Ariel Godinho
Felipe Vasconcelos

Modelo Epidemiológico Modificado (SAIR) para vírus de computador

Brasil
2018, São Paulo

Ariel Godinho
Felipe Vasconcelos

Modelo Epidemiológico Modificado (SAIR) para vírus de computador

Relatório Técnico para a resolução do Modelo Epidemiológico Modificado (SAIR) para vírus de computador, como parte dos requisitos necessários para aprovação na disciplina Métodos Numéricos e Aplicações (MAP3122), dentro do Programa de Graduação em Engenharia de Computação.

Escola Politécnica da Universidade de São Paulo – POLI-USP

Programa de Graduação em Engenharia de Computação

MAP3122 – Métodos Numéricos e Aplicações

Brasil

2018, São Paulo

Resumo

Neste Relatório Técnico, iremos aplicar um método numérico para prever o comportamento de um modelo matemático que estuda o comportamento de populações frente a uma doença. O modelo específico escolhido estuda a propagação de vírus de computador em uma rede qualquer, por meio da análise populacional, categorizando os indivíduos em 4 estados possíveis (Suscetível, Infectado, Removido e Antidotal).

A base da comprovação dos resultados obtidos é a validação do método via solução manufaturada. Assim, criamos um problema com solução conhecida e o resolvemos usando nosso método, comparando a aproximação obtida com a solução exata.

Este trabalho é baseado no modelo proposto em "A Modified Epidemioogical Model for Computer Viruses"(PIQUEIRA, 2009).

Palavras-chaves: Cálculo Numérico, Modelagem Matemática, Runge-Kutta, SAIR, SIR, Vírus de Computador.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

SIR	Modelo epidemiológico Suscetível-Infectado-Recuperado
SAIR	Modelo epidemiológico Suscetível-Infectado-Recuperado-Vacinado

Lista de símbolos

S	População de computadores suscetíveis não-infectados
A	População de computadores vacinados não-infectados
I	População de computadores infectados
R	População de computadores removidos
N	Qtd. de novos computadores adicionados a população
μ	Coef. da taxa de mortalidade não relacionada à infecção
β	Coef. de interação entre susceptíveis e infectados
α_{SA}	Coef. de interação entre susceptíveis e vacinados
α_{IA}	Coef. de interação entre infectados e vacinados
σ	Coef. de computadores consertados que voltam como susceptíveis
δ	Coef. de computadores removidos por inutilidade após infecção

Sumário

1 Introdução

1.1 Histórico

O primeiro trabalho acadêmico sobre a teoria dos programas de computador auto-replicantes foi feito em 1949 por John von Neumann, sendo posteriormente publicado como a "Teoria dos autômatos auto-reproduzidos". Em seu ensaio, von Neumann descreveu como um programa de computador poderia ser projetado para se reproduzir, sendo o design deste programa considerado o primeiro vírus de computador do mundo. Em 1972, Veith Risak, construindo diretamente sobre o trabalho de von Neumann sobre a auto-replicação, publicou seu artigo “Autômatos Auto Replicantes com Troca Mínima de Informações”. O artigo descreve um vírus totalmente funcional escrito em linguagem de programação para um sistema de computador. Em 1980, Jürgen Kraus escreveu sua tese de diploma “Auto-reprodução de programas”. Em seu trabalho, Kraus postulou que os programas de computador podem se comportar de forma semelhante à vírus biológicos.

1.2 Cenário Tecnológico

Infecções eram raras antes da internet se popularizar, e até os primeiros ataques pós-internet eram mais maliciosos do que criminosos. Com o aumento dos serviços financeiros on-line, a popularidade do comércio eletrônico e a presença de um mercado negro para informações de identificação pessoal, o malware tornou-se um grande negócio. Em retrospectiva, os primeiros dias de luta contra vírus eram quase pitorescos. As primeiras ferramentas eram basicamente verificadores de assinatura que procuravam mudanças em sistemas de arquivos ou aplicativos que correspondem a padrões conhecidos e, em seguida, sinalizavam ou bloqueavam a execução dos programas. Esta técnica ainda é usada hoje, mas tem algumas fraquezas fundamentais. Entre eles está a falha dos usuários em atualizar seus softwares regularmente e o fato de que leva tempo para catalogar todas as novas variantes de virus que são criadas todos os dias.
A arma mais comum de hoje é a verificação heurística de vírus, em que o código é analisado contra um conjunto de regras que indicam a presença de um vírus. Embora a abordagem heurística possa detectar a grande maioria dos vírus mais antigos, tem alguns dos mesmos pontos fracos que a abordagem da assinatura. Os desenvolvedores de vírus estão constantemente descobrindo novas maneiras de quebrar as regras, e é difícil para os fabricantes de software acompanharem.

1.3 Automodificação de Vírus

A maioria dos programas antivírus modernos tentam encontrar padrões de vírus dentro dos programas escaneando-os à procura de “assinaturas de vírus”. Infelizmente, o termo é enganador, na medida em que os vírus não possuem assinaturas únicas como os seres humanos tem. Essa "assinatura"de vírus é apenas uma seqüência de bytes que um programa antivírus busca porque é conhecido como parte do vírus. Um termo melhor seria "string de pesquisa". Diferentes programas antivírus irão empregar diferentes strings de pesquisa e, de fato, diferentes métodos de busca, na tentativa de identificar vírus. Se um scanner de vírus encontrar esse padrão em um arquivo, ele executará outras verificações para se certificar de que encontrou o vírus e não apenas uma sequência coincidente em um arquivo inocente, antes de notificar o usuário de que o arquivo está infectado. O usuário pode então excluir, ou (em alguns casos) "limpar"ou "curar"o arquivo infectado. Alguns vírus empregam técnicas que dificultam a detecção por assinatura, mas provavelmente não são impossíveis de serem detectados. Esses vírus modificam seu código em cada infecção. Ou seja, cada arquivo infectado contém uma variante diferente do vírus.

1.3.1 Vírus Cifrado

Um método de evadir a detecção de assinaturas é usar criptografia simples para cifrar (codificar) o corpo do vírus, deixando apenas o módulo de criptografia e uma chave criptográfica estática em texto claro que não muda de uma infecção para a próxima. Nesse caso, o vírus consiste em um pequeno módulo de decodificação e uma cópia criptografada do código do vírus. Se o vírus estiver criptografado com uma chave diferente para cada arquivo infectado, a única parte do vírus que permanece constante é o módulo de decodificação, que seria (por exemplo) anexado ao final. Neste caso, um scanner de vírus não pode detectar diretamente o vírus usando assinaturas, mas ainda pode detectar o módulo de decodificação, o que ainda possibilita a detecção indireta do vírus. Uma vez que estas seriam chaves simétricas armazenadas no hospedeiro infectado, é perfeitamente possível decifrar o vírus final, mas isso provavelmente não é necessário, uma vez que o código auto-modificador é de tamanha raridade que pode ser motivo para o antivírus pelo menos "marcar"o arquivo como suspeito.
Uma maneira antiga, mas compacta, seria o uso de operações aritmética como adição ou subtração e o uso de condições lógicas, como XOR, onde cada byte em um vírus é com uma constante, de modo que a operação XOR tenha apenas que ser repetida para decodificação. É suspeito que um código se modifique, então o código para criptografar/decriptografar pode ser parte da assinatura em muitas definições de vírus. Uma abordagem mais antiga e simples não usava uma chave, onde a criptografia consiste apenas em operações sem parâmetros, como incrementar e decrementar, rotação bit a bit, negação aritmética e operações NOT. Alguns vírus farão uso de um meio de criptografia dentro de um executável

em que o vírus é cifrado em determinados eventos, como o scanner de vírus que está sendo desabilitado para atualizações ou o computador sendo reiniciado. Isso é chamado de criptovirologia. Nos tempos indicados, o executável irá decifrar o vírus e executar suas rotinas ocultas, infectando o computador e às vezes desativando o software antivírus.

1.3.2 Código Polimórfico

O código polimórfico foi a primeira técnica que representou uma séria ameaça para os scanners de vírus. Assim como os vírus criptografados comuns, um vírus polimórfico infecta arquivos com uma cópia criptografada de si mesma, que é decodificada por um módulo de decriptografia. No caso de vírus polimórficos, no entanto, este módulo de decodificação também é modificado em cada infecção. Um vírus polimórfico bem escrito não tem partes que permanecem idênticas entre as infecções, tornando muito difícil a detecção diretamente usando "assinaturas". O software antivírus pode detectá-lo, decifrando o vírus usando um emulador, ou por análise de padrão estatístico do corpo do vírus criptografado. Para que o código seja polimórfico, o vírus deve ter um mecanismo polimórfico (também chamado de "motor mutante"ou "motor de mutação") em algum lugar em seu corpo criptografado.
Alguns vírus empregam código polimórfico de uma forma que se restringe significativamente a taxa de mutação do vírus. Por exemplo, um vírus pode ser programado para se modificar apenas um pouco ao longo do tempo, ou pode ser programado para abster-se de mutar quando infecta um arquivo em um computador que já contém cópias do vírus. A vantagem de usar esse código polimórfico lento é que torna mais difícil para profissionais e pesquisadores antivírus obter amostras representativas do vírus, porque os arquivos de "isca"que são infectados em uma execução normalmente contêm amostras idênticas ou similares do vírus. Isso tornará mais provável que a detecção pelo scanner de vírus não seja confiável e que algumas instâncias do vírus possam evitar a detecção.

1.3.3 Código Metamórfico

Para evitar ser detectado por emulação, alguns vírus se reescrevem completamente cada vez que estão para infectar novos executáveis. Os vírus que utilizam esta técnica são ditos ter código metamórfico. Para permitir o metamorfismo, é necessário um "motor metamórfico". Um vírus metamórfico geralmente é muito grande e complexo. Por exemplo, o W32/Simile consistiu em mais de 14.000 linhas de código de linguagem de montagem, 90% dos quais faz parte do mecanismo metamórfico.

1.4 Importância de Sistemas Seguros

O combate a programas maliciosos são de extrema importância na atualidade e continuarão a ser postos em consideração no futuro, pois o avanço de novas tecnologias e de programas maliciosos pedem que se tenha um esforço proporcional na busca por formas de proteger os sistemas atuais.
Assim dentro da necessidade de garantir a segurança e confiabilidade dos sistemas é fundamental entender como ocorre o processo de contaminação dos vírus em uma rede de computador. Pois assim, é possível ter um maior entendimento para criar soluções mais eficazes no combate a esses vírus.

1.5 Modelagem Inspirada na Área Biológica

O nome de vírus de computador não foi escolhido por acaso, mas sim por sua semelhança com o processo biológico de contaminação de doenças. O estudo de modelagem de vírus de computador como um vírus dentro da área biológica vem sendo pesquisada desde 1980, modelando essa dinâmica de modo análogo à contaminação de doenças reais. Essa modelagem pela área biológica pode ser feita por 2 cenários diferentes, uma micro – mais focada na proteção de um único sistema – e outro macro – mais focada no conjunto de sistemas conectados em uma rede única.
No cenário micro os vírus são vistos dentro do cenário com um único computador e de como se pode mitigar a contaminação desse computador por meios preventivos. Mas essa visão se torna limitada quando se pensa em uma rede de computadores, dificultando a visibilidade de possíveis soluções com uma atuação em massa para toda a rede de computadores. No cenário macro, é possível melhor entender essas forças que influenciam a dinâmica de propagação dos vírus em uma rede. Nessa visão, os estudos foram positivamente baseados em variações do modelo SIR, que modelam baseados em 3 estados para cada indivíduo (que nesse caso é um computador).
Os 3 estados são: Suscetível, Infectado e Recuperado. O estado suscetível caracteriza um sistema que pode ser infectado, mas que ainda não teve contato com o vírus. Já o estado de Infectado caracteriza os sistemas que estão infectados por esse vírus. Enquanto que o estado de Recuperado caracteriza os computadores que foram recuperados da infecção desse vírus.
O modelo usado para a solução do nosso problema será uma variante do modelo epidemiológico SIR, para representar indivíduos que possuam uma imunização contra o vírus em circulação na rede. O modelo possui o acrônimo de SAIR, em que A representar os indivíduos vacinados. Com esse modelo será possível aumentar a precisão da modelagem desse problema se comparado com a realidade dessa dinâmica. Na seção seguinte será descrito em maiores detalhes o funcionamento do Modelo SAIR proposto.

1.6 Sequência do Relatório

Nas sequência do relatório, na seção de Modelagem Matemática será exposto a modelagem do problema proposto usando o Modelo Epidemiológico Modificador SAIR.
Na seção de Metodologia Numérica iremos desenvolver como resolveremos o problema através das metodologias numéricas aprendidas na disciplina.
Na seção de Resultados será possível ver os resultados obtidos na solução do problema utilizando a metodologia explicada na seção de Metodologia Numérica.
Na seção de Conclusão será endereçado uma recapitulação de todo relatório evidenciando os pontos chave da resolução do problema proposto.

Parte I

Modelagem Matemática

2 Modelo SAIR

2.1 Descrição

O Modelo SIR, acrônimo para Modelo Suscetível-Infectado-Removido, baseia-se na interação entre 3 possíveis grupos para uma população de indivíduos – suscetíveis não-infectados, infectados e removidos por infecção ou não – frente a uma doença em questão. No caso deste trabalho, a doença é um vírus de computador, e consequentemente, os indivíduos são os computadores. Para uma aproximação mais realista do comportamento de um vírus entre os computadores, esse modelo modificado propõe mais um novo estado para essa população: o estado de indivíduos vacinados não-infectados (traduzido de "antidotal"do inglês). No caso, a vacina significaria que o computador possui um antivírus capaz de identificar o vírus e assim, não pode ser infectado. A seguir, é possível ver a descrição para o Modelo SAIR proposto (PIQUEIRA, 2009), assim como a explicação de cada variável. Na Figura 1 podemos ver a interação entre as quatro populações e seus coeficientes de interação e na equação 2.1 podemos ver o modelo completo.

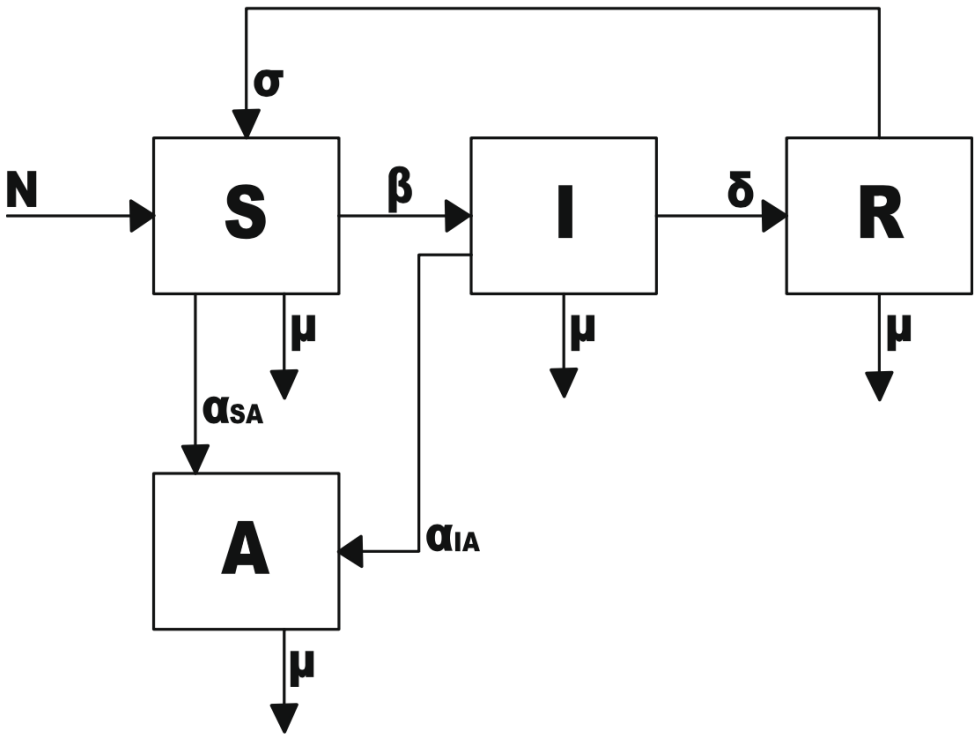

Figura 1: Interação entre os indivíduos no modelo SAIR.

S População de computadores suscetíveis não-infectados

A População de computadores vacinados não-infectados

I População de computadores infectados

R População de computadores removidos

N Qtd. de novos computadores adicionados a população

μ Coef. da taxa de mortalidade não relacionada à infecção

β Coef. de interação entre susceptíveis e infectados

α_{SA} Coef. de interação entre susceptíveis e vacinados

α_{IA} Coef. de interação entre infectados e vacinados

σ Coef. de computadores consertados que voltam como susceptíveis

δ Coef. de computadores removidos por inutilidade após infecção

$$\begin{cases} \dot{S} = N - \alpha_{SA}SA - \beta SI - \mu S + \sigma R \\ \dot{I} = \beta SI - \alpha_{IA}AI - \delta I - \mu I \\ \dot{R} = \delta I - \sigma R - \mu R \\ \dot{A} = \alpha_{SA}SA + \alpha_{IA}AI - \mu A \end{cases} \tag{2.1}$$

2.2 Equações e Modelagem

Supondo que a velocidade de propagação do vírus é mais rápida que a entrada de novos computadores na rede e da obsolência de computadores existentes na rede, usaremos $N = 0$ e $\mu = 0$. A partir dessa suposição e analisando o diagrama da Figura 1, vemos que o sistema que era aberto se torna um sistema fechado, ou seja, a população ao longo do tempo se mantém constante. O que simplifica o modelo para o apresentado na equação 2.2:

Nota-se que T = S + I + R + A, logo podemos eliminar uma das equações, tomar T = 100 e usar S, I, R e A como porcentagens.

$$\begin{cases} \dot{S} = -\alpha_{SA}SA - \beta SI + \sigma R \\ \dot{I} = \beta SI - \alpha_{IA}AI - \delta I \\ \dot{R} = \delta I - \sigma R \\ \dot{A} = \alpha_{SA}SA + \alpha_{IA}AI \end{cases} \tag{2.2}$$

2.2.1 Pontos de Equilíbrio

Um ponto de equilíbrio ocorre quando a proporção entre os indivíduos não muda com a variação do tempo, significando que o sistema convergiu para um ponto no sistema de coordenadas S x I x R.

2.2.1.1 Não-Endêmicos

Os pontos de equilíbrio não-endêmico significam que o sistema não possui indivíduos infectados ($I = 0$) ao atingir o equilíbrio.
O ponto de equilíbrio não-endêmico $P_1 = (S, I, R, A) = (0, 0, 0, T)$ e seus parâmetros iniciais são:

$T = 100$ $\quad$ $\mu = 0$

$S_0 = 74$ $\quad$ $\beta = 0.1$

$A_0 = 1$ $\quad$ $\alpha_{SA} = 0.025$

$I_0 = 25$ $\quad$ $\alpha_{IA} = 0.25$

$R_0 = 0$ $\quad$ $\sigma = 0.8$

$N = 0$ $\quad$ $\delta = 20$

O ponto de equilíbrio não-endêmico $P_2 = (S, I, R, A) = (T, 0, 0, 0)$ que indica um equilíbrio sem indivíduos infectados, tem parâmetros iniciais, tais que:

$T = 100$ $\quad$ $\mu = 0$

$S_0 = 75$ $\quad$ $\beta = 0.1$

$A_0 = 0$ $\quad$ $\alpha_{SA} = 0.025$

$I_0 = 25$ $\quad$ $\alpha_{IA} = 0.25$

$R_0 = 0$ $\quad$ $\sigma = 0.8$

$N = 0$ $\quad$ $\delta = 20$

2.2.1.2 Endêmicos

Um Ponto de Equilíbrio Endêmico ocorre quando a proporção entre os indivíduos não muda e temos $I \neq 0$. Neste ponto, denominado P3, temos que:

$$\begin{cases} S = \dfrac{\delta}{\beta} \\ \\ I = \dfrac{T - \dfrac{\delta}{\beta}}{1 + \dfrac{\delta}{\sigma}} \\ \\ R = \dfrac{T - \dfrac{\delta}{\beta}}{1 + \dfrac{\sigma}{\delta}} \\ \\ A = 0 \end{cases} \tag{2.3}$$

Para tal ponto existir, é necessário que $T < \dfrac{\delta}{\beta}$, portanto os valores iniciais escolhidos para a equação serão:

$T = 100$	$\mu = 0$
$S_0 = 95$	$\beta = 0.1$
$A_0 = 0$	$\alpha_{SA} = 0.025$
$I_0 = 5$	$\alpha_{IA} = 0.25$
$R_0 = 0$	$\sigma = 0.8$
$N = 0$	$\delta = 9$

Assim podemos calcular o ponto P3 usando a equação 2.3:

$$\begin{cases} S = 90 \\ I = 0.82 \\ R = 9.18 \\ A = 0 \end{cases} \tag{2.4}$$

Logo, $P_3 = (90, 0.82, 9.18, 0)$.

Caso A seja diferente de 0, a equação convergirá para $P_1(0, 0, 0, T)$

Para todos os casos acima (P1, P2 e P3), escolhemos de forma arbitrária, o intervalo de tempo [0, 20] após realizar testes e verificar que foi atingindo os pontos de convergência P1, P2 e P3. Também assumidos uma unidade de tempo genérica, dado que o problema busca entender as dinâmicas de propagação a partir de uma configuração inicial segundo o modelo epidemiológico e não de representar fidedignamente as dinâmicas em uma unidade de tempo específica.

Parte II

Metodologia Numérica

3 Discretização do Domínio Computacional

3.1 Intervalo de estudo

Dado que o problema busca encontrar o estado de convergência a partir de uma configuração inicial. Escolhemos arbitrariamente o intervalo [0, 20] após realizar os primeiros testes da resolução do problema.

3.2 Passo de Integração

Fizemos testes com a resolução do problema para diferentes valores de m como vemos nas figuras 3.2 e 3.2.

Lembrando que

$$h = \frac{t_f - t_0}{2^{5+m}}$$

Como definimos o intervalo de [0, 20], temos que

$$h = \frac{20}{2^{5+m}}$$

Assim, escolhemos m = 4. Logo o passo de integração fica

$$h = \frac{20}{2^{5+4}} = \frac{20}{2^9} = \frac{5}{128}$$

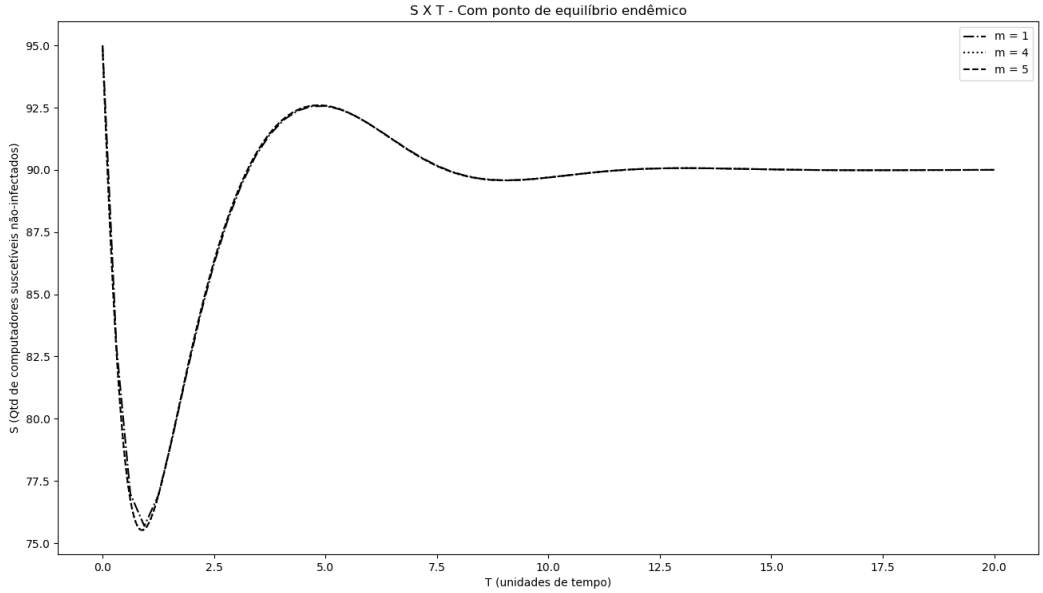

Figura 2: Teste para analisar convergência das curvas para diferentes valores de m.

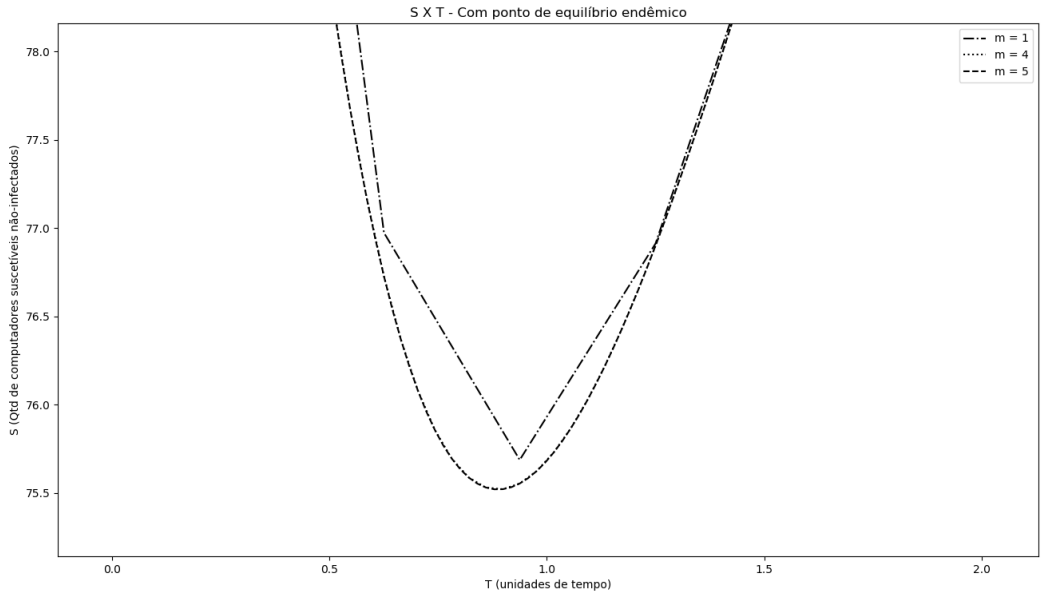

Figura 3: Aproximação para visualizar convergência das curvas para diferentes valores de m.

4 Discretização do Modelo Matemático

4.1 Método de Runge-Kutta

Os métodos de Runge–Kutta são uma familia de metódos iterativos implícitos e explícitos para a resolução numérica de equações diferenciais ordinárias. Estas técnicas foram desenvolvidas por volta de 1900 pelos matemáticos C. Runge e M.W. Kutta.

No nosso trabalho optamos por usar o RK33, devido a sua facilidade de intendimento e implementação.

4.1.1 Runge-Kutta de Terceira Ordem com Três Estágios - RK33

O RK33 é um método de passo simples usado para aproximar funções a partir de derivadas de primeira ordem.

O método é dado por:

$$y_{k+1} = y_k + h * \Phi(t_k, y_k, h)$$

$$\Phi(t, y, h) = \frac{1}{6}(k_1 + 4k_2 + k_3)$$

$$\begin{cases} k_1 = f(t, y) \\ k_2 = f(t + \frac{h}{2}, y + \frac{k_1}{2}) \\ k_3 = f(t + h, y - k_1 + 2k_2) \end{cases} \tag{4.1}$$

Para cada passo da aproximação, aplicamos o método para S, A, I e R, de forma a obter S_{k+1}, A_{k+1}, I_{k+1} e R_{k+1} a partir de S_k, A_k, I_k e R_k.

5 Método de Resolução Computacional

5.1 Teoria

A resolução do problema foi divida em duas partes. Em primeiro lugar, foi implementado o método iterativo explícito para resolução numérica do problema. Nessa etapa foi feito o teste da implementação via solução manufaturada (como será visto na próxima seção). Na segunda parte, usamos o método corretamente testado para resolver o problema proposto.

Para resolução com o método iterativo, foi preciso dividir o intervalo de estudo com um passo de integração definido. Para cada instante divido é calculado o método usando os valores da iteração anterior. Com os valores calculados em cada iteração é feito uma lista de valores para que seja plotado o gráfico.

Parte III

Resultados Numéricos

6 Verificação

6.1 Método das Soluções Manufaturadas

Esta seção consiste na verificação do método de Runge-Kutta de Terceira Ordem, via estratégia das soluções manufaturadas, implementado para este trabalho. Para isso, iremos criamos um Problema de Cauchy de Quatro Dimensões e, com o problema criado, iremos depurar o nosso programa para o método proposto.

6.2 Criação do Problema de Cauchy de 4 Dimensões

Dado o seguinte problema de Cauchy na forma normal:

$$\dot{y}(t) = f(t, y(t)) \quad \text{com} \quad y(t_0) = y_0 \tag{6.1}$$

Iremos criar um Problema de Cauchy de 4 dimensões, a partir da escolha de 4 soluções. Assim, seja as soluções exata escolhidas dada por:

$$\begin{cases} y_{e1}(t) = e^{t} \\ y_{e2}(t) = e^{2t} \\ y_{e3}(t) = e^{3t} \\ y_{e4}(t) = e^{4t} \end{cases} \tag{6.2}$$

Derivado as soluções e forçando as substituições das soluções exatas acima nos resultados das derivações:

$$\begin{cases} \dot{y_1}(t) = e^{t} = y_1(t) \\ \dot{y_2}(t) = 2e^{2t} = 2y_2(t) \\ \dot{y_3}(t) = 3e^{3t} = 3y_3(t) \\ \dot{y_4}(t) = 4e^{4t} = 4y_4(t) \end{cases} \tag{6.3}$$

Agora, escolhemos um valor no eixo t e calculamos a soluções exatas com esse valor t escolhido. Os resultados serão as condições iniciais do nosso problema. No caso, faremos no ponto $t_0 = 0$, assim:

$$\begin{cases} y_{e1}(t_0) = y_1(0) = 1 \\ y_{e2}(t_0) = y_2(0) = 1 \\ y_{e3}(t_0) = y_3(0) = 1 \\ y_{e4}(t_0) = y_4(0) = 1 \end{cases} \tag{6.4}$$

Pronto, agora temos o nosso Problema de Cauchy de 4 dimensões, que é dado por:

$$\begin{cases} y_1(t_0) = 1 \\ y_2(t_0) = 1 \\ y_3(t_0) = 1 \\ y_4(t_0) = 1 \\ \dot{y_1}(t) = y_1(t) \\ \dot{y_2}(t) = 2y_2(t) \\ \dot{y_3}(t) = 3y_3(t) \\ \dot{y_4}(t) = 4y_4(t) \end{cases} \tag{6.5}$$

Tal que $t_0 = 0$ e $t \in [0, 1]$

6.3 Resultados Numéricos Via Solução Manufaturada

Nesta seção são apresentados os gráficos das aproximações de y_1, y_2, y_3 e y_4 e suas respectivas tabelas de covergência. O objetivo desta seção é mostrar a convergência de maneira gráfica e numérica de modo a verificar a execução do método aplicado. Todas as Tabelas de Convergência mostram que o método implementado converge com ordem 3. Também é interessante notar como na Tabela da Figura 3, o erro se tornou tão pequeno que não foi mais possível representar o erro e calcular a convergência para m $>$ 8.

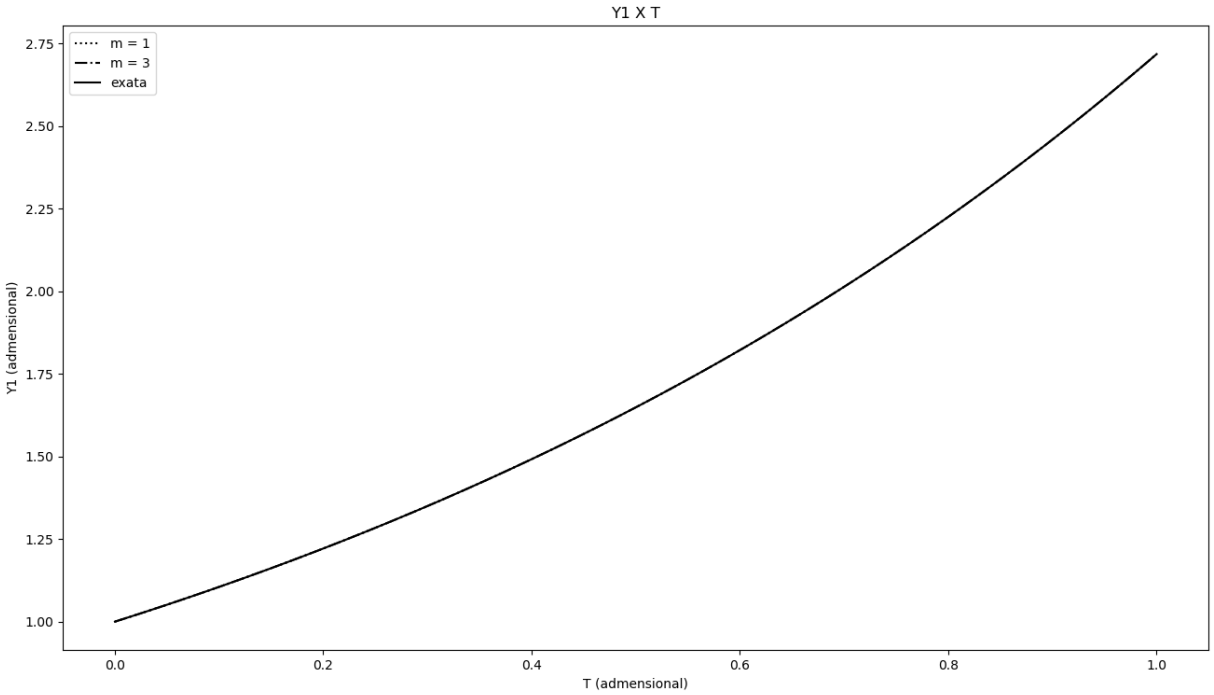

Figura 4: Aproximação de y_1 no intervalo de estudo.

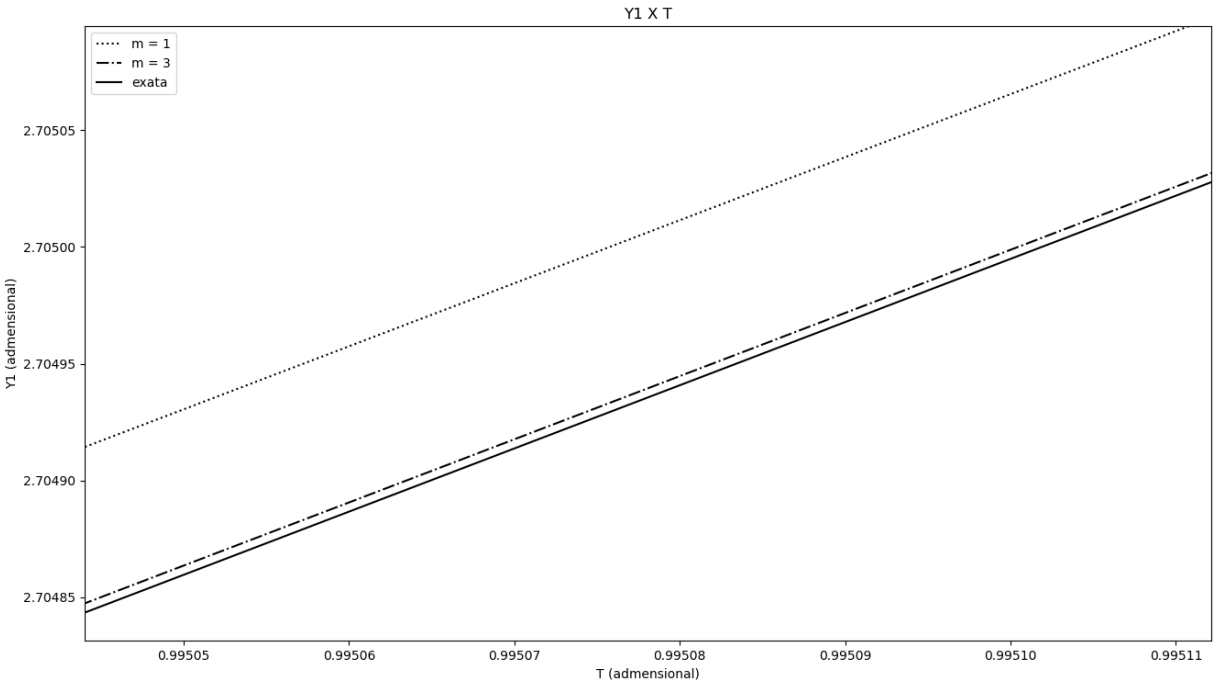

Figura 5: Aproximação de y_1 em um intervalo mais curto, evidenciando a convergência.

m	delta_t (m)	y (tf)	err = | e (tf , delta_t (m)) |	q = | err (m-1) / err (m) |	p'(m) = log 2 |q|
1	1.563E-02	2.71828E+00	4.26693E-07	-	-
2	7.813E-03	2.71828E+00	5.36710E-08	7.95017E+00	2.991
3	3.906E-03	2.71828E+00	6.72987E-09	7.97504E+00	2.995
4	1.953E-03	2.71828E+00	8.42550E-10	7.98750E+00	2.998
5	9.766E-04	2.71828E+00	1.05400E-10	7.99382E+00	2.999
6	4.883E-04	2.71828E+00	1.31801E-11	7.99690E+00	2.999
7	2.441E-04	2.71828E+00	1.66001E-12	7.93981E+00	2.989
8	1.221E-04	2.71828E+00	2.10054E-13	7.90275E+00	2.982
9	6.104E-05	2.71828E+00	1.99840E-14	1.05111E+01	3.394
10	3.052E-05	2.71828E+00	1.99840E-14	1.00000E+00	0.000

Figura 6: Tabela de convergência de y_1.

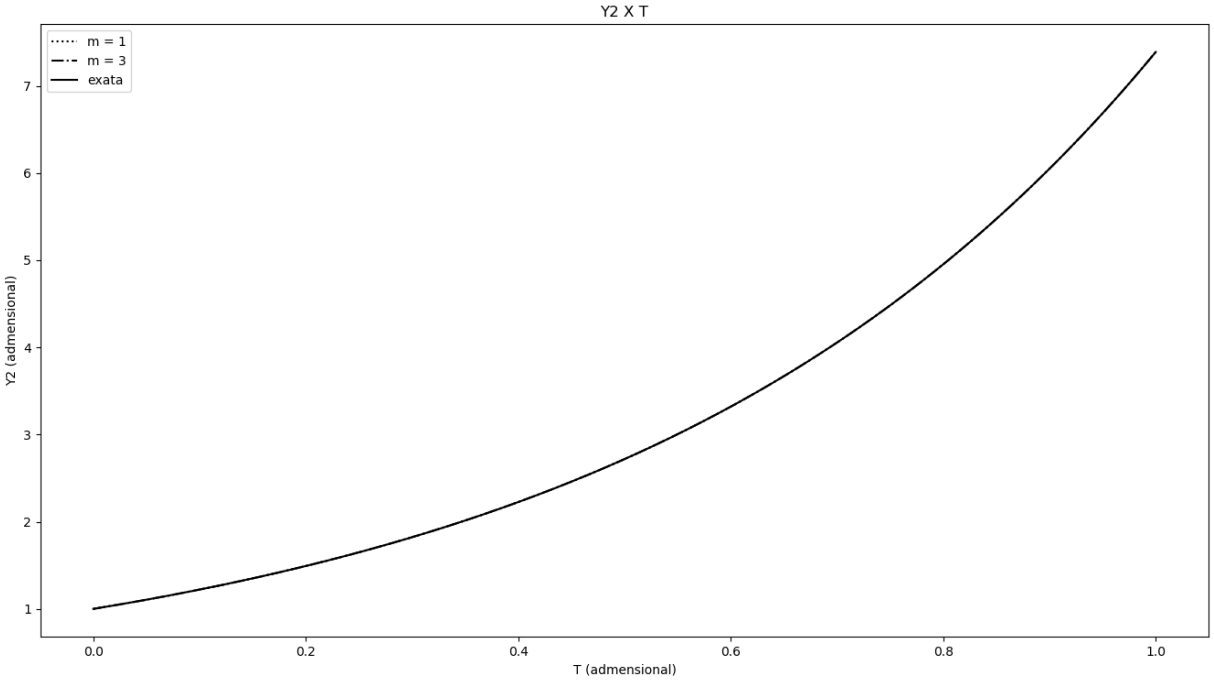

Figura 7: Aproximação de y_2 no intervalo de estudo.

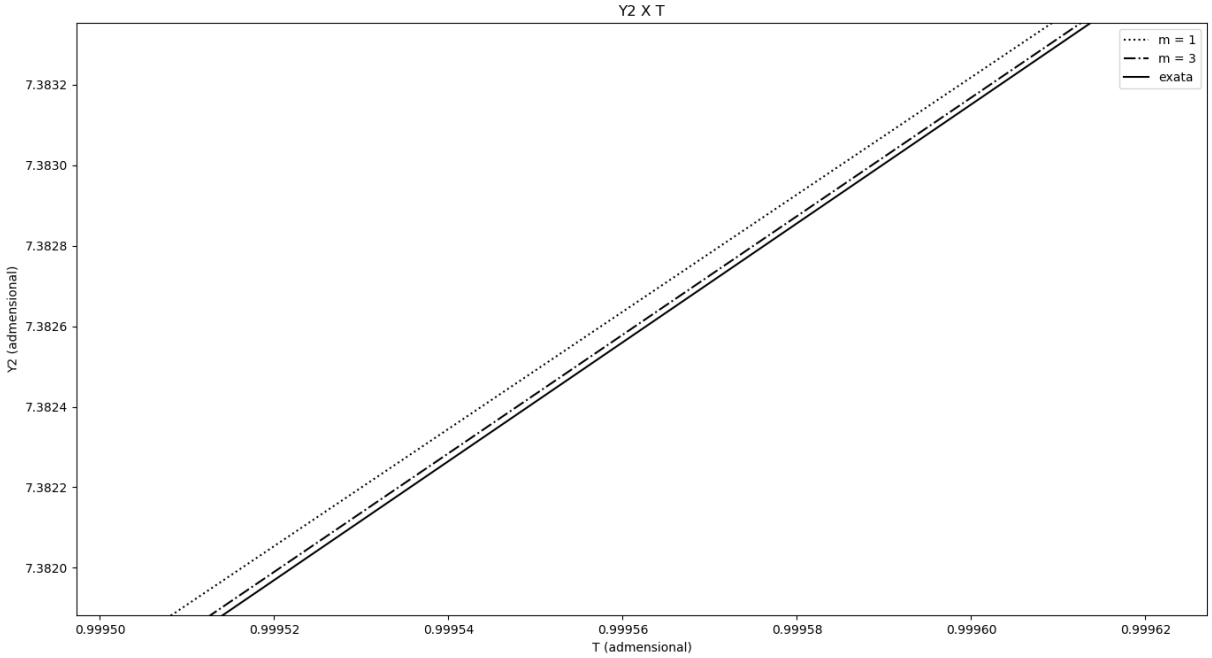

Figura 8: Aproximação de y_2 em um intervalo mais curto, evidenciando a convergência.

m	delta_t (m)	y (tf)	err = | e (tf , delta_t (m)) |	q = | err (m-1) / err (m) |	p'(m) = log 2 |q|
1	1.563E-02	7.38904E+00	1.83276E-05	-	-
2	7.813E-03	7.38905E+00	2.31975E-06	7.90069E+00	2.982
3	3.906E-03	7.38906E+00	2.91786E-07	7.95017E+00	2.991
4	1.953E-03	7.38906E+00	3.65873E-08	7.97504E+00	2.995
5	9.766E-04	7.38906E+00	4.58055E-09	7.98754E+00	2.998
6	4.883E-04	7.38906E+00	5.72999E-10	7.99399E+00	2.999
7	2.441E-04	7.38906E+00	7.16298E-11	7.99945E+00	3.000
8	1.221E-04	7.38906E+00	8.96971E-12	7.98574E+00	2.997
9	6.104E-05	7.38906E+00	1.15019E-12	7.79846E+00	2.963
10	3.052E-05	7.38906E+00	1.50102E-13	7.66272E+00	2.938

Figura 9: Tabela de convergência de y_2.

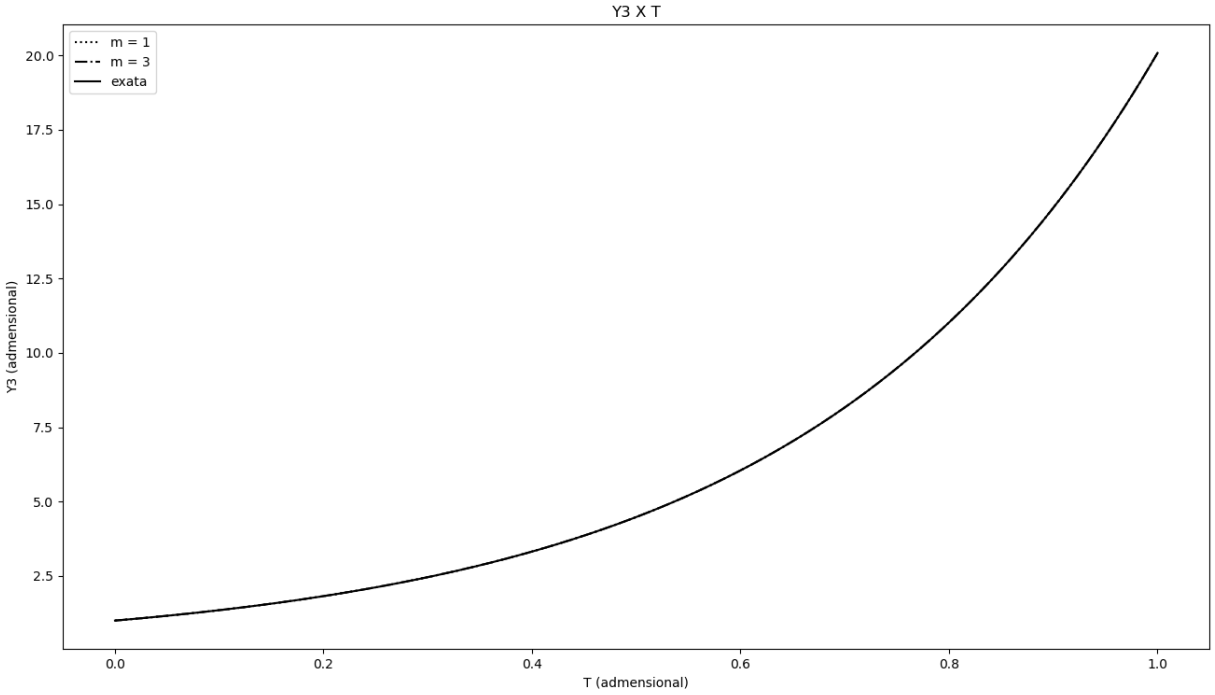

Figura 10: Aproximação de y_3 no intervalo de estudo.

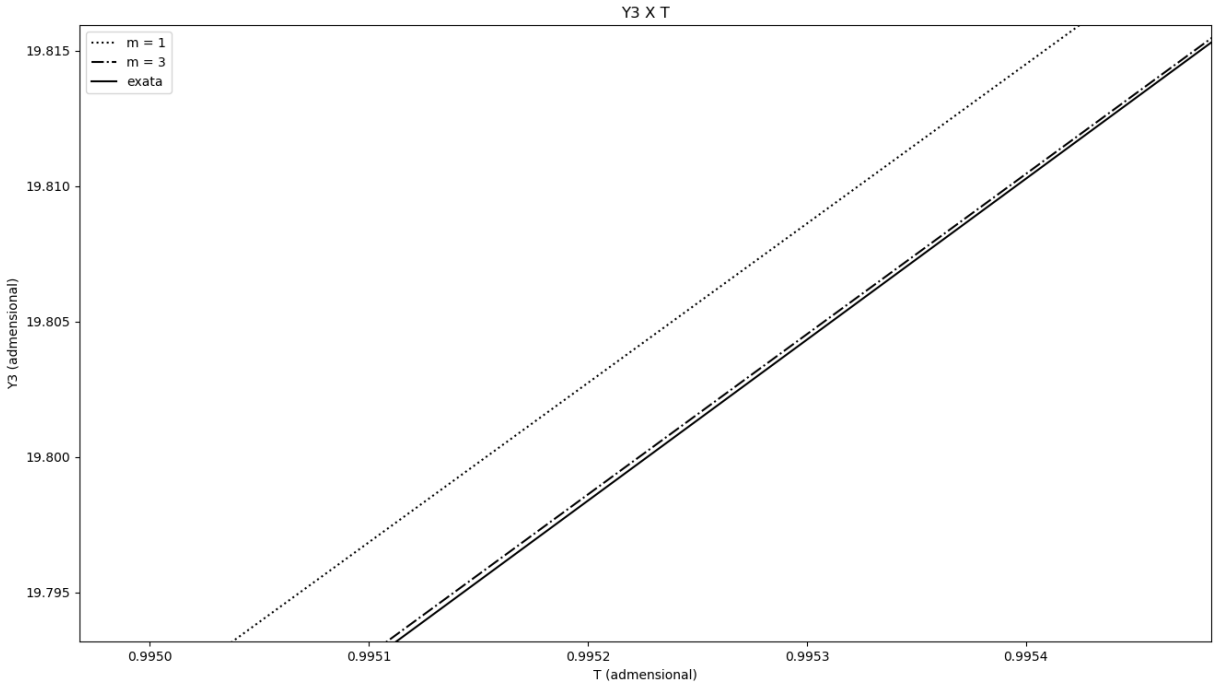

Figura 11: Aproximação de y_3 em um intervalo mais curto, evidenciando a convergência.

m	delta_t (m)	y (tf)	err = | e (tf , delta_t (m)) |	q = | err (m-1) / err (m) |	p'(m) = log 2 |q|
1	1.563E-02	2.00853E+01	2.49081E-04	-	-
2	7.813E-03	2.00855E+01	3.17239E-05	7.85153E+00	2.973
3	3.906E-03	2.00855E+01	4.00282E-06	7.92539E+00	2.986
4	1.953E-03	2.00855E+01	5.02703E-07	7.96260E+00	2.993
5	9.766E-04	2.00855E+01	6.29854E-08	7.98127E+00	2.997
6	4.883E-04	2.00855E+01	7.88240E-09	7.99064E+00	2.998
7	2.441E-04	2.00855E+01	9.85800E-10	7.99594E+00	2.999
8	1.221E-04	2.00855E+01	1.23201E-10	8.00156E+00	3.000
9	6.104E-05	2.00855E+01	1.54010E-11	7.99954E+00	3.000
10	3.052E-05	2.00855E+01	2.19913E-12	7.00323E+00	2.808

Figura 12: Tabela de convergência de y_3.

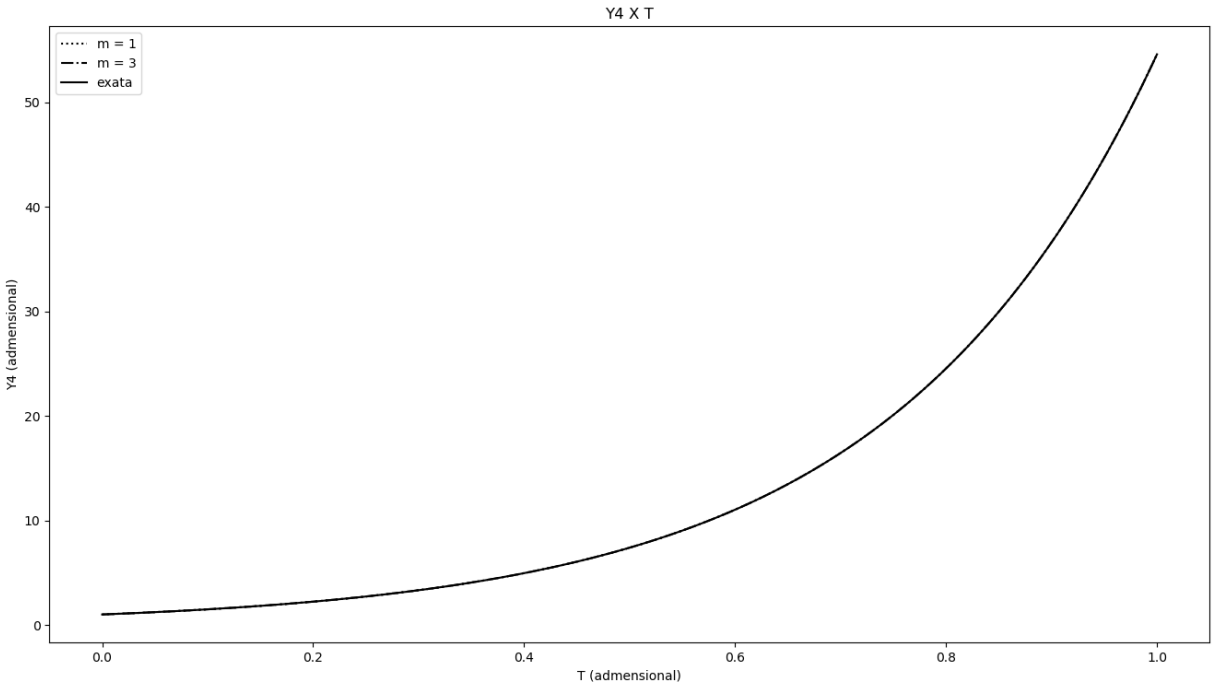

Figura 13: Aproximação de y_4 no intervalo de estudo.

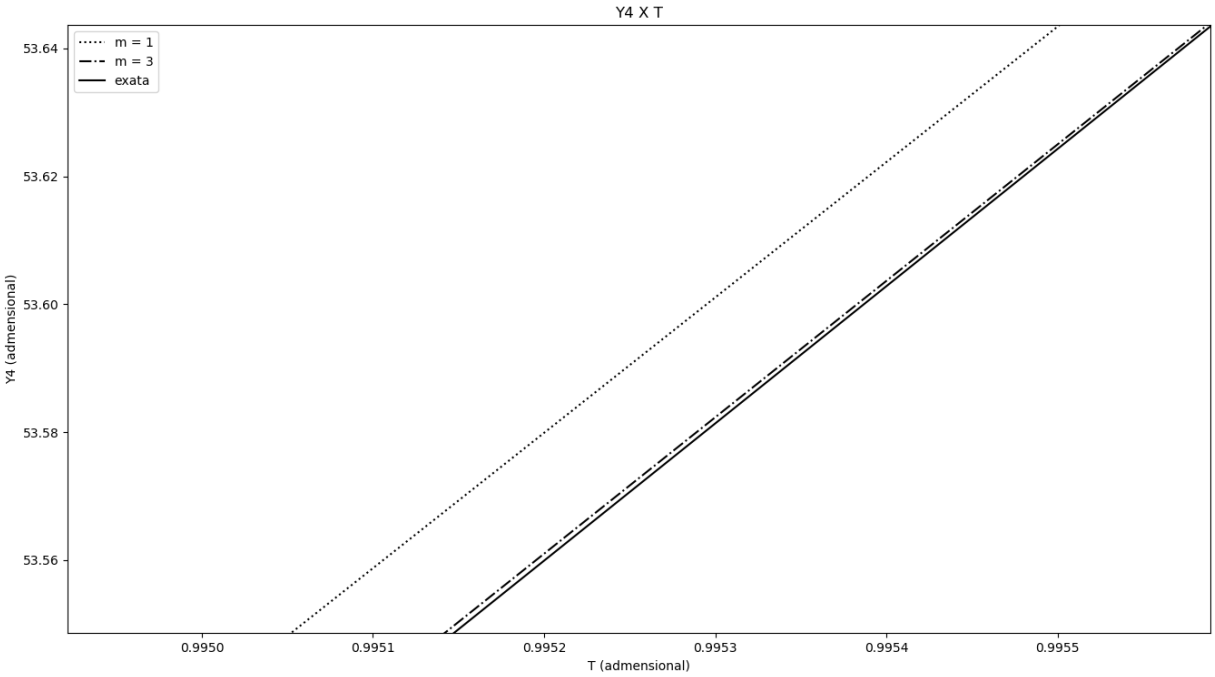

Figura 14: Aproximação de y_4 em um intervalo mais curto, evidenciando a convergência.

m	delta_t (m)	y (tf)	err = | e (tf , delta_t (m)) |	q = | err (m-1) / err (m) |	p'(m) = log 2 |q|
1	1.563E-02	5.45960E+01	2.11333E-03	-	-
2	7.813E-03	5.45979E+01	2.70847E-04	7.80266E+00	2.964
3	3.906E-03	5.45981E+01	3.42815E-05	7.90068E+00	2.982
4	1.953E-03	5.45981E+01	4.31204E-06	7.95017E+00	2.991
5	9.766E-04	5.45981E+01	5.40692E-07	7.97504E+00	2.995
6	4.883E-04	5.45981E+01	6.76919E-08	7.98754E+00	2.998
7	2.441E-04	5.45982E+01	8.46800E-09	7.99385E+00	2.999
8	1.221E-04	5.45982E+01	1.05870E-09	7.99848E+00	3.000
9	6.104E-05	5.45982E+01	1.32303E-10	8.00209E+00	3.000
10	3.052E-05	5.45982E+01	1.65983E-11	7.97089E+00	2.995

Figura 15: Tabela de convergência de y_4.

7 Aplicação do Método de Resolução

7.1 Visualização por Splines Cúbicos

Para aproximar o gráfico de computadores suscetíveis não-infectados usamos a biblioteca Scipy que implemente diferente ferramentas matemáticas e de engenharia.

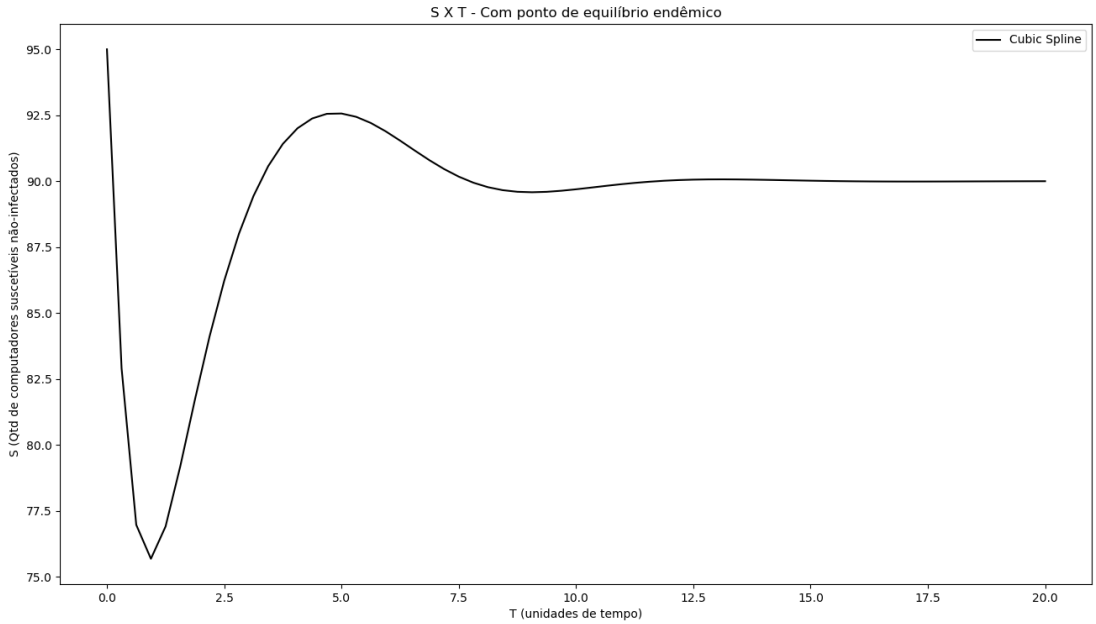

Figura 16: Aproximação por Splines Cúbicos para a variável de computadores suscetíveis não-infectados.

7.2 Análise e Discussão

Satisfatoriamente, os resultados da aproximação do problema convergiram para os resultados esperados de acordo com o material base (PIQUEIRA, 2009). As primeiras 5 Figuras correspondem ao calcular o ponto P1, vendo-o convergir corretamente para o ponto-endêmico definido pela teoria $P_1(0, 0, 0, T)$. Em sequência é possível ver as Figuras com as aproximações de P2 e P3 também convergindo para os valores teóricos.

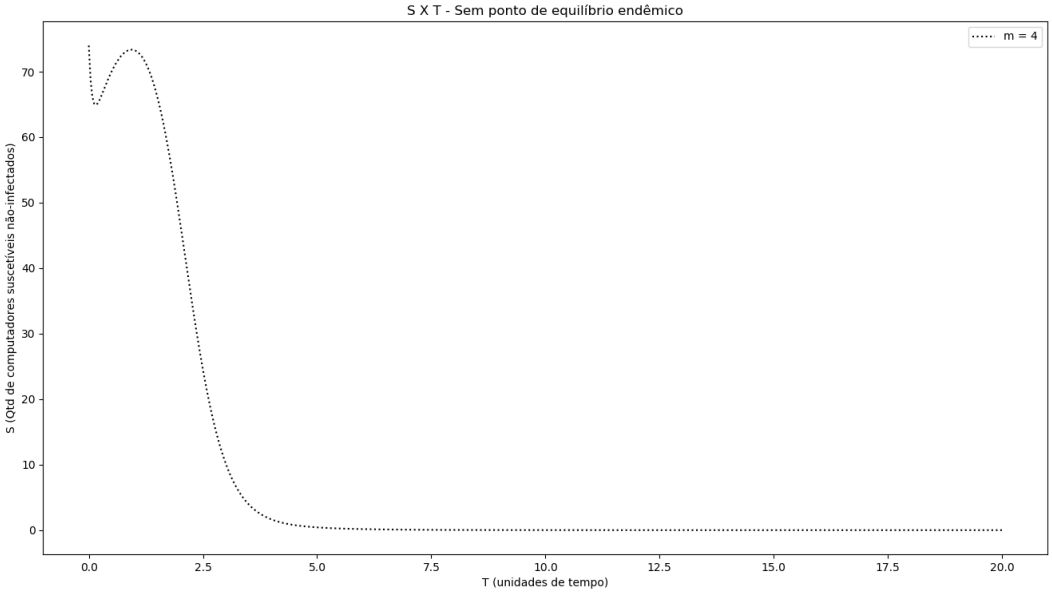

Figura 17: Gráfico da população suscetível no tempo, convergindo para P1.

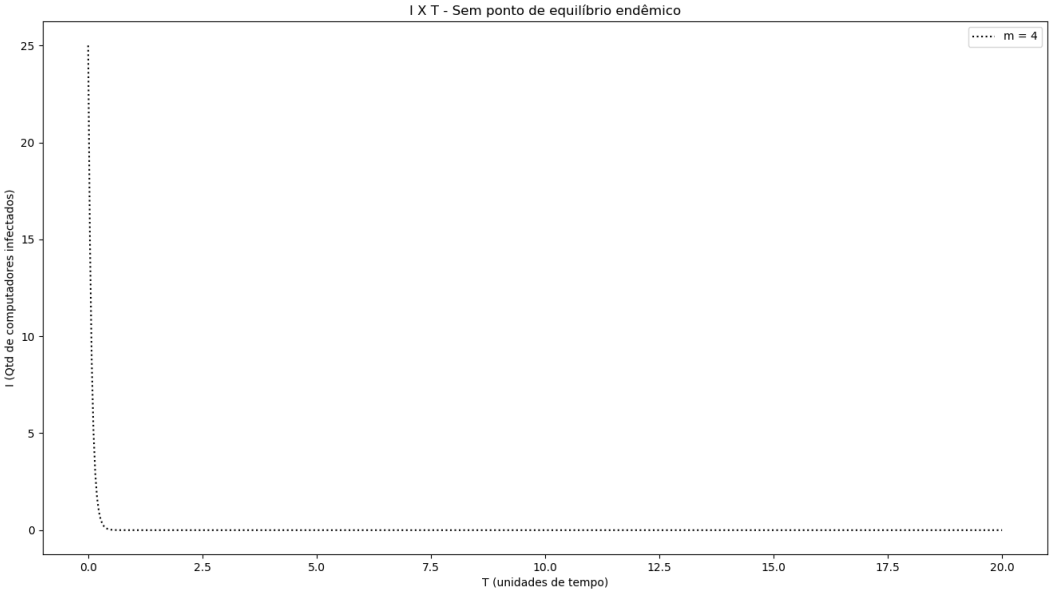

Figura 18: Gráfico da população infectada no tempo, convergindo para P1

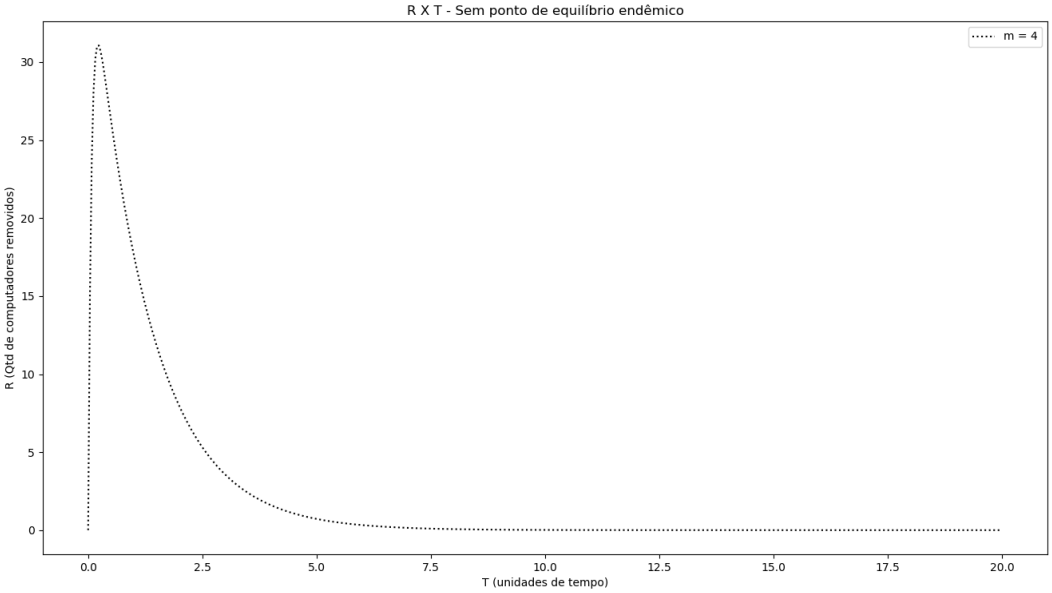

Figura 19: Gráfico da população removida no tempo, convergindo para P1.

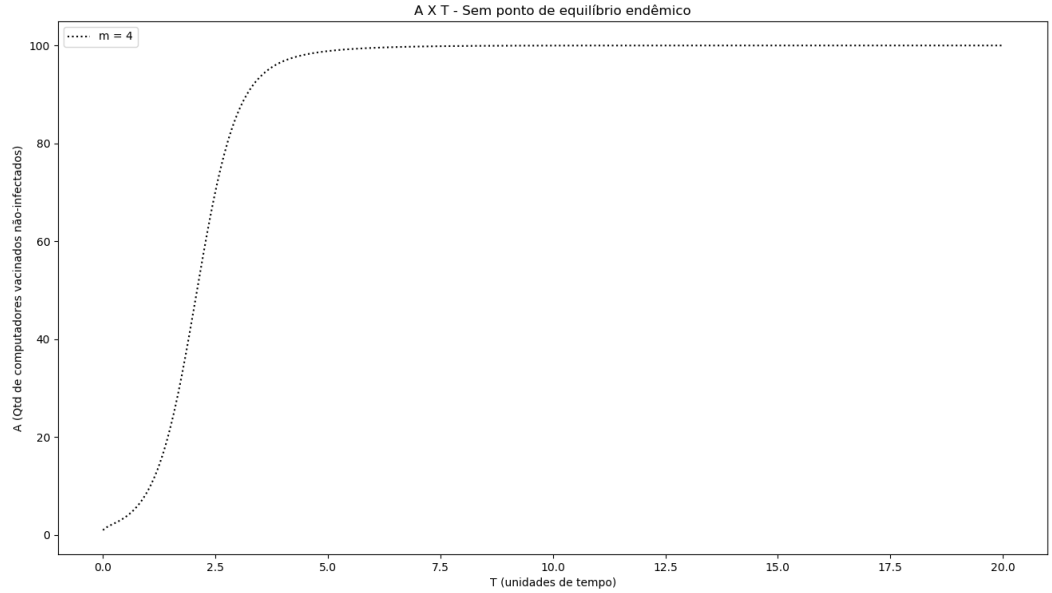

Figura 20: Gráfico da população antidotal no tempo, convergindo para P1.

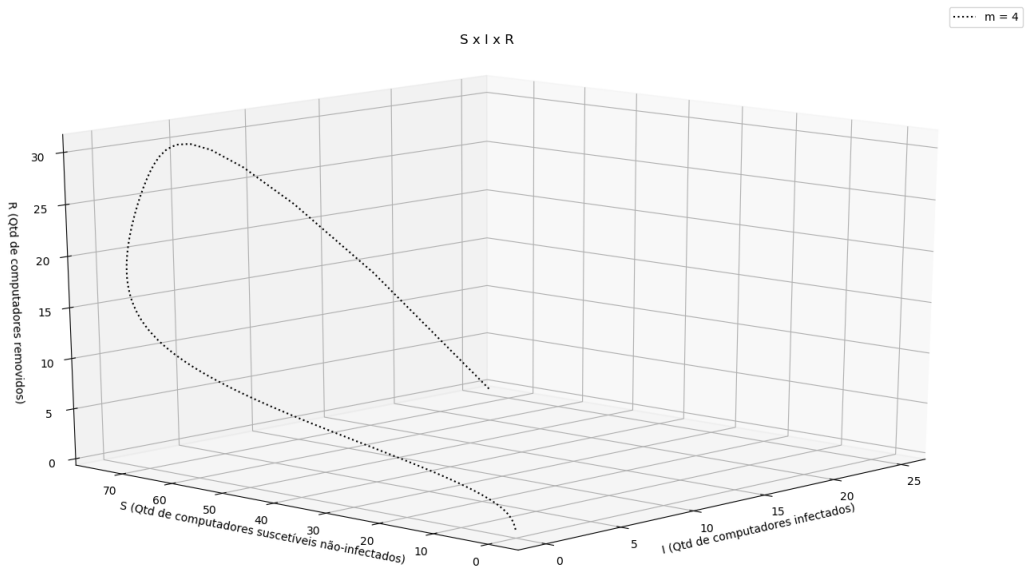

Figura 21: Gráfico das populações suscetível, infectada e antidotal no tempo, convergindo para P1.

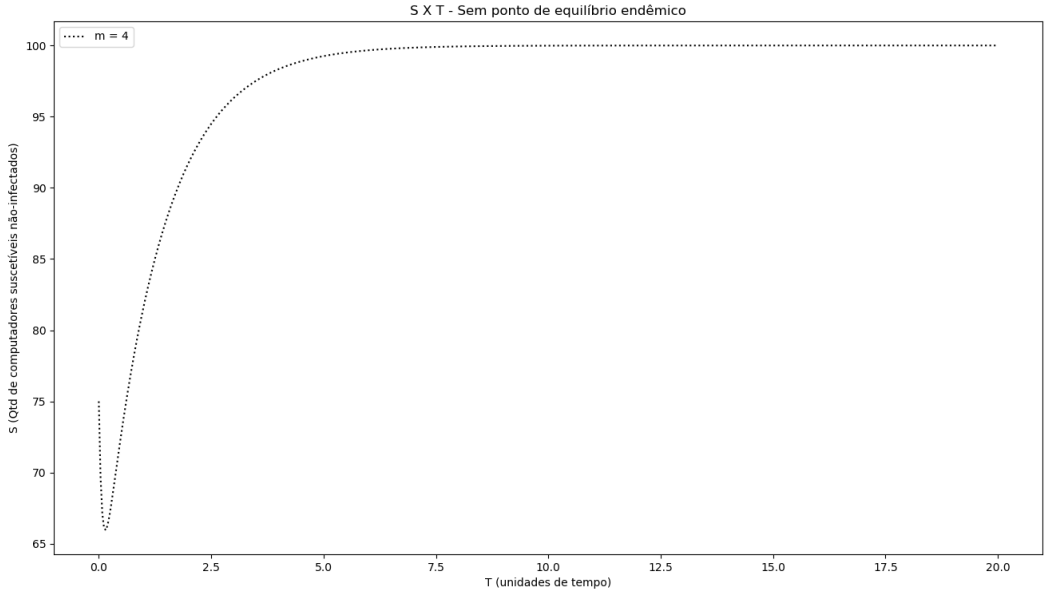

Figura 22: Gráfico da população suscetível no tempo, convergindo para P2.

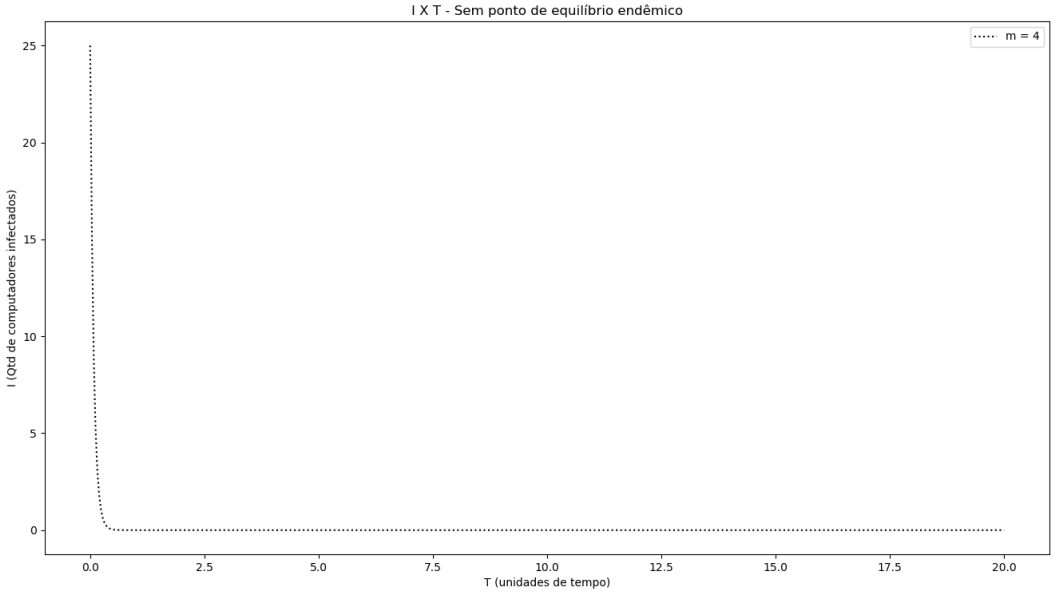

Figura 23: Gráfico da população infectada no tempo, convergindo para P2.

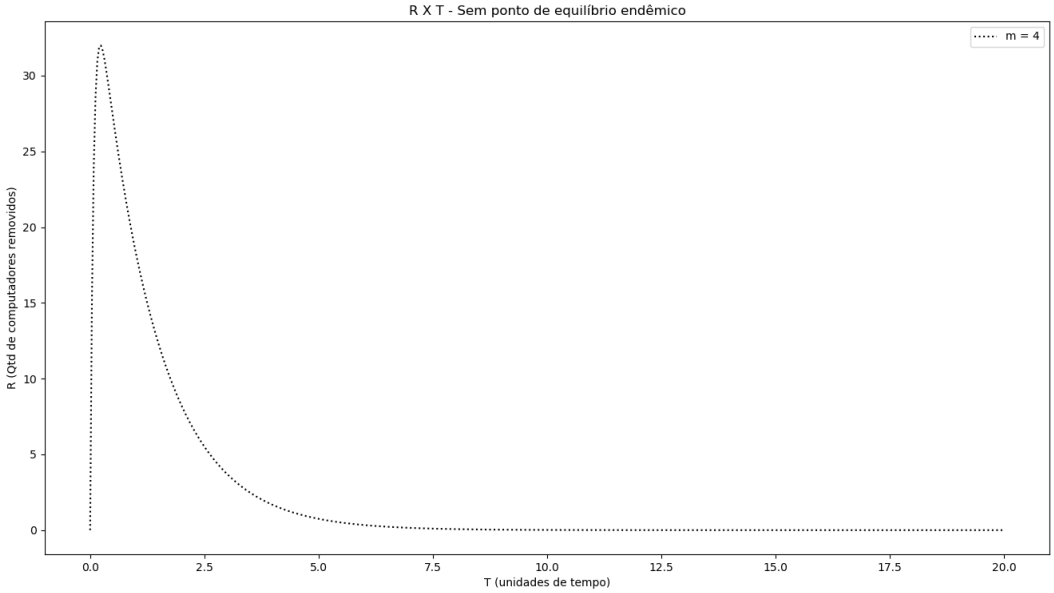

Figura 24: Gráfico da população removida no tempo, convergindo para P2.

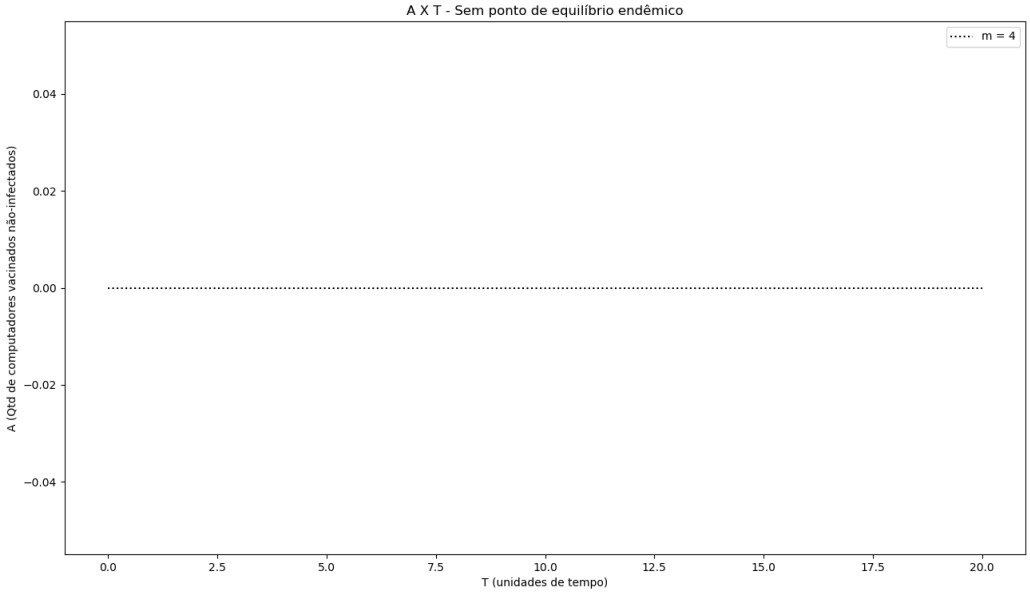

Figura 25: Gráfico da população antidotal no tempo, convergindo para P2.

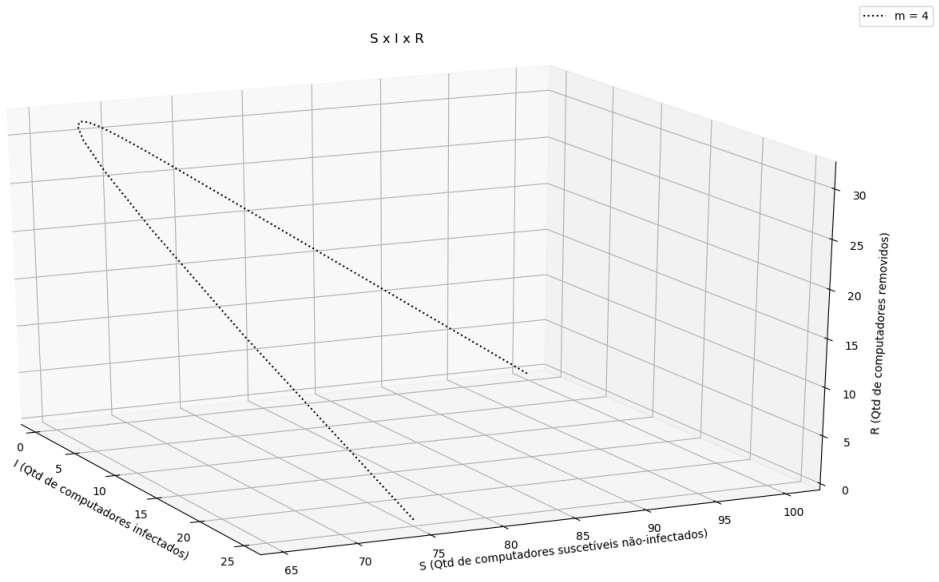

Figura 26: Gráfico das populações suscetível, infectada e antidotal no tempo, convergindo para P2.

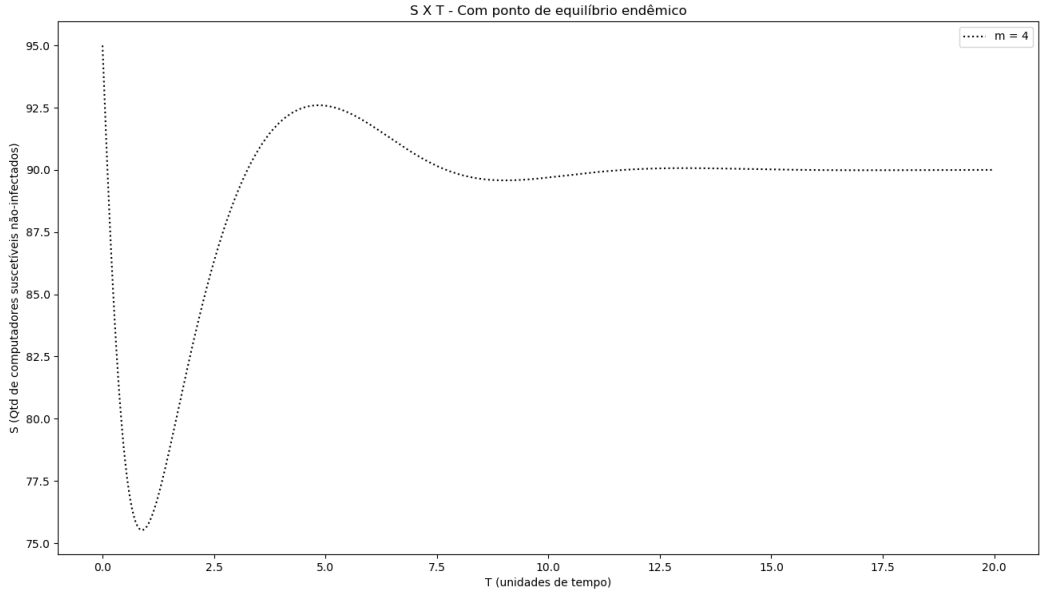

Figura 27: Gráfico da população suscetível no tempo, convergindo para P3.

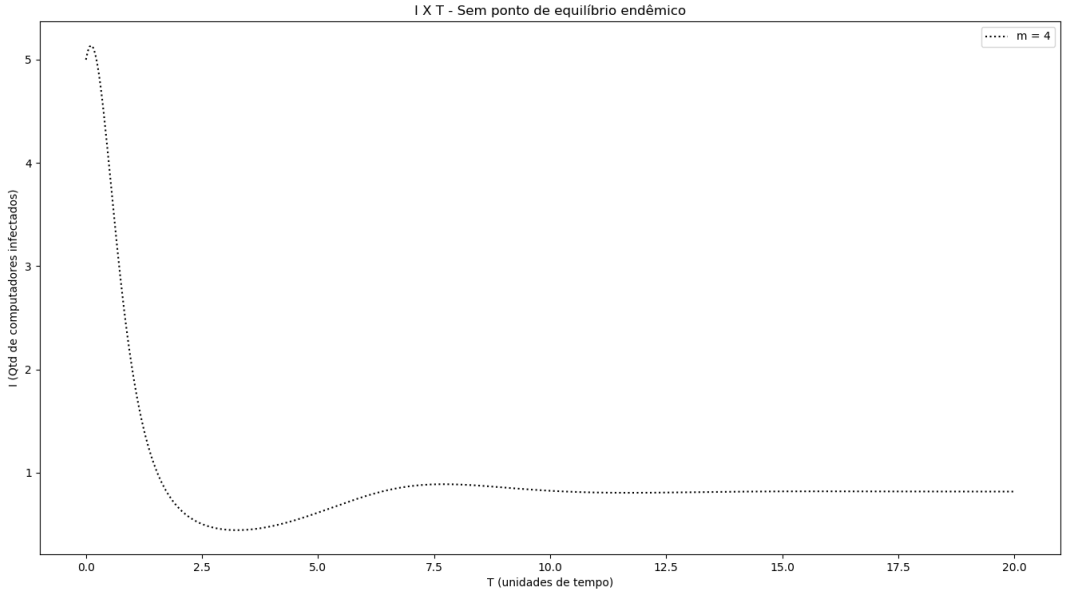

Figura 28: Gráfico da população infectada no tempo, convergindo para P3.

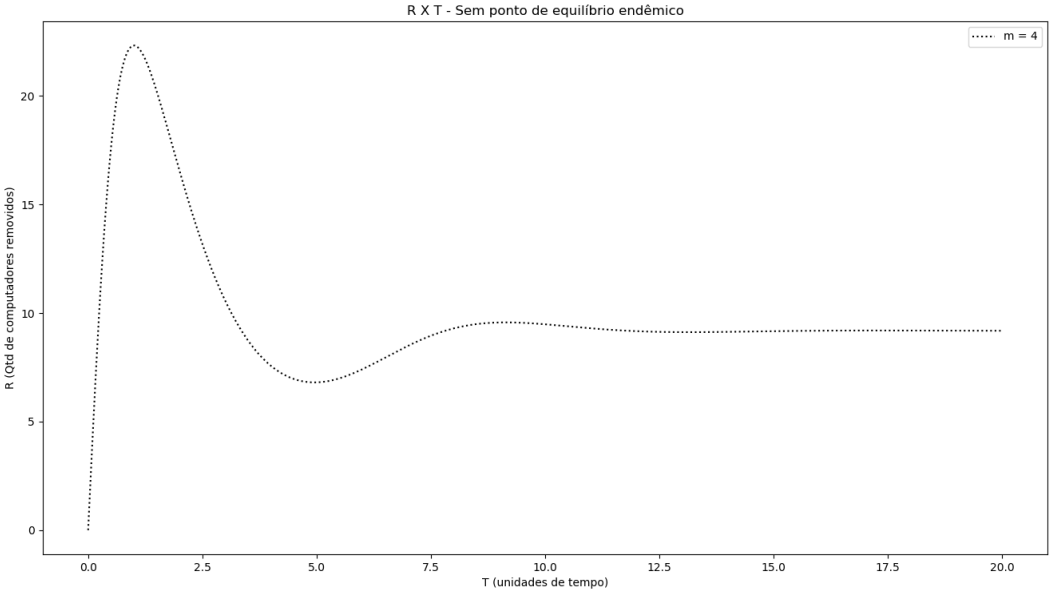

Figura 29: Gráfico da população removida no tempo, convergindo para P3.

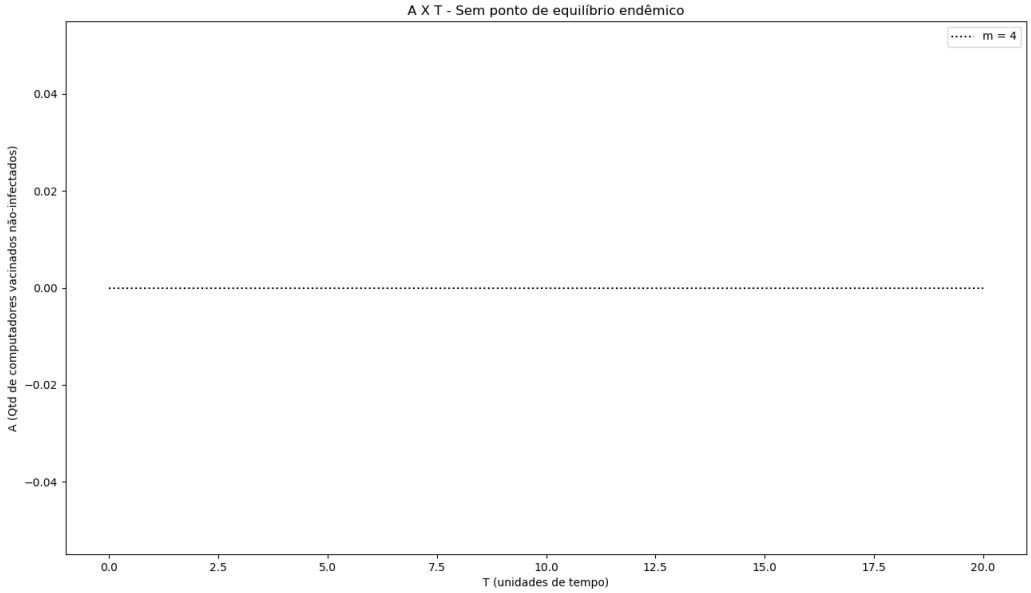

Figura 30: Gráfico da população antidotal no tempo, convergindo para P3.

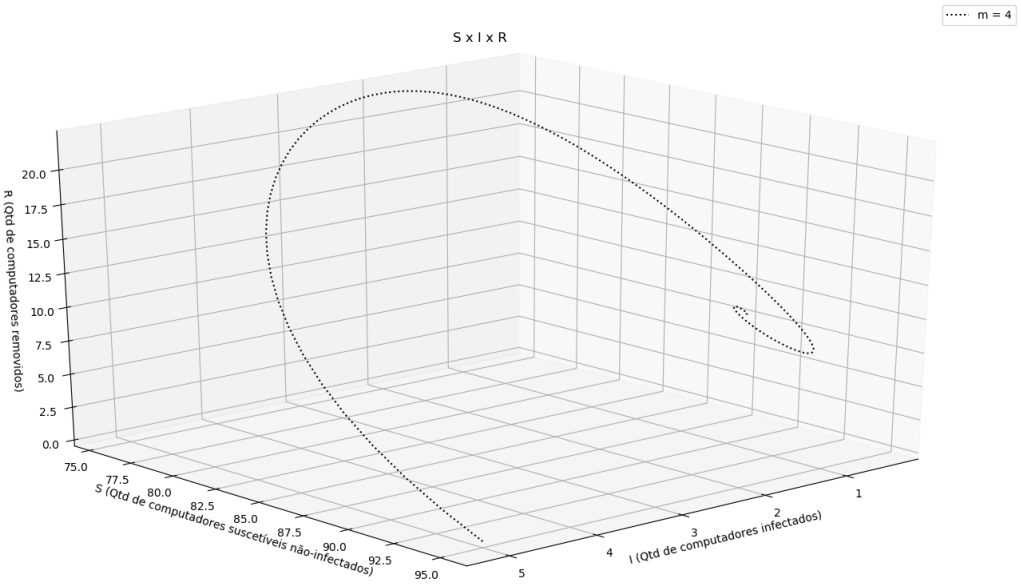

Figura 31: Gráfico das populações suscetível, infectada e antidotal no tempo, convergindo para P3.

Conclusão

Como evidenciado nas seções anteriores, este trabalho obteve um resultado satisfatório, validando a discretização do modelo e a implementação do método. Os resultados previstos se concretizaram e foram os mesmos apresentados pelo projeto base (PIQUEIRA, 2009). Entretanto, notamos uma divergência no último gráfico (Figura 7.2) onde nosso ponto final convergiu para o valor previsto pela Equação 2.4, enquanto o trabalho base obteve um ponto de convergência diferente. Supomos que foi um erro de edição, já que a legenda do gráfico indica que se trata de uma convergência pra P2, enquanto o texto que o referencia indica que se trata de uma convergência para P3.

Este projeto nos proporcionou desenvolver as técnicas adquiridas ao longo da disciplina, dando uma visão prática da aplicação de métodos numéricos e sua importância no mundo moderno. Hoje vemos a influência destas técnicas em diversas áreas do conhecimento, incluindo Finanças, Virologia, Física e Inteligência Artificial.

Referências

PIQUEIRA, V. A. J. A modified epidemiological model for computer viruses. *Appl. Math. Comput., 213 (2)*, 2009. Citado 4 vezes nas páginas 2, 16, 35 e 44.

Apêndices

APÊNDICE A – Implementação

```
"""
Referencias:
    - https://matplotlib.org/
    - https://docs.python.org/3/
    - https://rosettacode.org/wiki/Runge-Kutta_method#Alternate_solution
"""

import matplotlib.gridspec as gridspec
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from scipy.interpolate import CubicSpline
import math

#Classe CONSTRUTORDEGRAFICO
class ConstrutorDeGrafico:

    def __init__(self):
        self.fig = plt.figure()
        self.gs = gridspec.GridSpec(1, 1)

        #Grafico 2D
        self.eixo = self.fig.add_subplot(self.gs[0])
        #Grafico 3D
#        self.eixo = self.fig.add_subplot(self.gs[0], projection='3d')


    def adicionar(self, arquivo, m, tracejado):
        with open(arquivo) as ff:
            linhas = ff.readlines()
            T = [float(linha.split()[0]) for linha in linhas]
            Y1 = [float(linha.split()[1]) for linha in linhas] #S
            Y2 = [float(linha.split()[2]) for linha in linhas] #I
            Y3 = [float(linha.split()[3]) for linha in linhas] #R
            Y4 = [float(linha.split()[4]) for linha in linhas] #A
        ff.close
```

```
# Populao  de computadores removidos
#  computadores vacinados no-infectados

        #Grafico 3D
        #TROCAR PARA IMPRIMIR UM GRAFICO POR VEZ. SINCRONIZAR COM SOLUCAO EXATA
        self.__configurarEixo2D(self.eixo, 'S X T - Com ponto de equilbrio
            endmico', T, Y1, tracejado, m, 'S (Qtd de computadores suscetveis
            no-infectados)')
#        self.__configurarEixo2D(self.eixo, 'I X T - Sem ponto de equilbrio
    endmico', T, Y2, tracejado, m, 'I (Qtd de computadores infectados)')
#        self.__configurarEixo2D(self.eixo, 'R X T - Sem ponto de equilbrio
    endmico', T, Y3, tracejado, m, 'R (Qtd de computadores removidos)')
#        self.__configurarEixo2D(self.eixo, 'A X T - Sem ponto de equilbrio
    endmico', T, Y4, tracejado, m, 'A (Qtd de computadores vacinados
    no-infectados)')

        #Grafico 3D
#        self.__configurarEixo3D(self.eixo, 'S x I x R', Y1, Y2, Y3, tracejado,
    m)


    def __configurarEixo3D(self, eixo, titulo, S, I, R, tracejado, m):
        eixo.set_title(titulo)
        eixo.plot(I, S, R, tracejado, label='m = ' + str(m), color="black")
        eixo.set_xlabel('I (Qtd de computadores infectados)')
        eixo.set_ylabel('S (Qtd de computadores suscetveis no-infectados)')
        eixo.set_zlabel('R (Qtd de computadores removidos)')
        eixo.legend()

    def __configurarEixo2D(self, eixo, titulo, T, valores, tracejado, m,
        rotulo_y):
        cs = CubicSpline(T, valores)
        eixo.set_title(titulo)
        eixo.plot(T, cs(T), label='Cubic Spline', color="black")
#        eixo.plot(T, valores, tracejado, label='m = ' + str(m), color="black")
        eixo.set_ylabel(rotulo_y)
        eixo.set_xlabel('T (unidades de tempo)')
        eixo.legend()

    def adicionarSolucaoExata(self):
        with open('solucao_exata.txt') as ff:
```

```
            linhas = ff.readlines()
            Te = [float(linha.split()[0]) for linha in linhas]
            Ye = [float(linha.split()[1]) for linha in linhas]
        ff.close
        self.eixo.plot(Te, Ye, label='exata', color="black")
        self.eixo.legend()

    def mostrar(self):
        self.gs.tight_layout(self.fig)
#Fim da classe CONSTRUTORDEGRAFICO


#Classe SIMULADOR
class Simulador:

    def __init__(self, m):

        self.m = m
        with open('parametros_iniciais.txt') as f:
            self.t0 = float(f.readline())
            self.tf = float(f.readline())
            self.n = int(f.readline())

            self.y1_0 = float(f.readline()) #S
            self.y2_0 = float(f.readline()) #I
            self.y3_0 = float(f.readline()) #R
            self.y4_0 = float(f.readline()) #A

            self.npc = float(f.readline()) #novos comp. adicionados
            self.tm = float(f.readline()) #taxa de mortalidade
            self.isi = float(f.readline()) #int. suscept. infect.
            self.isv = float(f.readline()) #int. suscept. vac.
            self.iiv = float(f.readline()) #int. infec. vac.
            self.cpc = float(f.readline()) #comp. consertados
            self.crm = float(f.readline()) #comp. removidos

        f.closed
        self.__simular()


    def __simular(self):
        self.__metodoDePassoUnico()
```

```
def __metodoDePassoUnico(self):
    h = (self.tf - self.t0)/self.__calcularN()

    with open(self.nomeArquivoDeSaida(), 'w+') as f:
    #Lista dos resultados com C.I. adicionada
        T = [self.t0]
        Y1 = [self.y1_0]
        Y2 = [self.y2_0]
        Y3 = [self.y3_0]
        Y4 = [self.y4_0]
        f.write(str(self.t0) + ' ' + str(self.y1_0) + ' ' + str(self.y2_0)
            + ' ' + str(self.y3_0) + ' ' + str(self.y4_0) + '\n')

    #Metodo Explicito
        for i in range(0, self.__calcularN()):
          tk = T[i]
          t_k1 = tk + h
          T.append(t_k1)

          y1_k = Y1[i] #S
          y2_k = Y2[i] #I
          y3_k = Y3[i] #R
          y4_k = Y4[i] #A

          #Calculando k1 do RK33
          y1_rk1 = self.__RK33_k1(self.__y1Linha, tk, h, y1_k, y2_k, y3_k,
              y4_k) #S
          y2_rk1 = self.__RK33_k1(self.__y2Linha, tk, h, y1_k, y2_k, y3_k,
              y4_k) #I
          y3_rk1 = self.__RK33_k1(self.__y3Linha, tk, h, y1_k, y2_k, y3_k,
              y4_k) #R
          y4_rk1 = self.__RK33_k1(self.__y4Linha, tk, h, y1_k, y2_k, y3_k,
              y4_k) #A

          #Calculando k2 do RK33
          y1_rk2 = self.__RK33_k2(self.__y1Linha, tk, h, y1_k, y2_k, y3_k,
              y4_k, y1_rk1, y2_rk1, y3_rk1, y4_rk1) #S
          y2_rk2 = self.__RK33_k2(self.__y2Linha, tk, h, y1_k, y2_k, y3_k,
              y4_k, y1_rk1, y2_rk1, y3_rk1, y4_rk1) #I
```

```
            y3_rk2 = self.__RK33_k2(self.__y3Linha, tk, h, y1_k, y2_k, y3_k,
                y4_k, y1_rk1, y2_rk1, y3_rk1, y4_rk1) #R
            y4_rk2 = self.__RK33_k2(self.__y4Linha, tk, h, y1_k, y2_k, y3_k,
                y4_k, y1_rk1, y2_rk1, y3_rk1, y4_rk1) #A

            #Calculando k3 do RK33
            y1_rk3 = self.__RK33_k3(self.__y1Linha, tk, h, y1_k, y2_k, y3_k,
                y4_k, y1_rk1, y2_rk1, y3_rk1, y4_rk1, y1_rk2, y2_rk2, y3_rk2,
                y4_rk2) #S
            y2_rk3 = self.__RK33_k3(self.__y2Linha, tk, h, y1_k, y2_k, y3_k,
                y4_k, y1_rk1, y2_rk1, y3_rk1, y4_rk1, y1_rk2, y2_rk2, y3_rk2,
                y4_rk2) #I
            y3_rk3 = self.__RK33_k3(self.__y3Linha, tk, h, y1_k, y2_k, y3_k,
                y4_k, y1_rk1, y2_rk1, y3_rk1, y4_rk1, y1_rk2, y2_rk2, y3_rk2,
                y4_rk2) #R
            y4_rk3 = self.__RK33_k3(self.__y4Linha, tk, h, y1_k, y2_k, y3_k,
                y4_k, y1_rk1, y2_rk1, y3_rk1, y4_rk1, y1_rk2, y2_rk2, y3_rk2,
                y4_rk2) #A


            y1_k1 = y1_k + self.__RK33(y1_rk1, y1_rk2, y1_rk3) #S
            y2_k1 = y2_k + self.__RK33(y2_rk1, y2_rk2, y2_rk3) #I
            y3_k1 = y3_k + self.__RK33(y3_rk1, y3_rk2, y3_rk3) #R
            y4_k1 = y4_k + self.__RK33(y4_rk1, y4_rk2, y4_rk3) #A

            Y1.append(y1_k1) #S
            Y2.append(y2_k1) #I
            Y3.append(y3_k1) #R
            Y4.append(y4_k1) #A

            f.write(str(t_k1) + ' ' + str(y1_k1) + ' ' + str(y2_k1) + ' ' +
                str(y3_k1) + ' ' + str(y4_k1) + '\n')

    f.closed

def __calcularN(self):
    return self.n*(2**self.m)

def nomeArquivoDeSaida(self):
    return 'saida_' + str(self.m) + '.txt'
```

```
#RUNGE-KUTTA 3 ORDEM
def __RK33_k1(self, ylinha, tk, h, y1_k, y2_k, y3_k, y4_k):
    k1 = h * ylinha(tk, y1_k, y2_k, y3_k, y4_k)
    return k1

def __RK33_k2(self, ylinha, tk, h, y1_k, y2_k, y3_k, y4_k, y1_rk1, y2_rk1,
    y3_rk1, y4_rk1):
    y1_rk2_fator = (y1_rk1/2)
    y2_rk2_fator = (y2_rk1/2)
    y3_rk2_fator = (y3_rk1/2)
    y4_rk2_fator = (y4_rk1/2)
    k2 = h * ylinha(tk + h/2, y1_k + y1_rk2_fator, y2_k + y2_rk2_fator,
        y3_k + y3_rk2_fator, y4_k + y4_rk2_fator)
    return k2

def __RK33_k3(self, ylinha, tk, h, y1_k, y2_k, y3_k, y4_k, y1_rk1, y2_rk1,
    y3_rk1, y4_rk1, y1_rk2, y2_rk2, y3_rk2, y4_rk2):
    y1_rk3_fator = - y1_rk1 + (2*y1_rk2)
    y2_rk3_fator = - y2_rk1 + (2*y2_rk2)
    y3_rk3_fator = - y3_rk1 + (2*y3_rk2)
    y4_rk3_fator = - y4_rk1 + (2*y4_rk2)
    k3 = h * ylinha(tk + h, y1_k + y1_rk3_fator, y2_k + y2_rk3_fator, y3_k
        + y3_rk3_fator, y4_k + y4_rk3_fator)
    return k3

def __RK33(self, k1, k2, k3):
    return (k1 + 4 * k2 + k3)/6
#FIM do RUNGE-KUTTA 3 ORDEM

def __y1Linha(self, tk, y1_k, y2_k, y3_k, y4_k, ):
    return self.npc - (self.isv*y1_k*y4_k) - (self.isi*y1_k*y2_k) -
        (self.tm*y1_k) + (self.cpc*y3_k)

def __y2Linha(self, tk, y1_k, y2_k, y3_k, y4_k):
    return (self.isi*y1_k*y2_k) - (self.iiv*y4_k*y2_k) - (self.crm*y2_k)

def __y3Linha(self, tk, y1_k, y2_k, y3_k, y4_k):
    return (self.crm*y2_k) - (self.cpc*y3_k)

def __y4Linha(self, tk, y1_k, y2_k, y3_k, y4_k):
    return (self.isv*y1_k*y4_k) + (self.iiv*y4_k*y2_k)
```

```
#Fim da classe SIMULADOR


#Classe GERADORDESOLUCAOEXATA
class GeradorDeSolucaoExata:

    def __init__(self):
        with open('parametros_iniciais.txt') as f:
            self.t0 = float(f.readline())
            self.tf = float(f.readline())
            self.i = 2**10
        f.closed

        with open('solucao_exata.txt', 'w+') as f:
            h = (self.tf - self.t0)/self.i
            t = self.t0

            for k in range(0, self.i+1):
                t_k1 = t + h * k
#                y_k1 = self.__calcularSolucaoExata1(t_k1) #S
#                y_k1 = self.__calcularSolucaoExata2(t_k1) #I
#                y_k1 = self.__calcularSolucaoExata3(t_k1) #R
#                y_k1 = self.__calcularSolucaoExata4(t_k1) #A

                f.write(str(t_k1) + ' ' + str(y_k1) + '\n')
        f.closed

    def __calcularSolucaoExata1(self, tk):
        return ( math.e ** (1*tk) )

    def __calcularSolucaoExata2(self, tk):
        return ( math.e ** (2*tk) )

    def __calcularSolucaoExata3(self, tk):
        return ( math.e ** (3*tk) )

    def __calcularSolucaoExata4(self, tk):
        return ( math.e ** (4*tk) )

#Fim da classe GERADORDESOLUCAOEXATA
```

```
def main():

    sim_a = Simulador(1)
    sim_b = Simulador(2)
    sim_c = Simulador(3)
    sim_d = Simulador(4)
    sim_e = Simulador(5)
    sim_f = Simulador(6)
    sim_g = Simulador(7)
    sim_h = Simulador(8)
    sim_i = Simulador(9)
    sim_j = Simulador(10)

    construtorDeGrafico = ConstrutorDeGrafico()

    construtorDeGrafico.adicionar(sim_a.nomeArquivoDeSaida(), sim_a.m, '-.')
#    construtorDeGrafico.adicionar(sim_d.nomeArquivoDeSaida(), sim_d.m, ':')
#    construtorDeGrafico.adicionar(sim_e.nomeArquivoDeSaida(), sim_e.m, '--')

#    GeradorDeSolucaoExata()
#    construtorDeGrafico.adicionarSolucaoExata()

    construtorDeGrafico.mostrar()

main()
```