h/2 * (a1 * sum1 + a2 * sum2 + a3 * sum3)

                # Guardar o valor da quadratura no vetor respetivo
                b.append(Q)

            #Trapézios
            if Trapezios==1:
                sum1 = 0
                i = 0
                while i<n:
                    sum1 += fu(tmin+i*h) + fu(tmin+(i+1)*h)
                    i += 1
                sum1 *= (h/2)
                c.append(sum1)

```

```
            #Simpson
            if Simpson==1:
                sum1 = 0
                sum1 += fu(tmin) + fu(tmax)
                i = 1
                sum2 = 0
                while i<=n/2:
                    sum2 += fu(tmin + (2*i-1)*h)
                    i += 1
                sum2 *= 4
                i = 1
                sum3 = 0
                while i<=n/2-1:
                    sum3 += fu(tmin + 2*i*h)
                    i += 1
                sum3 *= 2
                sum1 += sum2 + sum3
                sum1 *= (h/3)
                d.append(sum1)

            #Ponto médio/retângulos
            if Retangulos==1:
                sum1 = 0
                i = 0
                while i<n:
                    sum1 += fu(tmin + h/2 + i*h)
                    i += 1
                sum1 *= h
                e.append(sum1)

            # Alterar o valor de n
            n *= 2

        # Tabela
        if contador==0:
            ficheiro=open(r"ProjetoMC_Grupo2_Exercicio2c.txt", "w")

        if Gauss==1:
            # Criar vetores com as linhas da tabela
            Gauss_col3=[]
            Gauss_col4=[]

            # Calcular elementos da terceira e quarta colunas
            if iter>1:
                col3 = abs(b[1] - b[0])
```

```
                Gauss_col3.append(col3)
                Gauss_col4.append(-1)
            if iter>2:
                for cont in range (1, iter-1):
                    col3 = abs(b[cont+1] - b[cont])
                    col4 = col3 / (abs(b[cont] - b[cont-1]))
                    Gauss_col3.append(col3)
                    Gauss_col4.append(col4)

            Gauss_col3.append(-1)
            Gauss_col4.append(-1)

            # Print da tabela
            ficheiro.write ("\nEis a tabela final para a quadratura de Gauss composta, número de passos inicial n=%d e %d iterações:\n" % (n_copia, iter))
            ficheiro.write ("{} {} {} {} {} {} {} {}".format (4*' ', "n", 11*' ', "Q_n", 14*' ', "|Q_2n - Q_n|", 4*' ', "|(Q_2n - Q_n)/(Q_n-Q_n/2)|"))
            for cont in range(iter):
                ficheiro.write ('\n')
                ficheiro.write ("{} {}".format("%7.0f" % a[cont], '   '))
                ficheiro.write ("{} {}".format("%.15f" % b[cont], '   '))
                if Gauss_col3[cont]==-1:
                    ficheiro.write ("{} {} {} {}".format(9*' ', '-', 9*' ', '    '))
                else:
                    ficheiro.write ("{} {}".format("%.15e" % Gauss_col3[cont], '    '))
                if Gauss_col4[cont]==-1:
                    ficheiro.write ("{} {} {}".format(9*' ', '-', '    '))
                else:
                    ficheiro.write ("{} {}".format("%.15e" % Gauss_col4[cont], '   '))
            ficheiro.write ('\n')
            contador+=1

        if Trapezios==1:
            # Criar vetor com as linhas da tabela
            Trap_col3=[]
            Trap_col4=[]

            # Calcular elementos das terceira e quarta colunas
            if iter>1:
                col3 = abs(c[1] - c[0])
                Trap_col3.append(col3)
                Trap_col4.append(-1)
            if iter>2:
                for cont in range (1, iter-1):
                    col3 = abs(c[cont+1] - c[cont])
```

```
                    col4 = col3 / (abs(c[cont] - c[cont-1]))
                    Trap_col3.append(col3)
                    Trap_col4.append(col4)

            Trap_col3.append(-1)
            Trap_col4.append(-1)

            # Print da tabela
            ficheiro.write ("\nEis a tabela final para a regra dos trapézios composta, número de passos inicial n=%d e %d iterações:\n" % (n_copia, iter))
            ficheiro.write ("{} {} {} {} {} {} {} {}".format (4*' ', "n", 11*' ', "Q_n", 14*' ', "|Q_2n - Q_n|", 4*' ', "|(Q_2n - Q_n)/(Q_n-Q_n/2)|"))
            for cont in range(iter):
                ficheiro.write ('\n')
                ficheiro.write ("{} {}".format("%7.0f" % a[cont], '   '))
                ficheiro.write ("{} {}".format("%.15f" % c[cont], '   '))
                if Trap_col3[cont]==-1:
                    ficheiro.write ("{} {} {} {}".format(9*' ', '-', 9*' ', '    '))
                else:
                    ficheiro.write ("{} {}".format("%.15e" % Trap_col3[cont], '    '))
                if Trap_col4[cont]==-1:
                    ficheiro.write ("{} {} {}".format(9*' ', '-', '    '))
                else:
                    ficheiro.write ("{} {}".format("%.15e" % Trap_col4[cont], '   '))
            ficheiro.write ('\n')
            contador+=1

        if Simpson==1:
            # Criar vetor com as linhas da tabela
            Simp_col3=[]
            Simp_col4=[]

            # Calcular elementos das terceira e quarta colunas
            if iter>1:
                col3 = abs(d[1] - d[0])
                Simp_col3.append(col3)
                Simp_col4.append(-1)
            if iter>2:
                for cont in range (1, iter-1):
                    col3 = abs(d[cont+1] - d[cont])
                    col4 = col3 / (abs(d[cont] - d[cont-1]))
                    Simp_col3.append(col3)
                    Simp_col4.append(col4)

            Simp_col3.append(-1)
```

```
            Simp_col4.append(-1)

            # Print da tabela
            ficheiro.write ("\nEis a tabela final para a regra de Simpson composta, número de passos inicial n=%d e %d iterações:\n" % (n_copia, iter))
            ficheiro.write ("{} {} {} {} {} {} {} {}".format (4*' ', "n", 11*' ', "Q_n", 14*' ', "|Q_2n - Q_n|", 4*' ', "|(Q_2n - Q_n)/(Q_n-Q_n/2)|"))
            for cont in range(iter):
                ficheiro.write ('\n')
                ficheiro.write ("{} {}".format("%7.0f" % a[cont], '   '))
                ficheiro.write ("{} {}".format("%.15f" % d[cont], '   '))
                if Simp_col3[cont]==-1:
                    ficheiro.write ("{} {} {} {}".format(9*' ', '-', 9*' ', '      '))
                else:
                    ficheiro.write ("{} {}".format("%.15e" % Simp_col3[cont], '    '))
                if Simp_col4[cont]==-1:
                    ficheiro.write ("{} {} {}".format(9*' ', '-', '    '))
                else:
                    ficheiro.write ("{} {}".format("%.15e" % Simp_col4[cont], '   '))
            ficheiro.write ('\n')
            contador+=1

        if Retangulos==1:
            # Criar vetor com as linhas da tabela
            Ret_col3=[]
            Ret_col4=[]

            # Calcular elementos das terceira e quarta colunas
            if iter>1:
                col3 = abs(e[1] - e[0])
                Ret_col3.append(col3)
                Ret_col4.append(-1)
            if iter>2:
                for cont in range (1, iter-1):
                    col3 = abs(e[cont+1] - e[cont])
                    col4 = col3 / (abs(e[cont] - e[cont-1]))
                    Ret_col3.append(col3)
                    Ret_col4.append(col4)

            Ret_col3.append(-1)
            Ret_col4.append(-1)

            # Print da tabela
            ficheiro.write ("\nEis a tabela final para a regra dos retângulos, número de passos inicial n=%d e %d iterações:\n" % (n_copia, iter))
```

```python
            ficheiro.write ("{} {} {} {} {} {} {} {}".format (4*' ', "n", 11*' ', "Q_n", 14*' ', "|Q_2n - Q_n|", 4*' ', "|(Q_2n - Q_n)/(Q_n-Q_n/2)|"))

            for cont in range(iter):
                ficheiro.write ('\n')
                ficheiro.write ("{} {}".format("%7.0f" % a[cont], '   '))
                ficheiro.write ("{} {}".format("%.15f" % e[cont], '   '))
                if Ret_col3[cont]==-1:
                    ficheiro.write ("{} {} {} {}".format(9*' ', '-', 9*' ', '    '))
                else:
                    ficheiro.write ("{} {}".format("%.15e" % Ret_col3[cont], '    '))
                if Ret_col4[cont]==-1:
                    ficheiro.write ("{} {} {}".format(9*' ', '-', '    '))
                else:
                    ficheiro.write ("{} {}".format("%.15e" % Ret_col4[cont], '   '))
            ficheiro.write ('\n')
            contador+=1

        # Gráficos
        grafico = input("\nDigite:\n->'i' se desejar obter gráficos das sucessivas iteradas;\n->'e' se desejar obter gráficos dos módulos dos respetivos erros;\n-> algo sem 'i' nem 'e', se não desejar obter qualquer gráfico.\n\n")

        if 'i' in grafico:
            plt.figure(0)
            # Gráfico(s) das sucessivas iteradas
            # Plot da linha horizontal com o valor verdadeiro
            vec1 = [n_copia-5, n_copia*pow(2,iter-1)+5]
            vec2 = [TrueValue[0], TrueValue[0]]
            axes = plt.gca()
            axes.set_xlim([n_copia-5, n_copia*pow(2,iter-1)+5])
            plt.xticks(a)
            plt.plot(vec1, vec2, color='black', linestyle='solid', linewidth=0.5, label="Valor verdadeiro")

            # Plot dos pontos das diferentes quadraturas
            if Gauss==1:
                plt.plot(a, b, color='lightskyblue', linestyle='dashed', marker='o', linewidth=1, label='Gauss')
            if Trapezios==1:
                plt.plot(a, c, color='darkorange', linestyle='dashed', marker='v', linewidth=1, label='Trapézios')
            if Simpson==1:
```

```
                plt.plot(a, d, color='indigo', linestyle='dashed', marker='s', linewidth=1, label='Simpson')
            if Retangulos==1:
                plt.plot(a, e, color='maroon', linestyle='dashed', marker='*', linewidth=1, label='Retângulos')

            plt.xlabel("n", fontsize=15)
            plt.ylabel("$Q_{n}(f)$", fontsize=15)
            plt.title("Convergência para diferentes regras de quadratura compostas", fontsize=15)
            plt.legend(loc='best', shadow=False, fontsize='large')
            if 'e' not in grafico:
                plt.show()

        if 'e' in grafico:
            plt.figure(1)
            # Gráfico dos módulos dos erros das sucessivas iteradas
            # Plot da linha horizontal com o valor verdadeiro
            vec1=[n_copia-5, n_copia*pow(2,iter-1)+5]
            vec2=[0, 0]
            axes = plt.gca()
            axes.set_xlim([n_copia-5, n_copia*pow(2,iter-1)+5])
            plt.xticks(a)
            plt.plot(vec1, vec2, color='black', linestyle='solid', linewidth=0.5, label="Valor verdadeiro")

            # Plot dos pontos das diferentes quadraturas
            if Gauss==1:
                b_copia = []
                for j in b:
                    b_copia.append(abs(TrueValue[0]-j))
                plt.plot(a, b_copia, color='lightskyblue', linestyle='dashed', marker='o', linewidth=1, label='Gauss')
            if Trapezios==1:
                c_copia=[]
                for j in c:
                    c_copia.append(abs(TrueValue[0]-j))
                plt.plot(a, c_copia, color='darkorange', linestyle='dashed', marker='v', linewidth=1, label='Trapézios')
            if Simpson==1:
                d_copia=[]
                for j in d:
                    d_copia.append(abs(TrueValue[0]-j))
                plt.plot(a, d_copia, color='indigo', linestyle='dashed', marker='s', linewidth=1, label='Simpson')
```

```
            if Retangulos==1:
                e_copia=[]
                for j in e:
                    e_copia.append(abs(TrueValue[0]-j))
                plt.plot(a, e_copia, color='maroon', linestyle='dashed', marker='*', linewidth=1, label='Retângulos')

            plt.xlabel("n", fontsize=15)
            plt.ylabel("$|L(15)-Q_{n}(f)|$", fontsize=15)
            plt.title("Erros associados às diferentes fórmulas de quadratura compostas", fontsize=15)
            plt.legend(loc='best', shadow=False, fontsize='large')
            plt.show()

        print ("\n-=-=-=-=-=-")

    elif val=='quit':
        break

    else:
        print("\nDigitou algo inválido.")

if contador>0:
    # Fechar o ficheiro
    ficheiro.close()

    if contador==1:
        print ("\nA tabela foi escrita no ficheiro 'ProjetoMC_Grupo2_Exercicio2c.txt'.")
    if contador>1:
        print ("\nAs tabelas foram escritas no ficheiro 'ProjetoMC_Grupo2_Exercicio2c.txt'.")

print('\n')
```

Programa 6: GrupoII_Ex2d.py

```
# Importação das bibliotecas necessárias
# https://numpy.org/doc/stable/reference/routines.math.html
import numpy as np
import matplotlib.pyplot as plt
import math

# Definição das funções necessárias
def sin(x):
    return np.sin(x)
def cos(x):
    return np.cos(x)
def tan(x):
    return np.tan(x)
def arcsin(x):
    return np.arcsin(x)
def arccos(x):
    return np.arccos(x)
def arctan(x):
    return np.arctan(x)
def sinh(x):
    return np.sinh(x)
def cosh(x):
    return np.cosh(x)
def tanh(x):
    return np.tanh(x)
def arcsinh(x):
    return np.arcsinh(x)
def arccosh(x):
    return np.arccosh(x)
def arctanh(x):
    return np.arctanh(x)
def exp(x):
    return np.exp(x)
def log(x):
    return np.log(x)
def log10(x):
    return np.log10(x)
def sqrt(x):
    return np.sqrt(x)
def cbrt(x):
    return np.cbrt(x)
def fabs(x):
    return np.fabs(x)
```

```python
def power(x, y):
    return np.power(x, y)

# Definição das constantes necessárias
e = np.e
pi = np.pi

# Definição das constantes específicas do exercício
a1 = 5/9
a2 = 8/9
a3 = 5/9

#print("\n\nMatemática Computacional\n")
#print("Grupo II Exercício 2.d)\n\n")

# Função do exercício 2 do grupo II
val = "sqrt(1+exp(-t/2)*pow(1-t/4,2))"

# Definição do cálculo da função
def fu(t):
    return eval(val)

# O extremo mínimo de integração é sempre 0
tmin = 0

# Array para os integrais
Q = []

# Array para os extremos máximos de integração
t = []

# Para tmax=0=tmin, o integral é 0
Q.append(0)
t.append(0)

for x in range(1,101):
    # Extremo máximo de integração para cada integral
    tmax = x*0.15
    t.append(tmax)

    # Definição do step size (h) com n=1
    n = 1
    h = tmax/n

    # Cálculo do nodo utilizado
```

```
def node1(i): return i + h*((1-sqrt(3/5))/2)
def node3(i): return i + h*((1+sqrt(3/5))/2)
def node2(i): return i + h/2

# Q1=Q_n Q2=Q_2n
Q1 = 0
Q2 = 1

# Cálculo de Q_n e de Q_2n até |Q_n-Q_2n| <= 10^-6
while (math.fabs(Q1-Q2) > 0.000001):
    i = tmin
    sum1 = 0
    sum2 = 0
    sum3 = 0
    while (i < tmax):
        sum1 += fu(node1(i))
        sum2 += fu(node2(i))
        sum3 += fu(node3(i))
        i += h

    Q1 = h/2 * (a1 * sum1 + a2 * sum2 + a3 * sum3)

    n *= 2
    h = tmax/n

    i = tmin
    sum1 = 0
    sum2 = 0
    sum3 = 0
    while (i < tmax):
        sum1 += fu(node1(i))
        sum2 += fu(node2(i))
        sum3 += fu(node3(i))
        i += h

    Q2 = h/2 * (a1 * sum1 + a2 * sum2 + a3 * sum3)

    n /= 2
    n += 1
    h = tmax/n

Q.append(Q1)
#print(n-1)
#print(Q1)
```

```
# ---- Gráfico ----
# Plot dos pontos
plt.plot(t, Q, linewidth=0, marker='o', markeredgecolor='lime',markerfacecolor='lime',
↪  markersize=1)
# Definição do título e dos eixos
plt.xlabel('Extremo máximo de integração - \u03C4')
plt.ylabel('L(\u03C4)')
plt.title(r'$f(t) = \sqrt{1+e^{-\frac{t}{2}}{\left(1-\frac{t}{4}\right)}^2}$')
# Desenho do gráfico
plt.show()
```

Programa 7: Runge_Kutta.py

```
# Importação das bibliotecas necessárias
# https://numpy.org/doc/stable/reference/routines.math.html
import numpy as np
import matplotlib.pyplot as plt

# Definição das funções necessárias
def sin(x):
    return np.sin(x)
def cos(x):
    return np.cos(x)
def tan(x):
    return np.tan(x)
def arcsin(x):
    return np.arcsin(x)
def arccos(x):
    return np.arccos(x)
def arctan(x):
    return np.arctan(x)
def sinh(x):
    return np.sinh(x)
def cosh(x):
    return np.cosh(x)
def tanh(x):
    return np.tanh(x)
def arcsinh(x):
    return np.arcsinh(x)
def arccosh(x):
    return np.arccosh(x)
def arctanh(x):
    return np.arctanh(x)
def exp(x):
    return np.exp(x)
def log(x):
    return np.log(x)
def log10(x):
    return np.log10(x)
def sqrt(x):
    return np.sqrt(x)
def cbrt(x):
    return np.cbrt(x)
def fabs(x):
    return np.fabs(x)
def power(x, y):
```

```
    return np.power(x, y)

# Definição das constantes necessárias
e = np.e
pi = np.pi

# Funções disponíveis
func = ["sin", "cos", "tan"]
func2 = ["exp", "log", "log10", "sqrt", "cbrt", "fabs", "power"]
consts = ["pi", "e "]

# Definição do método de Runge-Kutta
def RK(h, f, t0, y0):
    k1 = f(t0, y0)
    k2 = f(t0+h/3, y0+h/3*k1)
    k3 = f(t0+2*h/3, y0+2*h/3*k2)
    return y0 + h/4*(k1+3*k3)

print("\n\nMatemática Computacional")
print("Método Runge-Kutta")
while (True):
    print("\n\nA função deve ser inserida em função de t e de y")
    print("<man> para opções ou <quit> para sair")
    val = input("Insira uma função: f(t, y) = ")

    # Impressão de instruções de utilização, pertante o input "man"
    if (val == "man"):
        print("\nLista de funções suportadas:\n")
        for x in func:
            print(x)
        for x in func:
            print("arc" + x)
        for x in func:
            print(x + "h")
        for x in func:
            print("arc" + x + "h")
        for x in func2:
            print(x)
        print("\nLista de constantes suportadas:\n")
        for x in consts:
            print(x)
        continue
    elif (val == "quit"):
        break

```

```
    # Input do intervalo
    tmin = float(eval(input("Insira o t mínimo: ")))
    tmax = float(eval(input("Insira o t máximo: ")))
    # Teste
    if (tmin > tmax):
        print("Erro")
        exit()

    # Input do valor da função em tmin
    y0 = float(eval(input("Insira o valor inicial da função: ")))

    # Input do valor de n desejado
    n = float(eval(input("Insira o n pretendido: ")))
    # Cálculo de h
    h0 = (tmax-tmin) / n

    # Definição do cálculo da função
    def fu(t, y):
        return eval(val)

    # ---- Gráfico ----
    # Cálculo da função entre tmin e tmax
    t = np.arange(tmin, tmax+h0, h0)
    y = []
    y.append(y0)
    print("\n---- Aproximações (t, y) ----\n")
    print("%10.6g , %10.6g" % (tmin, y0))
    it = 1
    while (it < len(t)):
        yt = RK(h0, fu, t[it-1], y[it-1])
        print("%10.6g , %10.6g" % (t[it], yt))
        y.append(yt)
        it += 1
    # Plot da função
    plt.plot(t, y, color='darkorange', linestyle='dashed', linewidth=1, label='Spline linear')
    plt.plot(t, y, linewidth=0, marker='o', markeredgecolor='lightskyblue', markerfacecolor='lightskyblue', markersize=2, label='Pontos de cálculo')
    # Definição de legendas
    plt.xlabel('$t$')
    plt.ylabel('$y$')
    plt.title("f(t, y) = " + val)
    plt.legend(loc='best', shadow=False, fontsize='small')
    # Desenho do gráfico
    plt.show()
```

Programa 8: SEIRP.py

```python
# Importação das bibliotecas necessárias
# https://numpy.org/doc/stable/reference/routines.math.html
import numpy as np
import matplotlib.pyplot as plt

print("\n\nMatemática Computacional")
print("Simulação COVID-19")

# Definição do sistema de equações diferenciais
def Sder(S, E, IA, IS, P):
    return b - (beta1*S*P)/(1+alpha1*P) - (beta2*S*(IA+IS))/(1+alpha2*(IA+IS)) + psi*E - mu*S
def Eder(S, E, IA, IS, P):
    return (beta1*S*P)/(1+alpha1*P) + (beta2*S*(IA+IS))/(1+alpha2*(IA+IS)) - (psi + mu + omega)*E
def IAder(E, IA):
    return (1-delta)*omega*E - (mu+sigma + gammaA)*IA
def ISder(E, IS):
    return delta*omega*E - (mu+sigma + gammaS)*IS
def Rder(IA, IS, R):
    return gammaA*IA + gammaS*IS - mu*R
def Pder(IA, IS, P):
    return etaA*IA + etaS*IS - muP*P
def f(S, E, IA, IS, R, P):
    return [Sder(S, E, IA, IS, P), Eder(S, E, IA, IS, P), IAder(E, IA), ISder(E, IS), Rder(IA, IS, R), Pder(IA, IS, P)]

# Definição do método de Runge Kutta
def RK(h, S_0, E_0, IA_0, IS_0, R_0, P_0):
    k1 = f(S_0, E_0, IA_0, IS_0, R_0, P_0)
    k2 = f(S_0+h/3*k1[0], E_0+h/3*k1[1], IA_0+h/3*k1[2], IS_0+h/3*k1[3], R_0+h/3*k1[4], P_0+h/3*k1[5])
    k3 = f(S_0+2*h/3*k2[0], E_0+2*h/3*k2[1], IA_0+2*h/3*k2[2], IS_0+2*h/3*k2[3], R_0+2*h/3*k2[4], P_0+2*h/3*k2[5])
    return [S_0+h/4*(k1[0]+3*k3[0]), E_0+h/4*(k1[1]+3*k3[1]), IA_0+h/4*(k1[2]+3*k3[2]), IS_0+h/4*(k1[3]+3*k3[3]), R_0+h/4*(k1[4]+3*k3[4]), P_0+h/4*(k1[5]+3*k3[5])]

# Função auxiliar de leitura
def read(var, string):
    inp = input(string)
    if (inp == "quit"):
        exit()
    elif (inp != ''):
```

```python
        return float(eval(inp))
    else:
        print(var)
        return var

while (True):
    print("\n\n(enter para parâmetro default, <quit> para sair)")

    # Defingão dos valores default para os parâmetros
    b = 2.30137e-5
    mu = 3.38238e-5
    muP = 0.1724
    alpha1 = 0.1
    alpha2 = 0.1
    beta1 = 0.00414
    beta2 = 0.0115
    delta = 0.7
    psi = 0.0051
    omega = 0.09
    sigma = 0.005
    gammaS = 0.05
    gammaA = 0.0714
    etaS = 0.1
    etaA = 0.05
    S0 = 50000
    E0 = 500
    IA0 = 30
    IS0 = 20
    R0 = 0
    P0 = 500
    days = 90

    # Leitura dos parâmetros
    print("---- Parâmetros ----")
    b = read(b, "Taxa de natalidade: ")
    mu = read(mu, "Taxa de mortalidade: ")
    muP = read(muP, "Taxa de mortalidade natural de patógenos no meio ambiente: ")
    alpha1 = read(alpha1, "Proporção de interação com um ambiente infeccioso: ")
    alpha2 = read(alpha2, "Proporção de interação com um indivíduo infeccioso: ")
    beta1 = read(beta1, "Taxa de transmissão de S para E devido ao contacto com P: ")
    beta2 = read(beta2, "Taxa de transmissão de S para E devido ao contacto com IA e/ou IS: ")
    delta = read(delta, "Proporção de indivíduos infecciosos sintomáticos: ")
    psi = read(psi, "Taxa de progressão de E para S devido a sistema imunológico robusto: ")
```

```
    omega = read(omega, "Taxa de progressão de E para IA ou IS: ")
    sigma = read(sigma, "Taxa de mortalidade devido à COVID-19: ")
    gammaS = read(gammaS, "Taxa de recuperação da população humana sintomática: ")
    gammaA = read(gammaA, "Taxa de recuperação da população humana assintomática: ")
    etaS = read(etaS, "Taxa de disseminação do vírus para o ambiente por indivíduos infecciosos sintomáticos: ")
    etaA = read(etaA, "Taxa de disseminação do vírus para o ambiente por indivíduos infecciosos assintomáticos: ")
    # Leitura do estado inicial
    print("\n---- Estado inicial da epidemia ----")
    S0 = read(S0, "Grupo suscetível(0): ")
    E0 = read(E0, "Grupo exposto(0): ")
    IA0 = read(IA0, "Infecciosos assintomáticos(0): ")
    IS0 = read(IS0, "Infecciosos sintomáticos(0): ")
    R0 = read(R0, "Recuperados(0): ")
    P0 = read(P0, "Patógenos(0): ")
    # Leitura do número de dias a avaliar
    days = read(days, "\nNúmero de dias da simulação: ")

    # Definição dos arrays onde serão armazenados os valores
    t = np.arange(0, days+1, 1)
    Svals = []
    Evals = []
    IAvals = []
    ISvals = []
    Rvals = []
    Pvals = []

    # Armazenamento dos valores iniciais
    Svals.append(S0)
    Evals.append(E0)
    IAvals.append(IA0)
    ISvals.append(IS0)
    Rvals.append(R0)
    Pvals.append(P0)

    # Cálculo dos valores seguintes
    i = 1
    while (i <= days):
        vals = RK(1, Svals[i-1], Evals[i-1], IAvals[i-1], ISvals[i-1], Rvals[i-1], Pvals[i-1])
        Svals.append(vals[0])
        Evals.append(vals[1])
        IAvals.append(vals[2])
        ISvals.append(vals[3])
```

```
        Rvals.append(vals[4])
        Pvals.append(vals[5])
        i += 1

    # Gráfico
    plt.plot(t, Svals, color='lightskyblue', linestyle='dashed', linewidth=1, label='Grupo suscetível')
    plt.plot(t, Evals, color='darkorange', linestyle='dashed', linewidth=1, label='Grupo exposto')
    plt.plot(t, IAvals, color='indigo', linestyle='dashed', linewidth=1, label='Infetados assintomáticos')
    plt.plot(t, ISvals, color='maroon', linestyle='dashed', linewidth=1, label='Infetados sintomáticos')
    plt.plot(t, Rvals, color='forestgreen', linestyle='dashed', linewidth=1, label='Recuperados')
    plt.plot(t, Pvals, color='orchid', linestyle='dashed', linewidth=1, label='Patógenos')
    plt.xlabel("Dias")
    plt.title("Evolução da epidemia")
    plt.legend(loc='best', shadow=False, fontsize='small')
    plt.show()
    val = input("\nSair? (<quit> para terminar, enter para proceder) ")
    if (val == "quit"):
        break
    elif (val == ''):
        continue
    else:
        print("Erro")
        break
```

Programa 9: SEIRP2.py

```python
# Importação das bibliotecas necessárias
# https://numpy.org/doc/stable/reference/routines.math.html
import numpy as np
import matplotlib.pyplot as plt

# Estilo de linha
dashdotdotted = (0, (3, 5, 1, 5, 1, 5))

print("\n\nMatemática Computacional")
print("Simulação COVID-19")

# Definição do sistema de equações diferenciais
def Sder(S, E, IA, IS, P):
    return b - (beta1*S*P)/(1+alpha1*P) - (beta2*S*(IA+IS))/(1+alpha2*(IA+IS)) + psi*E -
    ↪ mu*S
def Eder(S, E, IA, IS, P):
    return (beta1*S*P)/(1+alpha1*P) + (beta2*S*(IA+IS))/(1+alpha2*(IA+IS)) - (psi + mu +
    ↪ omega)*E
def IAder(E, IA):
    return (1-delta)*omega*E - (mu+sigma + gammaA)*IA
def ISder(E, IS):
    return delta*omega*E - (mu+sigma + gammaS)*IS
def Rder(IA, IS, R):
    return gammaA*IA + gammaS*IS - mu*R
def Pder(IA, IS, P):
    return etaA*IA + etaS*IS - muP*P
def f(S, E, IA, IS, R, P):
    return [Sder(S, E, IA, IS, P), Eder(S, E, IA, IS, P), IAder(E, IA), ISder(E, IS),
    ↪ Rder(IA, IS, R), Pder(IA, IS, P)]

# Definição do método de Runge-Kutta
def RK(h, S_0, E_0, IA_0, IS_0, R_0, P_0):
    k1 = f(S_0, E_0, IA_0, IS_0, R_0, P_0)
    k2 = f(S_0+h/3*k1[0], E_0+h/3*k1[1], IA_0+h/3*k1[2], IS_0+h/3*k1[3], R_0+h/3*k1[4],
    ↪ P_0+h/3*k1[5])
    k3 = f(S_0+2*h/3*k2[0], E_0+2*h/3*k2[1], IA_0+2*h/3*k2[2], IS_0+2*h/3*k2[3],
    ↪ R_0+2*h/3*k2[4], P_0+2*h/3*k2[5])
    return [S_0+h/4*(k1[0]+3*k3[0]), E_0+h/4*(k1[1]+3*k3[1]), IA_0+h/4*(k1[2]+3*k3[2]),
    ↪ IS_0+h/4*(k1[3]+3*k3[3]), R_0+h/4*(k1[4]+3*k3[4]), P_0+h/4*(k1[5]+3*k3[5])]

# Função auxiliar de leitura
def read(var, string):
    inp = input(string)
```

```
    if (inp == "quit"):
        exit()
    elif (inp != ''):
        return float(eval(inp))
    else:
        print(var)
        return var

# Defincção dos valores default para os parâmetros
b = 2.30137e-5
mu = 3.38238e-5
muP = 0.1724
beta1 = 0.00414
beta2 = 0.0115
delta = 0.7
psi = 0.0051
omega = 0.09
sigma = 0.005
gammaS = 0.05
gammaA = 0.0714
etaS = 0.1
etaA = 0.05
S0 = 50000
E0 = 500
IA0 = 30
IS0 = 20
R0 = 0
P0 = 500
days = 90

# Definição dos pares alpha1 e alpha2 (e outros atributos)
par = [[0.1, 0.1, 0, 'solid'], [0.05, 0.1, 1, 'dotted'], [0.1, 0.05, 2, 'dashed'],
↪  [0.05, 0.05, 3, dashdotdotted]]

# Definição dos arrays onde serão armazenados os dados
dataS = []
dataE = []
dataIA = []
dataIS = []
dataR = []
dataP = []

for p in par:
    # Definição dos parâmetros
    alpha1 = p[0]
```

```
    alpha2 = p[1]

    # Definição dos arrays onde serão armazenados os valores
    t = np.arange(0, days+1, 1)
    Svals = []
    Evals = []
    IAvals = []
    ISvals = []
    Rvals = []
    Pvals = []

    # Armazenamento dos valores iniciais
    Svals.append(S0)
    Evals.append(E0)
    IAvals.append(IA0)
    ISvals.append(IS0)
    Rvals.append(R0)
    Pvals.append(P0)

    # Cálculo dos valores seguintes
    i = 1
    while (i <= days):
        vals = RK(1, Svals[i-1], Evals[i-1], IAvals[i-1], ISvals[i-1], Rvals[i-1],
        ↪  Pvals[i-1])
        Svals.append(vals[0])
        Evals.append(vals[1])
        IAvals.append(vals[2])
        ISvals.append(vals[3])
        Rvals.append(vals[4])
        Pvals.append(vals[5])
        i += 1

    # Armazenamento dos dados
    dataS.append(Svals)
    dataE.append(Evals)
    dataIA.append(IAvals)
    dataIS.append(ISvals)
    dataR.append(Rvals)
    dataP.append(Pvals)

print("\n---- Combinações consideradas ----")
for p in par:
    leg = '$\\alpha _1 = $' + str(p[0]) + "$, \\alpha_2 = $" + str(p[1])
    print("alpha_1 = %f, alpha_2 = %f" % (p[0], p[1]))
    plt.figure(0)
```

```
    plt.plot(t, dataS[p[2]], color='lightskyblue', linestyle=p[3], linewidth=1, label=leg)
    plt.title('Grupo suscetível')
    plt.figure(1)
    plt.plot(t, dataE[p[2]], color='darkorange', linestyle=p[3], linewidth=1, label=leg)
    plt.title('Grupo exposto')
    plt.figure(2)
    plt.plot(t, dataIA[p[2]], color='indigo', linestyle=p[3], linewidth=1, label=leg)
    plt.title('Infetados assintomáticos')
    plt.figure(3)
    plt.plot(t, dataIS[p[2]], color='maroon', linestyle=p[3], linewidth=1, label=leg)
    plt.title('Infetados sintomáticos')
    plt.figure(4)
    plt.plot(t, dataR[p[2]], color='forestgreen', linestyle=p[3], linewidth=1, label=leg)
    plt.title('Recuperados')
    plt.figure(5)
    plt.plot(t, dataP[p[2]], color='orchid', linestyle=p[3], linewidth=1, label=leg)
    plt.title('Patógenos')
it = 0
while (it < 6):
    plt.figure(it)
    plt.xlabel("Dias")
    plt.legend(loc='best', shadow=False, fontsize='small')
    it += 1
plt.show()
```

B Ficheiros de texto

ProjetoMC_Grupo2_Exercicio2b.txt

```
Eis a tabela final para número de passos inicial n=10 e 7 iteradas:
  n          Q_n               |Q_2n - Q_n|       |(Q_2n - Q_n)/(Q_n-Q_n/2)|
 10    15.449224165648204    5.950797721254730e-08             -
 20    15.449224225156181    5.648335132946158e-10     9.491727659926581e-03
 40    15.449224225721014    8.174794174919953e-12     1.447292694662754e-02
 80    15.449224225729189    1.225686219186173e-13     1.499348109517601e-02
160    15.449224225729312    5.329070518200751e-15     4.347826086956522e-02
320    15.449224225729317    0.000000000000000e+00     0.000000000000000e+00
640    15.449224225729317             -                        -
```

ProjetoMC_Grupo2_Exercicio2c.txt - obtenção dos valores das Tabelas 5, 6 e 7

```
Eis a tabela final para a regra dos trapézios composta, número de passos inicial
↪  n=10 e 20 iterações:
     n           Q_n               |Q_2n - Q_n|       |(Q_2n - Q_n)/(Q_n-Q_n/2)|
     10    15.514850849804633    4.911887370212931e-02             -
     20    15.465731976102504    1.237445121426539e-02     2.519286433420186e-01
     40    15.453357524888238    3.099576353031708e-03     2.504819243586726e-01
     80    15.450257948535207    7.752672223304558e-04     2.501203822813282e-01
    160    15.449482681312876    1.938401324359518e-04     2.500300887908916e-01
    320    15.449288841180440    4.846149114534626e-05     2.500075218497841e-01
    640    15.449240379689295    1.211546392632101e-05     2.500018806681820e-01
   1280    15.449228264225368    3.028871649490839e-06     2.500004678244779e-01
   2560    15.449225235353719    7.572182791903970e-07     2.500001211070424e-01
   5120    15.449224478135440    1.893046377432483e-07     2.500000897305981e-01
  10240    15.449224288830802    4.732610925373137e-08     2.499997349136222e-01
  20480    15.449224241504693    1.183144782146428e-08     2.499983203358608e-01
  40960    15.449224229673245    2.958053357815515e-09     2.500161774325791e-01
  81920    15.449224226715192    7.395684065159003e-10     2.500186159799572e-01
 163840    15.449224225975623    1.848246000690779e-10     2.499087284430994e-01
 327680    15.449224225790799    4.636468986518594e-11     2.508577854238950e-01
 655360    15.449224225744434    1.290345608140342e-11     2.783035132753534e-01
1310720    15.449224225731530    1.387334691571596e-12     1.075165198237885e-01
2621440    15.449224225730143    2.717825964282383e-13     1.959026888604353e-01
5242880    15.449224225729871             -                        -

Eis a tabela final para a regra de Simpson composta, número de passos inicial n=10
↪  e 16 iterações:
     n           Q_n               |Q_2n - Q_n|       |(Q_2n - Q_n)/(Q_n-Q_n/2)|
     10    15.451576779535070    2.217761333270118e-03             -
     20    15.449359018201800    1.263103849886704e-04     5.695400271156504e-02
```

```
     40    15.449232707816812    7.951399279448879e-06     6.295127103097731e-02
     80    15.449224756417532    4.975120990735604e-07     6.256912545687728e-02
    160    15.449224258905433    3.110248414373018e-08     6.251603569369973e-02
    320    15.449224227802949    1.944030714184919e-09     6.250403360712935e-02
    640    15.449224225858918    1.214974787444589e-10     6.249771562657622e-02
   1280    15.449224225737421    7.585043704239069e-12     6.242963872684475e-02
   2560    15.449224225729836    4.938272013532696e-13     6.510538641686182e-02
   5120    15.449224225729342    4.973799150320701e-14     1.007194244604317e-01
  10240    15.449224225729292    7.105427357601002e-14     1.428571428571429e+00
  20480    15.449224225729363    5.329070518200751e-14     7.500000000000000e-01
  40960    15.449224225729310    3.019806626980426e-14     5.666666666666667e-01
  81920    15.449224225729340    6.750155989720952e-14     2.235294117647059e+00
 163840    15.449224225729273    3.019806626980426e-14     4.473684210526316e-01
 327680    15.449224225729242              -                         -

Eis a tabela final para a regra dos retângulos, número de passos inicial n=10 e 20
↪  iterações:
      n             Q_n                 |Q_2n - Q_n|        |(Q_2n - Q_n)/(Q_n-Q_n/2)|
     10    15.416613102400387    2.436997127357898e-02               -
     20    15.440983073673966    6.175298508216187e-03     2.533978575063491e-01
     40    15.447158372182182    1.549041908360138e-03     2.508448630133668e-01
     80    15.448707414090542    3.875869574443414e-04     2.502107627640962e-01
    160    15.449095001047986    9.691715018078639e-05     2.500526612655793e-01
    320    15.449191918198167    2.423056331224416e-05     2.500131634808203e-01
    640    15.449216148761479    6.057720572272274e-06     2.500032910589080e-01
   1280    15.449222206482052    1.514435110649970e-06     2.500008200414400e-01
   2560    15.449223720917162    3.786090481128213e-07     2.500001785816552e-01
   5120    15.449224099526210    9.465240324857405e-08     2.500003729978706e-01
  10240    15.449224194178614    2.366303242240519e-08     2.499992774643224e-01
  20480    15.449224217841646    5.915802958611494e-09     2.500018954886844e-01
  40960    15.449224223757449    1.478701605606147e-09     2.499578866895215e-01
  81920    15.449224225236151    3.697930850421471e-10     2.500795857934854e-01
 163840    15.449224225605944    9.250378241176804e-11     2.501501140867059e-01
 327680    15.449224225698448    2.288302880515403e-11     2.473739798367739e-01
 655360    15.449224225721331    4.384048679639818e-12     1.915851575842261e-01
1310720    15.449224225725715    2.987832203871221e-12     6.815235008103727e-01
2621440    15.449224225728702    7.993605777301127e-13     2.675386444708680e-01
5242880    15.449224225729502              -                         -
```

ProjetoMC_Grupo2_Exercicio2c.txt - obtenção dos gráficos das Figuras 8, 9 e 10

```
Eis a tabela final para a quadratura de Gauss composta, número de passos inicial
↪ n=10 e 5 iterações:
    n          Q_n                 |Q_2n - Q_n|        |(Q_2n - Q_n)/(Q_n-Q_n/2)|
    10    15.449224165648204    5.950797721254730e-08             -
    20    15.449224225156181    5.648335132946158e-10     9.491727659926581e-03
    40    15.449224225721014    8.174794174919953e-12     1.447292694662754e-02
    80    15.449224225729189    1.225686219186173e-13     1.499348109517601e-02
   160    15.449224225729312             -                        -

Eis a tabela final para a regra dos trapézios composta, número de passos inicial
↪ n=10 e 5 iterações:
    n          Q_n                 |Q_2n - Q_n|        |(Q_2n - Q_n)/(Q_n-Q_n/2)|
    10    15.514850849804633    4.911887370212931e-02             -
    20    15.465731976102504    1.237445121426539e-02     2.519286433420186e-01
    40    15.453357524888238    3.099576353031708e-03     2.504819243586726e-01
    80    15.450257948535207    7.752672223304558e-04     2.501203822813282e-01
   160    15.449482681312876             -                        -

Eis a tabela final para a regra de Simpson composta, número de passos inicial n=10
↪ e 5 iterações:
    n          Q_n                 |Q_2n - Q_n|        |(Q_2n - Q_n)/(Q_n-Q_n/2)|
    10    15.451576779535070    2.217761333270118e-03             -
    20    15.449359018201800    1.263103849886704e-04     5.695400271156504e-02
    40    15.449232707816812    7.951399279448879e-06     6.295127103097731e-02
    80    15.449224756417532    4.975120990735604e-07     6.256912545687728e-02
   160    15.449224258905433             -                        -

Eis a tabela final para a regra dos retângulos, número de passos inicial n=10 e 5
↪ iterações:
    n          Q_n                 |Q_2n - Q_n|        |(Q_2n - Q_n)/(Q_n-Q_n/2)|
    10    15.416613102400387    2.436997127357898e-02             -
    20    15.440983073673966    6.175298508216187e-03     2.533978575063491e-01
    40    15.447158372182182    1.549041908360138e-03     2.508448630133668e-01
    80    15.448707414090542    3.875869574443414e-04     2.502107627640962e-01
   160    15.449095001047986             -                        -

Eis a tabela final para a quadratura de Gauss composta, número de passos inicial
↪ n=10 e 5 iterações:
    n          Q_n                 |Q_2n - Q_n|        |(Q_2n - Q_n)/(Q_n-Q_n/2)|
    10    15.449224165648204    5.950797721254730e-08             -
    20    15.449224225156181    5.648335132946158e-10     9.491727659926581e-03
    40    15.449224225721014    8.174794174919953e-12     1.447292694662754e-02
    80    15.449224225729189    1.225686219186173e-13     1.499348109517601e-02
```

```
   160    15.449224225729312              -                        -

Eis a tabela final para a regra de Simpson composta, número de passos inicial n=10
↪  e 5 iterações:
     n             Q_n                |Q_2n - Q_n|      |(Q_2n - Q_n)/(Q_n-Q_n/2)|
     10    15.451576779535070    2.217761333270118e-03             -
     20    15.449359018201800    1.263103849886704e-04     5.695400271156504e-02
     40    15.449232707816812    7.951399279448879e-06     6.295127103097731e-02
     80    15.449224756417532    4.975120990735604e-07     6.256912545687728e-02
    160    15.449224258905433              -                        -
```