Equações de Friedmann

João Pedro Bassetti Freitas - 13903411
-joaopb7f10@usp.br
Carlos Maria Engler Pinto - 5104641
-carlos.engler@usp.br

12 de Abril de 2024

Sumário

1 Introdução **2**

2 Modelagem Matemática **3**
2.1 Dedução das equações de Friedmann 7
2.2 Problema de cauchy 8

3 Metodologia Numérica **8**
3.1 Método para Aproximação de Solução de EDO e Discretização das Equações de Friedmann 8
3.2 Algoritmo do processo de aproximação de solução 10

4 Resultados **11**
4.1 Verificação por meio de solução manufaturada 11
4.2 Resultados da aplicação às Equações de Friedmann 14
4.2.1 Sistema de unidades 15
4.2.2 Definição do período da simulação 15
4.2.3 Resultados numéricos obtidos 16

5 Conclusão **21**

6 Referências Bibliográficas **23**

7 Apêndices **24**

Este documento detalha as Equações de Friedmann, fundamentais para entender a expansão do universo na cosmologia. A análise começa com a derivação teórica dessas equações a partir das equações de campo de Einstein, enfatizando a importância do princípio cosmológico e da isotropia e homogeneidade do universo. O estudo utiliza métodos numéricos, especialmente o método de Runge-Kutta de quarta ordem, para resolver as equações e explorar o impacto de diferentes valores da constante cosmológica () na expansão do universo. O documento também apresenta técnicas de interpolação, como splines cúbicos, para melhor visualizar e entender as soluções numéricas obtidas. O trabalho conclui destacando a relevância das Equações de Friedmann na cosmologia moderna e sugere direções futuras para a pesquisa.

Palavras-chave: Equações de Friedman, Universo de Sitter, Spline cúbico, Método Runge-Kutta, expansão do universo

1 Introdução

As equações de Friedmann são um conjunto de duas equações diferenciais utilizadas para descrever características todoQuais características? da expansão do universo. Elas surgem como soluções das equações de campo de Einstein, que têm a forma:

$$G_{\mu\nu} - \Lambda g_{\mu\nu} = \kappa T_{\mu\nu}$$

Para contextualizar o problema, é necessário entender algumas variáveis. A primeira é o $G_{\mu\nu}$, conhecido como tensor de Einstein, representando uma matriz onde μ e ν são índices de posição. Em seguida, temos Λ, a constante cosmológica, interpretada como a densidade de energia no vácuo. O terceiro é $g_{\mu\nu}$, o tensor métrico, que define a geometria do espaço-tempo. Na outra ponta da equação, temos κ, a constante gravitacional de Einstein, e $T_{\mu\nu}$, o tensor de energia-momento, utilizado para compreender as propriedades físicas do universo.

Para obter as equações de Friedmann a partir da equação acima, é necessário considerar o universo como sendo isotrópico e homogêneo, ou seja, similar independentemente da direção ou do local considerado, em larga escala. Essas suposições são denominadas o *Princípio Cosmológico* e se traduzem matematicamente como sendo radialmente simétrico e composto, em larga escala, por matéria perfeitamente fluida.

Além disso, para obter a solução desejada, deve-se considerar os efeitos de dilatação temporal, definidos na teoria da relatividade de Einstein. Para isso, é utilizada a Métrica Friedmann–Lemaître–Robertson–Walker, que descreve o comportamento e curvatura do espaço-tempo:

$$ds^2 = g_{\mu\nu} dx^\mu dx^\nu = -c^2 dt^2 + a^2(t)\, d\mathbf{x}^2_{\text{espaço}}$$

Para uma compreensão mais aprofundada, é importante esclarecer essas variáveis em maior detalhe. Primeiramente, ds representa a variação infinitesimal da posição no espaço-tempo. A segunda equação refere-se a uma forma abreviada de representar o lado direito, onde $g_{\mu\nu}$ contém todas as informações pre-

sentes em ds^2. Por fim, na última equação, temos o fator de escala dependente do tempo $a(t)$, indicando a expansão ou contração do universo, e $\mathbf{x}^2_{\text{espaço}}$, que representa a variação infinitesimal no espaço. Sendo assim, obtemos as equações de Friedmann:

$$\boxed{\left(\frac{\dot{a}\left(t\right)}{a\left(t\right)}\right)^2 = \frac{8\pi G}{3}\rho\left(t\right) - \frac{\kappa c^2}{a^2\left(t\right)} + \frac{\Lambda c^2}{3}}$$

$$\boxed{\frac{\ddot{a}\left(t\right)}{a\left(t\right)} = -\frac{4\pi G}{3}\left(\rho\left(t\right) + \frac{3p\left(t\right)}{c^2}\right) + \frac{\Lambda c^2}{3}}$$

Figura 1: Equações de Friedmann

As equações de Friedmann são equações diferenciais cujas soluções são o fator de escala do crescimento do universo, dependentes do tempo. O fator de escala descreve como o crescimento do universo varia em função do tempo.

Esse conjunto de equações depende de três funções do tempo: $a(t)$ (fator de escala); $p(t)$ (pressão do modelo fluido do universo); $\rho(t)$ (densidade de energia do universo).

Por meio de uma equação de estado, podemos expressar a pressão em função da densidade de energia e assim possuir um sistema de duas equações para encontrar as funções representadas pelo fator de escala e a densidade de energia. Sendo assim, o objetivo será resolver a equação de Friedmann dado um contexto do universo a ser estudado, ou seja, uma condição inicial (representada pela constante de Hubble) e uma forma da equação de estado correspondente.

Com o intuito de entender os padrões de expansão cósmica, o trabalho se baseará na referência fornecida pelo artigo da Profound Physics. Este estudo resolverá as equações diferenciais de Friedmann para vários cenários cosmológicos, como por exemplo, o universo proposto por Willem De Sitter, caracterizado por uma densidade de massa constante e pressão nula, será analisado os comportamentos de expansão em diferentes universos e a influência da matéria negra todomatéria escura Além disso, serão apresentadas explicações e ilustrações gráficas dos fenômenos de "Big Crunch"e "Heat Death", revelando os momentos cruciais desses eventos no universo.

2 Modelagem Matemática

As equações de Friedmann são, assim como mencionado, equações utilizadas para descrever a forma como o universo se expande considerando diversas hipóteses sobre sua forma e seus aspectos qualitativos.

O primeiro passo para a obtenção das equações de Friedmann são as considerações feitas por Einstein na Teoria da Relatividade Geral. Mais especificamente, a Relatividade Geral concluiu uma descrição do funcionamento da gravidade, afirmando que o universo é definido por espaço-tempo, ou seja, um espaço quadridimensional definido em conjunto com o tempo.

A grande conclusão de Einstein foi a de que o a gravidade é uma característica geométrica do espaço-tempo, e relacionada à descrição da influência da massa sobre o espaço quadridimensional descrito, que causa uma deformação. Essa relação entre a distribuição de massa do universo e sua curvatura são definidas, como desfecho da Teoria de Einstein, nas Equações de Campo de Einstein, que descrevem um sistema de equações diferenciais de segunda ordem:

$$G_{\mu\nu} - \Lambda g_{\mu\nu} = \kappa T_{\mu\nu}$$

Essa equação é descrita como uma equação de tensor, em que um tensor é uma defiição matemática utilizada para descrever uma relação multilinear entre objetos algébricos associados a um espaço vetorial (que pode ser pensado como uma generalização de uma matriz ou vetor em n dimensões). Descrevendo a equação: `G` é denominado o Tensor de Einstein, que é utilizado para descrever a curvatura do espaço tempo de maneira consistente com a conservação de energia e momento; `g` é denominado o Tensor métrico, que é utilizado para definir toda a geometria do espaço tempo, incluindo tempo, distância, curvatura e ângulos; `T` é o Tensor de Energia-momento, que descreve a densidade e o fluxo de momento e energia do espaço-tempo; Λ é a constante cosmológica, que pode ser interpretada como a densidade de energia no vácuo; κ, por fim, é a constante gravitacional de Einstein, que é definida como $\frac{8\pi G}{c^4}$.

Uma maneira mnemônica de encarar o resultado descrito pelas equações de campo de Einstein pode ser decrito à partir da fala do físico téorico John Archibald Wheeler:

> "O espaço-tempo diz como a matéria se move, enquanto a matéria diz como o espaço-tempo se deforma"

E é à partir dessa interpretação que as equações de campo de Einstein podem ser manipuladas e solucionadas para obter as equações de Friedmann.

As considerações necessárias para encontrar as equações de Friedmann são denominadas o Princípio Cosmológico, que são:

1. O universo é homogêneo, ou seja, é parecido em todos os pontos no espaço e é desprovido de um centro espacia.

2. O universo é isotrópico, ou seja, é parecido em todas as direções, independentemente de rotações espaciais.

À priori, pode-se chegar à conclusão de que essas hipóteses são falsas, já que numa escala interplanetária nenhuma das hipóteses é concretizada. Porém, a sua formulação é realizada levando em consideração uma macroescala do universo, de dezenas de milhões de anos luz, ou seja, o princípio cosmológico seria verdadeiro em média nas maiores escalas.

Os formatos do universo definidos pelo princípio cosmológico são de três maneiras: Plano (sem curvaturas, ou seja, é um universo perfeitamente euclidiano), aberto (com curvatura negativa, ou seja, um universo com curvatura hiperbólica) e fechado (com curvatura positiva, ou seja, com curvatura esférica). Apesar de haverem formatos mais complexos que sastisfazem as condições descritas pelo princípio, como um toróide, as principais análises feitas para a obtenção das equações de Friedmann levam em consideração universos simplesmente conectados.

A descrição física desses três formatos são realizadas à partir do tensor métrico descrito anteriormente. Cada formato é descrito matematicamente, utilizando descrições polares e hipérbólicas para as suas respectivas formas e as generalizando para n dimensões. Após isso, os vetores da base que descrevem cada formato definido são calculados e expressos no tensor métrico seguindo as necessidades do princípio cosmológico, obtendo-se:

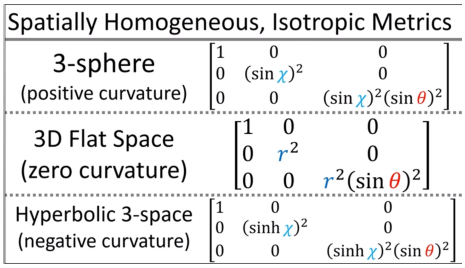

Figura 2: Tensores métricos para as formas simplesmente conexas isotrópicas

À partir dessa modelagem matemática, é possível obter a métrica Friedmann–Lemaître–Robertson–Walker (métrica FLRW).Essa métrica consiste em uma maneira única de expressar, de forma generalizada, as três métricas explicitadas incluindo uma variação de formato no tempo denominada $a(t)$, que será posteriormente uma das variáveis de estado responsáveis por diretamente descrever a forma como o universo se expande. A métrica FLRW pode ser expressa em sua forma tensorial pela figura 3:

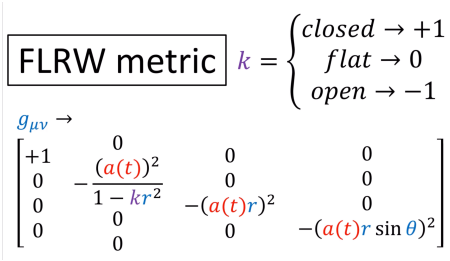

Figura 3: Métrica FLRW

A métrica FLRW é uma descrição generalista do formato do universo, assim como mencionado. Para o processo de obtenção das equações de Friedmann, essa métrica, em conjunto com os outros tensores das equações de campo de Einstein, são utilizadas como as soluções da equação.

Assim como a métrica LFRW, o tensor de Einstein também é definido no contexto do princípio cosmológico e é definido para a obtencção da solução das equações de campo de Einstein que levam às equações de Friedmann como solução.

Outro ponto muito importante para a dedução das equações de Friedmann é a dedução e o entendimento do Tensor de Energia-Momento, que consiste em um fluido com densidade de massa e pressão uniforme. O Tensor de Energia-Momento pode ser representado pela combinação linear dos parâmetros ρ (densidade de massa) e p (pressão uniforme).

Uma forma comum de visualizar o tensor de energia-momento é pela seguinte normalização:

$T_{\alpha\beta}$ = fluxo de momento-α percorrendo uma caixa com unidade constante β

Para entender melhor a nomenclatura é preciso pensar no seguinte exemplo: Em uma caixa de valor constante β, com eixo de referência quadridimensional, sendo os vetores x, y, z e t. É possivel calcular o tensor de energia-momento para um momento determinado, resultando nas seguintes equações:

$$\frac{P^t}{\Delta ct \Delta y \Delta z} = T^{tx}$$

$$\frac{P^x}{\Delta ct \Delta y \Delta z} = T^{xx}$$

$$\frac{P^y}{\Delta ct \Delta y \Delta z} = T^{yx}$$

$$\frac{P^z}{\Delta ct \Delta y \Delta z} = T^{zx}$$

Figura 4: Equações do Tensor Energia-Momento

Assim, com os tensores encontrados fazendo todas as combinação possíveis, pode-se montar uma matriz 4x4, como mostra a seguinte figura:

$$\begin{bmatrix} T^{tt} & T^{tx} & T^{ty} & T^{tz} \\ T^{xt} & T^{xx} & T^{xy} & T^{xz} \\ T^{yt} & T^{yx} & T^{yy} & T^{yz} \\ T^{zt} & T^{zx} & T^{zy} & T^{zz} \end{bmatrix}$$

Figura 5: Matriz 4x4 do Tensor Energia-Momento

A partir da matriz montada conseguiu-se montar algumas relações entre valores, indicando características semelhantes. Assim, as relações encontradas foram: a azul que indica a densidade da energia, a verde que indica a energia de fluxo, a vermelha que mostra o momento densidade, a roxa que tem relação com a pressão e por fim a amarela que indica a tensão de cisalhamento do material.

Logo, agora é preciso encontrar o tensor de energia-momento que se adequa às condições que provam a equação de Friedmann. Dessa forma, será considerando um fluido perfetio, que é um fluido com partículas com pressão constante e de mesma densidade, resultando na seguinte tabela:

$$T_{\alpha\beta} = \begin{vmatrix} \rho c^2 & 0 & 0 & 0 \\ 0 & -p & 0 & 0 \\ 0 & 0 & -p & 0 \\ 0 & 0 & 0 & -p \end{vmatrix}$$

2.1 Dedução das equações de Friedmann

Para deduzir a primeira equação de Friedmann, é necessário fazer a substituição dos componentes matricias no tipo-00 na fórmula de Einstein. Dessa forma, substituindo a constante gravitacional de Einstein em função de constantes conhecidas, chega-se na seguinte fórmula:

$$G_{00} - \Lambda g_{00} = \frac{8\pi G}{c^4} T_{00}$$

As componentes matriciais no tipo-00, de acordo com as demonstrações feitas anteriormente, assumem os seguintes valores:

- $G_{00} = \frac{3(\dot{a}^2 + kc^2)}{a^2}$;
- $T_{00} = \rho c^2$;
- $g_{00} = +1$;

Substituindo os valores encontrados, obtêm-se a formula:

$$\frac{3(\dot{a}^2 + kc^2)}{a^2} - \Lambda = \frac{8\pi G\rho}{c^2}$$

Fazendo a álgebra necessária, obtêm-se a primeira equação de Friedmann:

$$\left(\frac{\dot{a}}{a}\right)^2 = \frac{8\pi G\rho}{3} + \frac{\Lambda c^2}{3} - \frac{kc^2}{a^2} \tag{I}$$

Agora, para obtermos a segunda equação, deve-se multiplicar a fórmula de Einstein por $g^{\mu\nu}$, que representa o inverso do tensor métrico.

As componentes matriciais multiplicadas por $g^{\mu\nu}$, de acordo com as demonstrações feitas anteriormente, assumem os seguintes valores:

- $G_{\mu\nu}g^{\mu\nu} = \frac{6(a\ddot{a} + \dot{a}^2 + kc^2)}{c^2 a^2}$;
- $T_{\mu\nu}g^{\mu\nu} = \rho c^2 - 3p$;
- $g_{\mu\nu}g^{\mu\nu} = +4$;

Substituindo os valores encontrados, obtêm-se a formula:

$$\frac{\ddot{a}}{a} + \left(\frac{\dot{a}}{a}\right)^2 + \frac{kc^2}{a^2} - \frac{2\Lambda c^2}{3} = \frac{4\pi G(\rho - \frac{3p}{c^2})}{3} \tag{A}$$

A partir da primeira equação de Friedmann, pode-se encontrar a seguinte relação:

$$\left(\frac{\dot{a}}{a}\right)^2 + \frac{kc^2}{a^2} = \frac{8\pi G\rho}{3} + \frac{\Lambda c^2}{3} \tag{B}$$

Substituindo a equação B na equação A e fazendo a álgebra necessária, obtêm-se a segunda equação de Friedmann:

$$\frac{\ddot{a}}{a} = -\frac{4\pi G(\rho - \frac{3p}{c^2})}{3} + \frac{\Lambda c^2}{3} \tag{II}$$

2.2 Problema de cauchy

Para transformar um problema em um problema de Cauchy, é necessário encontrar condições iniciais nulas e definir um intervalo de tempo.

As equações de Friedmann possuem várias interpretações, desde as interpretações de Einstein, as de De Sitter, entre outras. Uma das interpretações a serem analisadas neste relatório é o Universo de De Sitter, que consiste em um universo dominado totalmente pela matéria escura, ou seja, um universo de espaços vazios e com uma aceleração de expansão positiva.

Com base nisso, podemos definir condições iniciais para as equações de Friedmann, onde a constante cosmológica (Λ) é sempre positiva, a densidade de massa (ρ) e a pressão constante (p) são ambas iguais a zero. O intervalo de tempo a ser considerado é de 0 até o infinito, e o problema será analisado para os três tipos diferentes de geometria do universo: plano, hiperbólico e esférico.

Dessa forma, chega-se no seguinte problema de Cauchy:

$$\begin{cases} \left(\frac{\dot{a}(t)}{a(t)}\right)^2 = \frac{\Lambda c^2}{3} - \frac{kc^2}{a^2} \quad \text{(EDOI)} \quad \text{em que } \Lambda > 0 \text{ para } t \in [0; +\infty[\\ \\ \frac{\ddot{a}(t)}{a(t)} = +\frac{\Lambda c^2}{3} \quad \text{(EDOII)} \quad \text{em que } \Lambda > 0 \text{ para } t \in [0; +\infty[\\ \\ k = -1,\ 0,\ 1; \\ \rho = p = 0; \end{cases}$$

3 Metodologia Numérica

Nesta seção, detalharemos os métodos numéricos que serão utilizados para resolver e aproximar as soluções do sistema de EDOs determinado pelas equações de Friedmann, além de apresentar a discretização do domínio da solução.

3.1 Método para Aproximação de Solução de EDO e Discretização das Equações de Friedmann

Para se determinar uma aproximação da solução do sistema de equações diferenciais ordinárias, pode-se usar os métodos de Runge-Kutta.

Os métodos de Runge-Kutta consistem em uma generalização de métodos numéricos mais simples, ou seja, métodos de passo único, como o método de Euler. A distinção entre os métodos reside na maneira como as funções de incremento são determinadas. Enquanto o método de Euler utiliza a função de incremento do passo como sendo a derivada determinada no início do intervalo,

os métodos de Runge-Kutta, por sua vez, avaliam derivadas em pontos intermediários do passo e as associam por meio de pesos diferentes, gerando uma aproximação mais precisa à solução desejada.

O método de Runge-Kutta explícito é definido pela seguinte expressão:

$$y_{k+1} = y_k + h * \phi(x_k, y_k, h) \tag{1}$$

Tal que:

$$\Phi(t, y, h) = \sum_{r=1}^{R} c_r \kappa_r(t, y, h) \tag{2}$$

Que consiste na soma das expressões κ multiplicadas por seus respectivos coeficientes, até a ordem R. Os valores κ são definidos da seguinte forma:

$$\begin{cases} \kappa_1(t,y,h) & = f(t,y) \\ \kappa_2(t,y,h) & = f\left(t + \frac{h}{2}, y + \frac{h}{2}\beta_{21}\kappa_1\right) \\ \kappa_3(t,y,h) & = f\left(t + \frac{h}{2}, y + h\beta_{31}\kappa_1 + h\beta_{32}\kappa_2\right) \\ & \vdots \\ \kappa_r(t,y,h) & = f\left(t + h\alpha_r, y + h\sum_{s=1}^{r-1}\beta_{rs}\kappa_s\right), \quad 2 \leq r \leq R. \end{cases}$$

Em sua essência, a expressão κ_r é definida a partir do cálculo de derivadas e de suas somas sobre $_{r-1}$, tornando a função de incremento, no final, uma média ponderada da variação da função nos pontos intermediários do intervalo. Cada κ_r representa uma inclinação (definida pela variação da função original em pontos diferentes), e a taxa de incremento final adota o valor da média dessas inclinações para a variação do valor da função no intervalo.

A determinação dos coeficientes pode ser feita a partir da comparação da expressão de Runge-Kutta explícito com a expansão obtida pela série de Taylor. E para a aproximação das soluções do sistema de equações definido pelas equações de Friedmann, podemos utilizar o Método de Runge-Kutta de quarta ordem e de 4 estágios (RK-44), que é definido pelas seguintes expressões:

$$\begin{cases} \Phi(t,y,h) & = \frac{1}{6}(\kappa_1 + 2\kappa_2 + 2\kappa_3 + \kappa_4) \\ \kappa_1 & = f(t,y) \\ \kappa_2 & = f\left(t + \frac{h}{2}, y + \frac{h}{2}\kappa_1\right) \\ \kappa_3 & = f\left(t + \frac{h}{2}, y + \frac{h}{2}\kappa_2\right) \\ \kappa_4 & = f\left(t + h, y + h\kappa_3\right) \end{cases}$$

A partir dessa definição do método RK-44, podemos aplicá-lo no contexto de um sistema de equações com múltiplas variáveis. No contexto das equações de Friedmann, podemos expressar o sistema de EDOs como sendo duas equações de primeira ordem. Para isso, podemos reescrever a segunda equação de Friedmann da seguinte maneira:

$$\begin{cases} \dot{a} = v \\ \dot{v} = \ddot{a} = \frac{\Lambda c^2}{3} a \end{cases}$$

À partir dessa substituição de variáveis, podemos modelar os métodos numéricos para resolver a EDO de segunda ordem representada pela segunda equação

de Friedmann. Sendo assim, a equação de iteração para a obtenção dos valores de `v` e de `a` será:

$$\begin{cases} a(t+h) = a(t) + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ v(t+h) = v(t) + \frac{h}{6}(l_1 + 2l_2 + 2l_3 + l_4) \end{cases}$$

Tal que:

$$\begin{aligned} k_1 &= v & l_1 &= 3\Lambda c^2 a \\ k_2 &= v + \frac{h}{2} l_1 & l_2 &= 3\Lambda c^2 \left(a + \frac{h}{2} k_1\right) \\ k_3 &= v + \frac{h}{2} l_2 & l_3 &= 3\Lambda c^2 \left(a + \frac{h}{2} k_2\right) \\ k_4 &= v + h l_3 & l_4 &= 3\Lambda c^2 \left(a + h k_3\right) \end{aligned}$$

O cálculo de κ_a e l_v é feito de forma recursiva, ou seja, para obter o próximo κ_a, deve-se ter os respectivos κ_{a-1} e l_{v-1}.

A discretização do domínio do tempo, no contexto de estudo do comportamento do universo, varia dependendo das unidades e valor de constantes utilizadas. Para determinar uma quantidade de tempo cujo comportamento do fator de escala a seja considerável, precisamos parametrizá-lo à partir dos componentes da constante cosmológica associados à solução do universo de De Sitter estudado. Portanto, a discretização do domínio do tempo será explicitada dentro do contexto de cada simulação realizada.

3.2 Algoritmo do processo de aproximação de solução

O processo para a determinação da solução do problema de Cauchy apresentado, utilizando o método de Runge-Kutta, pode ser definido pelos seguintes passos:

- Definição de parâmetros e das condições iniciais (como definição das constantes, valor do período entre os pontos discretizados e das condições iniciais do sistema).
- Inicia-se o laço do programa principal, e nele calculam-se todos os κ_a e λ_H a cada passo da iteração.
- Ainda dentro desse loop, a cada iteração calcula-se o novo valor de `a(tk)` e `H(tk)`.
- Para obter valores intermediários entre os pontos da solução encontrados pelo método de Runge-Kutta, pode-se usar splines cúbicos para interpolar a função. Ou seja, dentro do loop principal, há outro loop para calcular valores intermediários entre os pontos da solução.
- Dentro do loop de cálculo dos splines, os coeficientes da expressão cúbica devem ser determinados. Para isso, pode-se usar o método de resolução de sistemas do escalonamento (que vai demandar outro laço) para resolver o sistema de equações que determinam os coeficientes da expressão desejada.

4 Resultados

Para verificar a validade do uso dos métodos sugeridos, o algorítmo será testado por uma solução manufaturada, onde serão analisados os comportamentos obtidos e comparados às curvas, taxas de convergência e erro do método.

4.1 Verificação por meio de solução manufaturada

A solução manufaturada escolhida para concluir a validade da implementação dos métodos propostos foi a seguinte equação diferencial ordinária de primeira ordem:

$$\begin{aligned} y' &= y + x, \\ y(0) &= 1. \end{aligned} \tag{3}$$

Por ser uma EDO, pode-se analiticamente obter a sua solução de maneira analítica. O resultado da equação é:

$$y = 2e^x - x - 1 \tag{4}$$

A escolha dessa solução manufaturada foi feita baseada na tendência similar à da solução do fator de expansão do universo de De Sitter. Como a expectativa para uma solução das equações de Friedmann no contexto do universo de De Sitter levam à uma tendência de crescimento exponencial, o funcionamento e eficácia dos métodos numéricos poderão ser vistos no contexto similar ao que serão submetidos para solucionar o problema escolhido.

Submetendo a forma padrão da equação diferencial escolhida para a solução manufaturada, como demonstrado nas seguintes linhas de código, foi-se possível aplicar o método de Runge-Kutta (RK-44) para aproximação dos pontos da solução em comparação à solução real:

```
1    def f(t, y):
2        return t + y
3
4    def phi(t, y, dt, f):
5        k1 = f(t, y)
6        k2 = f(t + dt/2, y + dt/2*k1)
7        k3 = f(t + dt/2, y + dt/2*k2)
8        k4 = f(t + dt, y + dt*k3)
9        return 1/6*(k1 + 2*k2 + 2*k3 + k4)
```

À partir da definição da função $f(t, y)$, e da função de crescimento, podemos implementá-los em um laço entre o intervalo definido para a discretização:

```
1    def multipleStepMethod(t0, y0, T, n):
2        t_n = [t0]
3        y_n = [y0]
4        h = (T - t0) / n
5
```

```
6         while t_n[-1] < T:
7             y_n.append(y_n[-1] + h*phi(t_n[-1], y_n[-1], h, f))
8             t_n.append(t_n[-1] + h)
9             h = min(h, T - t_n[-1])
10
11        return np.array(t_n), np.array(y_n)
```

Executando-se essa função `multipleStepMethod`, obtivemos a cada ciclo os valores de tk e Y_{tk}. E à partir disso foi-se possível montar o gráfico da figura 6

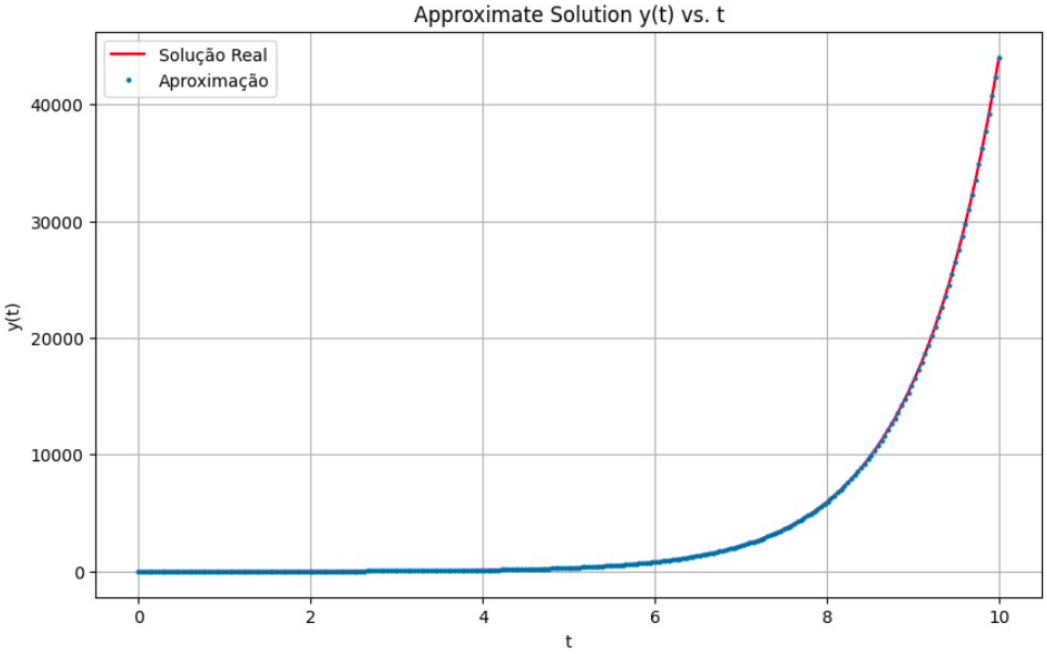

Figura 6: Comparação dos pontos obtidos à partir de RK44 com a curva da solução analítica

Tabela 1: Resultados da iteração

Iteração	**H**	**Stepsize**	**Erro**	**Ordem**
16	6.250×10^{-1}	6.250×10^{-1}	0.000×10^{0}	0.000×10^{0}
32	3.125×10^{-1}	9.375×10^{-1}	4.356×10^{-1}	3.979×10^{0}
64	1.562×10^{-1}	1.094×10^{0}	2.732×10^{-2}	3.995×10^{0}
128	7.812×10^{-2}	1.172×10^{0}	1.709×10^{-3}	3.999×10^{0}
256	3.906×10^{-2}	1.211×10^{0}	1.068×10^{-4}	4.000×10^{0}
512	1.953×10^{-2}	1.230×10^{0}	6.678×10^{-6}	4.000×10^{0}
1024	9.766×10^{-3}	1.240×10^{0}	4.174×10^{-7}	4.000×10^{0}
2048	4.883×10^{-3}	1.245×10^{0}	2.610×10^{-8}	3.999×10^{0}
4096	42.441×10^{-3}	1.248×10^{0}	1.593×10^{-9}	4.034×10^{0}

A tabela 1 representa os índices de convergência do método de Runge Kutta de quarta ordem e quatro estágios implementado. Assim como pode ser ana-

lisado pela última coluna, o método se aproxima da ordem 4, mostrando-se coerente com o método de Runge-Kutta descrito. A terceira coluna também demonstra a convergência do erro à 0, conforme o aumento do número de iterações.

À partir dos dados obtidos do método de Runge-Kutta de quatro iterações, foi então implementado um algorítmo de splines cúbicos, para interpolarmos os pontos obtidos. O código referente à chamada da função de geração do spline está representado à seguir:

```
1    def compute_spline(x: List[float], y: List[float]):
2        n = len(x)
3        if n < 3:
4            raise ValueError('Too short an array')
5        if n != len(y):
6            raise ValueError('Array lengths are different')
7
8        h = compute_changes(x)
9        if any(v < 0 for v in h):
10           raise ValueError('X must be strictly increasing')
11
12       A, B, C = create_tridiagonalmatrix(n, h)
13       D = create_target(n, h, y)
14
15       M = solve_tridiagonalsystem(A, B, C, D)
16
17       coefficients = [[(M[i+1]-M[i])*h[i]*h[i]/6, M[i]*h[i]*h[i]/2, (y[i+1] - y[i] -
    (M[i+1]+2*M[i])*h[i]*h[i]/6), y[i]] for i in range(n-1)]
18
19       def spline(val):
20           idx = min(bisect.bisect(x, val)-1, n-2)
21           z = (val - x[idx]) / h[idx]
22           C = coefficients[idx]
23           return (((C[0] * z) + C[1]) * z + C[2]) * z + C[3]
24
25       return spline
```

Para o cálculo dos splines, é necessária a solução de um sistema linear, que é determinado pela matriz tridiagonal gerada pela função `create_tridiagonalmatrix`. O código com a lógica para a solução da matriz tridiagonal está explícito à seguir:

```
1    def solve_tridiagonalsystem(A: List[float], B: List[float], C: List[float],
    D: List[float]):
2        c_p = C + [0] 3        d_p = [0] * len(B)
4        X = [0] * len(B)
5
6        c_p[0] = C[0] / B[0]
7        d_p[0] = D[0] / B[0]
8        for i in range(1, len(B)):
9            c_p[i] = c_p[i] / (B[i] - c_p[i - 1] * A[i - 1])
```

```
10            d_p[i] = (D[i] - d_p[i - 1] * A[i - 1]) / (B[i] - c_p[i - 1] * A[i - 1])
11
12        X[-1] = d_p[-1]
13        for i in range(len(B) - 2, -1, -1):
14            X[i] = d_p[i] - c_p[i] * X[i + 1]
15
16        return X
```

O resultado proveniente do algorítmo de splines implementado está demonstrado na figura 7

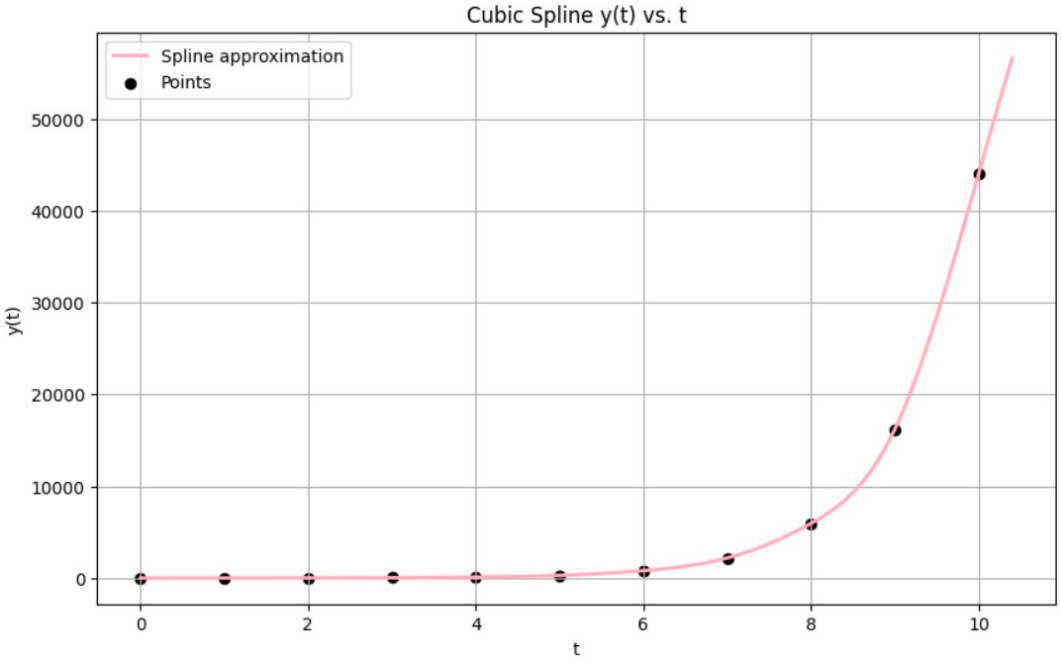

Figura 7: Interpolação por meio de Splines dos pontos da solução estimados

Para a validação inicial, aplicamos o método dos splines para intervalos de pontos maiores do que os definidos pela discretização, para diminuir a complexidade total do algorítmo desenvolvido.

4.2 Resultados da aplicação às Equações de Friedmann

No contexto do universo de De Sitter, assim como mencionado nas seções anteriores, são feitas diversas abstrações e simplificações das características que compõem o universo. Essas simplificações eliminam da composição do universo a presença de matéria e radiação, fazendo com que ele seja denominado um universo dominado pela constante cosmológica. Essa denominação implica no fato de que a constante cosmológica define por completo o comportamento da expansão do universo, o que é fisicamente abstraído no conceito de energia escura, servindo como um campo de expansão presente no universo. Nesse contexto, o

universo de De Sitter é considerado pelos cosmólogos como uma solução estática pras equações de campo de Einstein.

Sendo assim, os valores numéricos que definem a forma como o universo se expande podem ser calculadas de maneira analítica, e resultam nas equações:

$$a(t) = a_0 e^{Ht}$$

$$H = \sqrt{\frac{\Lambda}{3}}$$

Em que H é o parâmetro de Hubble, e é representativo da taxa que define o comportamento do fator de escala do universo nas diferentes soluções das equações de Friedmann.

À partir dessa solução analítica, é possível verificar que o comportamento de expansão do universo de De Sitter é uma exponencial, cujo valor de crescimento é determinado diretamente pelo valor da constante cosmológica. Portanto, os resultados da solução numérica da segunda equação de Friedmann devem convergir à curva exponencial, com crescimento relativo ao valor da constante utilizada.

4.2.1 Sistema de unidades

No contexto da física cosmológica, as unidades de medida do sistema internacional fazem pouco sentido, considerando a escala de tempo e espaço. Portanto, é convencionado dentro da cosmologia utilizar as unidades naturais.

As unidades naturais são atingidas por meio da normalização de constantes utilizadas no escopo da cosmologia e da física moderna. Nesse sistema unitário, a massa e a carga do elétron são normalizadas, as constantes gravitacional e de planck são normalizadas, e que mais afeta o contexto das equações de Friedmann, a velocidade da luz é normalizada:

$$G = \hbar = c = 1$$

Essas simplificações de constantes permitem uma simplificação considerável das expressões das equações, e, além disso, permitem uma visualização mais elementar das relações descritas por cada uma das fórmulas. Nesse contexto, normalizaremos a velocidade da luz para a simulação numérica do universo de De Sitter.

4.2.2 Definição do período da simulação

Devido ao fato de que a constante cosmológica é utilizada como um parâmetro variável para estudar e simular diferentes comportamentos de expansão do universo, a unidade temporal utilizada para definir o período de execução dos métodos numéricos perde parte de seu significado físico (em segundos, minutos, horas). Isso ocorre porque o período do eixo X definido na discretização passa a ser uma unidade de tempo relativa ao valor das unidades e da constante cosmológica utilizadas.

Assim, para obter uma representação adequada da escala de tempo, em relação à taxa na qual eventos significativos ocorrem de acordo com a taxa de crescimento exponencial adotada, utiliza-se o recíproco do Parâmetro de Hubble: $\tau = \sqrt{\frac{3}{\Lambda}}$.

Portanto, ao definir o escopo de simulação parametrizado a partir de τ, garantimos que estamos selecionando um período de tempo que considera um intervalo de mudanças significativas em relação à taxa exponencial de crescimento do universo. Isso é útil para assegurar que o espectro do fator de escala analisado represente de forma significativa o comportamento de expansão do universo, independentemente de sua velocidade ter sido definida como lenta ou rápida.

4.2.3 Resultados numéricos obtidos

O método de aproximação de solução (RK-44), teve de ser adpatado para compreender a lógica descrita entre os estágios de estimação dos $\kappa's$ e $l's$, para a proximação de `a` e `v`. O código referente a essa alteração foi definido da seguinte maneira:

```
1    def f_a(a, v):
2         # a' = v
3         return v
4
5
6    def f_v(a, v):
7         # v' = (lambda * c²) * a / 3
8       return  (Lambda * c**2 / 3) * a
9
10   def phi_deSitter(a, v, dt):
11
12         k1 = f_a(a, v)
13         l1 = f_v(a, v)
14
15         k2 = f_a(a + dt/2 * k1, v + dt/2 * l1)
16         l2 = f_v(a + dt/2 * k1, v + dt/2 * l1)
17
18         k3 = f_a(a + dt/2 * k2, v + dt/2 * l2)
19         l3 = f_v(a + dt/2 * k2, v + dt/2 * l2)
20
21         k4 = f_a(a + dt * k3, v + dt * l3)
22         l4 = f_v(a + dt * k3, v + dt * l3)
23
24         phi_a = (k1 + 2 * k2 + 2 * k3 + k4)/6
25         phi_v = (l1 + 2 * l2 + 2 * l3 + l4)/6
26
27         return phi_a,  phi_v
```

Por sua vez, o loop principal de execução do método está contido na seguinte função:

```
1    def methodForSecondOrder(t0, a0, v0, T, n):
2         t_n = [t0]
3         a_n = [a0[0]]
4         v_n = [v0[0]]
```

```
5         h = (T - t0) / n
6
7         while t_n[-1] < T:
8             phi_a, phi_v = phi_deSitter(a_n[-1], v_n[-1], h)
9
10             # list[-1] gets the last element in list
11             a_n.append(a_n[-1] + h * phi_a)
12             v_n.append(v_n[-1] + h * phi_v)
13
14             t_n.append(t_n[-1] + h)
15             h = min(h, T - t_n[-1])
16
17       return np.array(t_n), np.array(a_n), np.array(v_n)
```

A primeira execução do código foi feita utilizando à partir dos seguintes parâmetros:

$$\begin{aligned} \Lambda &= 1.1 \\ c &= 1 \\ t_0 &= 0 \\ a_0 &= [1.0] \quad \# \text{ Initial scale factor} \\ v_0 &= [1.0] \quad \# \text{ Initial rate of change of the scale factor} \end{aligned}$$

À partir da definição da constante cosmológica, o período de tempo para que hajam mudanças consideráveis é o de $\tau = 1.65$ unidades de tempo, portanto, o período de tempo escolhido foi o maior inteiro após $8 * \tau => T = 14$, que gerou o gráfico da figura 8

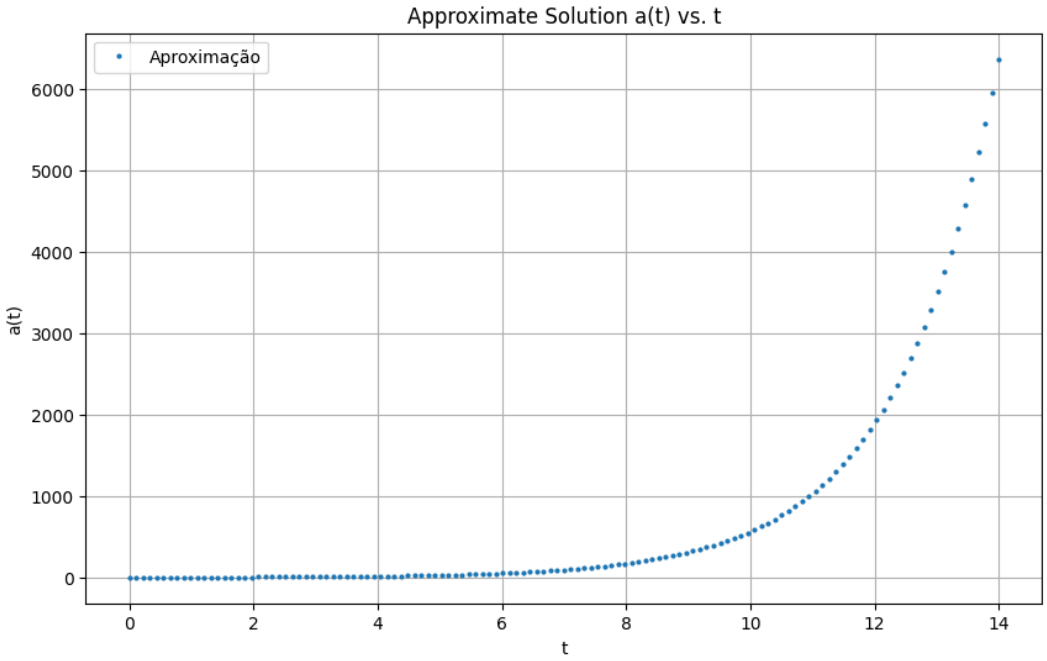

Figura 8: Solução aproximada por RK-44 do universo de De Sitter, com $\Lambda = 1.1$

À partir do gráfico, pode-se ver que a simulação obteve um comportamento do fator de escala similar ao esperado. O caráter exponencial pode ser visto à partir do crescimento explosivo do fator de escala.

Para verificar o funcionamento do método numérico, foi gerada a tabela de convergência 2

Tabela 2: Tabela de convergência do RK-44 para $\Lambda = 1.1$

Interação	**H**	**Stepsize**	**Erro**	**Ordem**
16	8.750×10^{-1}	2.625×10^{0}	2.108×10^{2}	———
32	4.375×10^{-1}	3.062×10^{0}	2.105×10^{1}	3.323×10^{0}
64	2.188×10^{-1}	3.281×10^{0}	1.654×10^{0}	3.670×10^{0}
128	1.094×10^{-1}	3.391×10^{0}	1.159×10^{-1}	3.836×10^{0}
256	5.469×10^{-2}	3.445×10^{0}	7.668×10^{-3}	3.918×10^{0}
512	2.734×10^{-2}	3.473×10^{0}	4.931×10^{-4}	3.959×10^{0}
1024	1.367×10^{-2}	3.486×10^{0}	3.126×10^{-5}	3.979×10^{0}
2048	6.836×10^{-3}	3.493×10^{0}	1.968×10^{-6}	3.990×10^{0}
4096	3.418×10^{-3}	3.497×10^{0}	1.234×10^{-7}	3.995×10^{0}
8192	1.709×10^{-3}	3.498×10^{0}	7.732×10^{-9}	3.997×10^{0}

À partir dessa tabela de convergência, é possível verificar a validade do método de Runge-Kutta implementado. Pode-se observar que a ordem do método converge ao valor 4 com o aumento do número de iterações, da mesma maneira como o valor do erro tende a zero.

Aplicando-se o método dos splines cúbicos nesse caso de teste, obtemos o gráfico da figura 9:

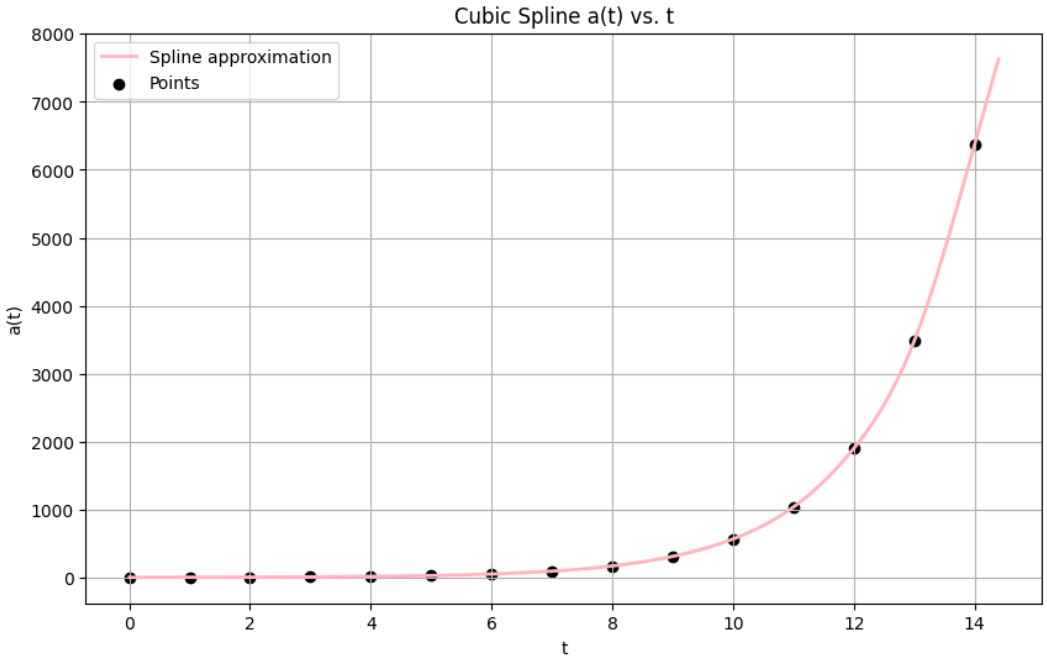

Figura 9: Interpolação por Splines do universo de De Sitter, com $\Lambda = 1.1$

Esse caso de teste foi feito com um valor considerado alto para a constante cosmológica, então para validar o funcionamento do método para valores menores da constante cosmológica. Portanto, a segunda execução do código foi feita com os seguintes parâmetros:

$$\Lambda = 0.001$$
$$c = 1$$
$$t_0 = 0$$
$$a_0 = [1.0] \quad \# \text{ Initial scale factor}$$
$$v_0 = [1.0] \quad \# \text{ Initial rate of change of the scale factor}$$

À partir da nova definição da constante cosmológica, o período de tempo para que hajam mudanças consideráveis é o de $\tau = 54.77$ unidades de tempo. Portanto, o período de tempo escolhido foi o maior inteiro após $4 * \tau => T = 220$, que gerou o gráfico da figura 10

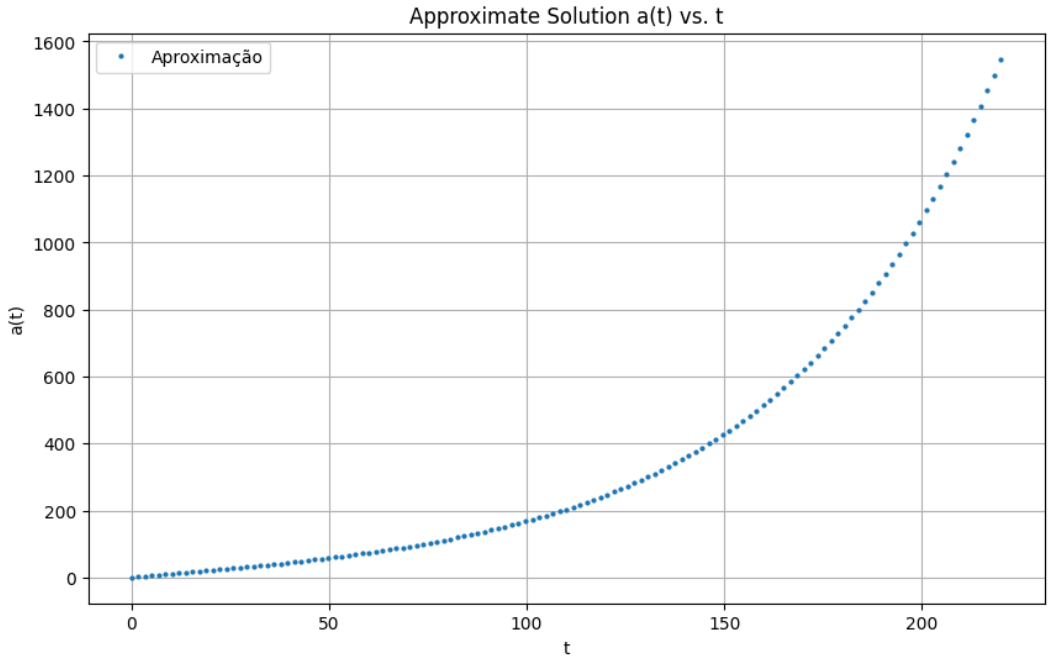

Figura 10: Solução aproximada por RK-44 do universo de De Sitter, com $\Lambda = 0.001$

À partir do gráfico, pode-se ver que apesar do comportamento continuar sendo exponencial, o fator de escala tem caráter consideravelmente menos explosivo que antes, assim como o esperado. A eficácia do método é verificada à partir da tabela 3:

Tabela 3: Tabela de convergncia do RK-44 para $\Lambda = 0.001$

Interação	**H**	**Stepsize**	**Erro**	**Ordem**
16	1.375×10^{1}	4.125×10^{1}	2.006×10^{0}	———
32	6.875×10^{0}	4.812×10^{1}	1.555×10^{-1}	3.689×10^{0}
64	3.438×10^{0}	5.156×10^{1}	1.083×10^{-2}	3.844×10^{0}
128	1.719×10^{0}	5.328×10^{1}	7.143×10^{-4}	3.922×10^{0}
256	8.594×10^{-1}	5.414×10^{1}	4.586×10^{-5}	3.961×10^{0}
512	4.297×10^{-1}	5.457×10^{1}	2.905×10^{-6}	3.981×10^{0}
1024	2.148×10^{-1}	5.479×10^{1}	1.828×10^{-7}	3.990×10^{0}
2048	1.074×10^{-1}	5.489×10^{1}	1.147×10^{-8}	3.995×10^{0}
4096	5.371×10^{-2}	5.495×10^{1}	7.117×10^{-10}	4.010×10^{0}
8192	2.686×10^{-2}	5.497×10^{1}	4.957×10^{-11}	3.844×10^{0}

De maneira similar ao que ocorreu no primeiro exemplo, com o aumento do número de iterações a ordem do método se aproxima de quatro, enquanto o erro converge a zero.

Aplicando-se o método dos splines cúbicos, obtemos a imagem da figura 11

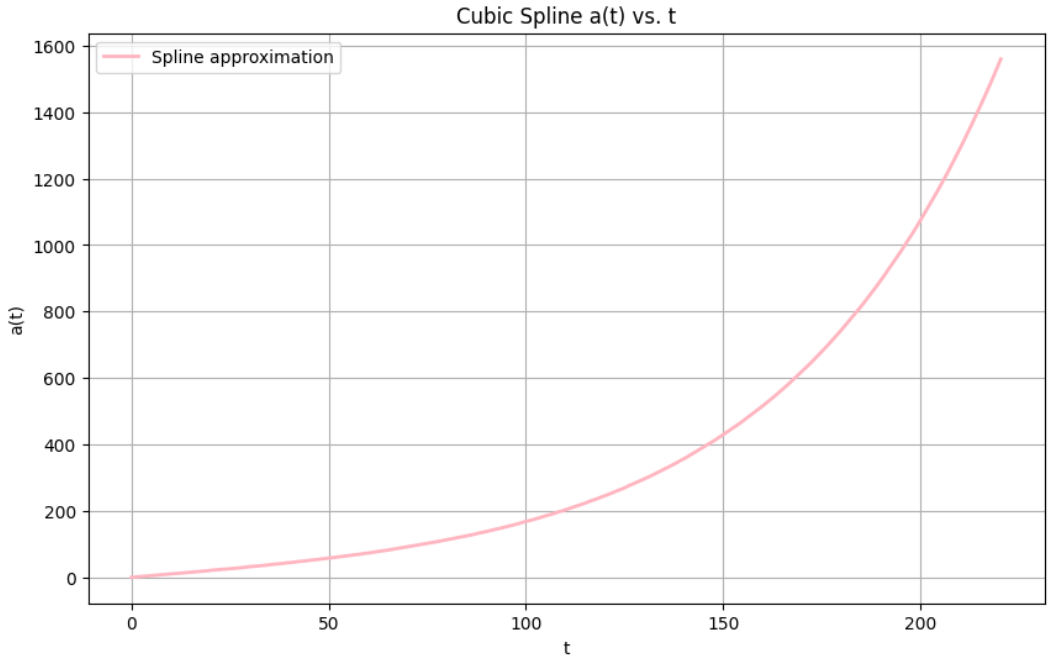

Figura 11: Interpolação por Splines do universo de De Sitter, com $\Lambda = 0.001$

Nota-se que os pontos pretos (que definem os extremos do intervalo que será interpolado), não foram plotados na última imagem pois dificultariam a visualização da curva definida pelos splines.

5 Conclusão

As Equações de Friedmann, derivadas das equações de campo de Einstein sob o princípio cosmológico de homogeneidade e isotropia, são essenciais para nosso entendimento da evolução do universo. Este estudo explorou a aplicação das equações de Friedmann no contexto cosmológico particular do universo de De Sitter. Para tal, calcularam-se os fatores de escala para diferentes definições cosmológicas, por meio de diferentes constantes cosmológicas Λ. O comportamento de expansão do universo de De Sitter pode ser validado tanto para constantes cosmológicas consideradas pequenas quanto para altas, e a sua interdepêndencia com esse fator comprova a definição de universo dominado por Λ, já que esse é responsável por determinar exclusivamente a forma de crescimento desse universo.

O estudo das equações de Friedmann sob a ótica de sua solução numérica permitiu o entendimento e aprofundamento do uso de diferentes métodos numéricos. O método de Runge Kutta se provou muito eficaz para aproximar as soluções numéricas às soluções analíticas, e o conhecimento matemático por trás do cálculo da sua ordem entre números de iterações diferentes, assim como realizado nas tabelas de convergência, permitiu uma forma direta de validação do funcionamento do algorítmo desenvolvido. Acima disso, a variação do método para incluir o funcionamento de EDOs de graus maiores, como a segundo equa-

ção de Friedmann, de segundo grau, permitiu um contato muito mais intrínseco com o funcionamento proposto do método de Runge Kutta de quarta ordem e quatro estágios.

Além disso, o desenvolvimento do método de Splines Cúbicos se provou muito valoroso para o contexto cosmológico abordado. Visto que a variação do período de simulação, assim como descrito na seção 4.2.2, pode acarretar na necessidade de escopos de tempo muito grandes pra incluir uma quantidade de dados adequada para compreender um comportamento significativo da variação do fator de escala, o método dos splines é útil para generalizar o formato da curva do fator de escala `a(t)` à partir de um total menor de pontos que se aproximam da solução analítica.

O contexto do universo de De sitter, considerado uma solução estática para as equações cosmológicas abordadas, serve historicamente como a base do estudo e suposição de universos com características variadas. O universo de De Sitter lida de maneira mais elementar com o processo de expansão do uinverso, justamente por conta das abstrações feitas quanto à presença de matéria e radiação, mas no contexto histórico serviu como o início da descoberta do conceito de energia escura, e da variação da espaço tempo causada intrinsecamente devido à geometria e composição do universo. Nesse contexto, o universo de De Sitter serve como uma forma de *benchmark* para as soluções e modelagens mais complexas e modernas do universo. A cosmologia atual exalta o fato de que o universo de De Sitter se aproxima do comportamento do nosso universo em dois contextos: em até 10^{-33} segundo após o Big Bang, e para comportamentos de expansão do futuro mais distante do nosso universo, onde sua escala é tão superior à concentração dos seus componentes de massa, radiação e energia que estes se tornam desprezíveis. O universo de De Sitter é, portanto, capaz se fazer essa associação entre uma escala micro-temporal e macro-temporal.

A constante cosmológica surge no contexto da solução das equações de Friedmann, e em especial em universos exclusivamente dominados pela constante cosmológica. A constante surgiu, assim como denominado por Einstein, como uma forma de corrigir seu "maior erro", que era o comportamento expansivo do universo. Nas soluções das equações, a constante cosmológica foi inserida artificialmente para estabilizar o crescimento obtido do universo, pois a concepção inicial da cosmologia era a de que o universo era estático. Após Hubble definir o parâmetro de Hubble e validar a expansão do universo à partir de medições do crescimento da distância entre galáxias, tornou-se evidente que o "maior erro"de Einstein foi, na verdade, definir que o comportamento expansivo resultante das suas equações de campo, e das respectivas soluções, não deveria ocorrer.

Além disso, o estudo do comportamento das diferentes soluções do universo de De Sitter viabiliza as discussões acerca dos diferentes fins do universo. A solução do universo de De Sitter mostra uma expansão indefinida do seu conteúdo, e nele haverá um momento que sua escala é tão superior àquela cujos elementos termodinâmicos sejam consideráveis, que estes serão desprezíveis. Neste momento, o universo será incapaz de suportar processos responsáveis por variar a entropia de seu conteúdo interno, sendo essencialmente um vazio absoluto estático. Esse fim é denominado *Heat Death of the Universe*, em detrimento a outros fins como o denominado *Big Crunch*, que ocorre para soluções cíclicas das equações de Friedmann e onde seu comportamento é de compressão,levando à compressão de todo o seu conteúdo em um estado similar ao que gerou o Big Bang. Exemplificações do fator de escala nesses fins e diferentes modelagens do

universo estão demonstradas na figura a seguir:

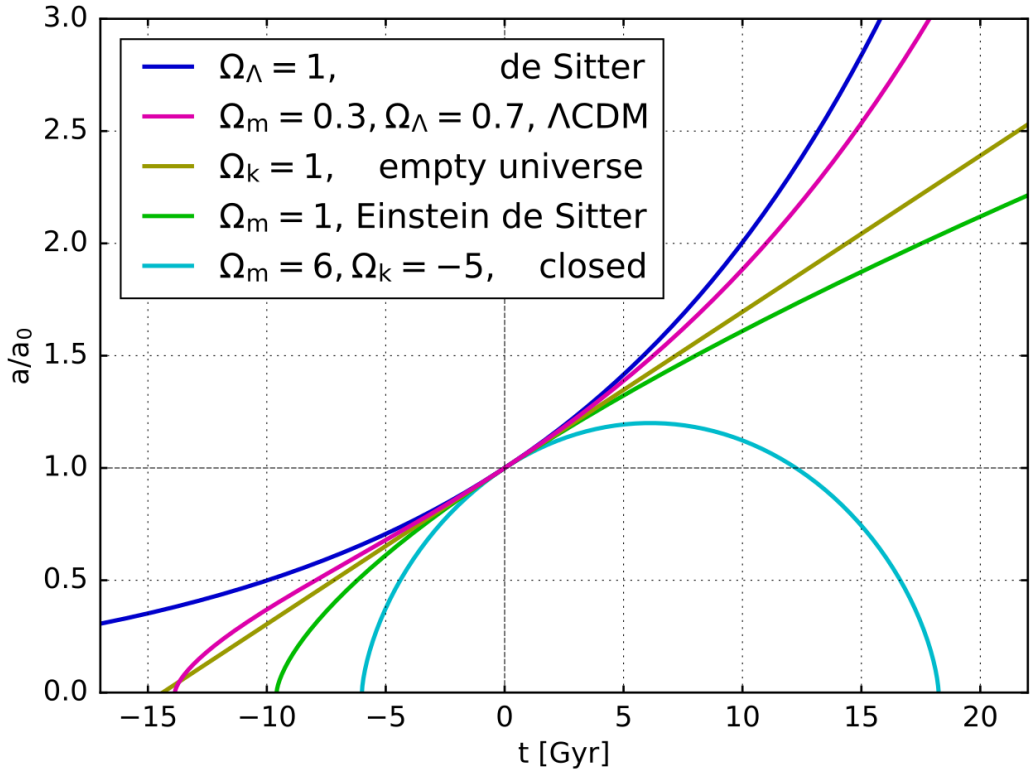

Figura 12: Diferentes soluções para o fator de escala

6 Referências Bibliográficas

1. http://fma.if.usp.br/ mlima/teaching/PGF5299_2021/Review.pdf
2. http://fma.if.usp.br/ mlima/AulaRaul2018/Lecture_Clusters.pdf
3. http://fma.if.usp.br/ mlima/AulaRaul2018/Sph_Col.pdf
4. http://fma.if.usp.br/ mlima/teaching/PGF5292_2021/Dodelson_MC.pdf
5. https://edisciplinas.usp.br/course/view.php?id=64013
6. hrefhttps://wwwth.mpp.mpg.de/members/raffelt/Vorlesungen/WS2009/notes/notes03.pdf https://wwwth.mpp.mpg.de/members/raffelt/Vorlesungen/WS2009/notes/notes03.pdf
7. https://ilpoliedrico.com/wp-content/uploads/2017/06/Solution-Friedmann-equations.pdf
8. https://www.sciencedirect.com/topics/physics-and-astronomy/de-sitter-universe
9. https://blog.scottlogic.com/2020/05/18/cubic-spline-in-python-and- alteryx.html
10. https://www.sciencedirect.com/topics/engineering/cubic-spline
11. https://ned.ipac.caltech.edu/level5/Watson/Watson2_4_1.html
12. https://people.physik.hu-berlin.de/ fjeger/Cosmolect1-7.pdf
13. https://en.wikipedia.org/wiki/Cosmological_constant

14. https://www.researchgate.net/publication/379179026_VALUE_OF_THE_COSMOLOGICAL_CONSTANT_FROM_THE_COSMOLOGICAL_EQUATIONS_OF_THE_UNIVERSE_CONNECTION_OF_THE_COSMOLOGICAL_CONSTANT_WITH_FUNDAMENTAL_PHYSICAL_CONSTANTS

7 Apêndices

O código a seguir foi desenvolvido com a finalidade de testar o spline cúbico, a partir da interpolação de pontos conhecidos da equação do terceiro grau: $x^3 - 5x^2 - 2x + 7$:

```
from typing import Tuple, List
import bisect
import matplotlib.pyplot as plt

def funcao_certa(x):
    return (x**3 - 5*x**2 - 2*x + 7)

def encontrar_y_novo(x_novo, x_vals, y_vals):
    y_novo = []  # Lista para armazenar os valores correspondentes de y

    for x in x_novo:
        # Encontra o índice do valor de x em x_vals
        index = x_vals.index(x)

        # Adiciona o valor correspondente de y em y_vals à lista y_novo
        y_novo.append(y_vals[index])

    return y_novo

def testspline(test_x, x_novo):
    test_y = [funcao_certa(y) for y in test_x]
    spline = compute_spline(test_x, test_y)

    for i, x in enumerate(test_x):
        assert abs(test_y[i] - spline(x)) < 1e-8, f'Error at {x}, {test_y[i]}'
    y_real = [funcao_certa(y) for y in x_novo]

    x_vals = [v / 10 for v in range(0, 85, 1)]
    y_vals = [spline(y) for y in x_vals]
    y_certo = [funcao_certa(y) for y in x_vals]
    y_spline = encontrar_y_novo(x_novo, x_vals, y_vals)


    print("PONTO X | VALOR SPLINE | VALOR ESPERADO")
    for x, y_s, y_r in zip(x_novo, y_spline, y_real):
        print(f"{x} & {y_s:.2f} & {y_r:.2f} \\\\")

    plt.figure(figsize=(10, 6))
```

```
    plt.plot(x_vals, y_vals, color='lightpink', linewidth=2,
    label='Spline approximation')  # Linha rosa clara
    plt.plot(x_vals, y_certo, color='green', linewidth=2,
    label='Exact Solution')  # Linha rosa clara
    plt.scatter(test_x, test_y, color='black', label='Implemented
    Points')  # Pontos pretos

    plt.xlabel('t')
    plt.ylabel('y(t)')
    plt.title('Cubic Spline y(t) vs. t')
    plt.legend()
    plt.grid(True, which='both', linestyle='--', linewidth=0.5)  #
    Ativa o grid para ambos os eixos
    plt.axhline(-17, color='black',linewidth=0.5) # Adiciona linha
    horizontal para o eixo y
    plt.show()

def compute_changes(x: List[float]) -> List[float]:
    return [x[i+1] - x[i] for i in range(len(x) - 1)]

def create_tridiagonalmatrix(n: int, h: List[float]) ->
Tuple[List[float], List[float], List[float]]:
    A = [h[i] / (h[i] + h[i + 1]) for i in range(n - 2)] + [0]
    B = [2] * n
    C = [0] + [h[i + 1] / (h[i] + h[i + 1]) for i in range(n - 2)]
    return A, B, C

def create_target(n: int, h: List[float], y: List[float]):
    return [0] + [6 * ((y[i + 1] - y[i]) / h[i] - (y[i] - y[i - 1]) /
    h[i - 1]) / (h[i] + h[i-1]) for i in range(1, n - 1)] + [0]

def solve_tridiagonalsystem(A: List[float], B: List[float], C:
List[float], D: List[float]):
    c_p = C + [0]
    d_p = [0] * len(B)
    X = [0] * len(B)

    c_p[0] = C[0] / B[0]
    d_p[0] = D[0] / B[0]
    for i in range(1, len(B)):
        c_p[i] = c_p[i] / (B[i] - c_p[i - 1] * A[i - 1])
        d_p[i] = (D[i] - d_p[i - 1] * A[i - 1]) / (B[i] - c_p[i - 1]
        * A[i - 1])

    X[-1] = d_p[-1]
    for i in range(len(B) - 2, -1, -1):
        X[i] = d_p[i] - c_p[i] * X[i + 1]

    return X
```

```
def compute_spline(x: List[float], y: List[float]):
    n = len(x)
    if n < 3:
        raise ValueError('Too short an array')
    if n != len(y):
        raise ValueError('Array lengths are different')

    h = compute_changes(x)
    if any(v < 0 for v in h):
        raise ValueError('X must be strictly increasing')

    A, B, C = create_tridiagonalmatrix(n, h)
    D = create_target(n, h, y)

    M = solve_tridiagonalsystem(A, B, C, D)

    coefficients = [[(M[i+1]-M[i])*h[i]*h[i]/6, M[i]*h[i]*h[i]/2,
    (y[i+1] - y[i] - (M[i+1]+2*M[i])*h[i]*h[i]/6), y[i]] for i in
    range(n-1)]

    def spline(val):
        idx = min(bisect.bisect(x, val)-1, n-2)
        z = (val - x[idx]) / h[idx]
        C = coefficients[idx]
        return (((C[0] * z) + C[1]) * z + C[2]) * z + C[3]

    return spline

#teste para a equação de tercerio grau: x^3 - 5x^2 - 2x + 7
test_x = [0,1,3,4,5,8]
x_teste = [2,3.5,6,7] # pode ser no max o valor final do test_x mais 0.4

testspline(test_x, x_teste)
```

Após a compilação deste código no Google Colab, os resultados obtidos são apresentados da seguinte forma:

1. Tabela de valores: A tabela 4 indica os valores esperados e os valores gerados pelo spline em pontos não conhecidos ou que não foram utilizados para gerar o spline.

2. Gráfico comparativo: A figura 13 mostra a interpolação feita pelo spline e a curva da solução exata.

Essas representações fornecem uma visualização clara dos resultados da interpolação realizada pelo spline cúbico em comparação com a solução exata da equação do terceiro grau.

Tabela 4: Tabela para medir a precisão do Spline

Ponto no eixo X	Valor gerado pelo Spline	Valor da solução exata
2	-8.81	-9.00
3.5	-18.21	-18.38
6	39.69	31.00
7	105.55	91.00

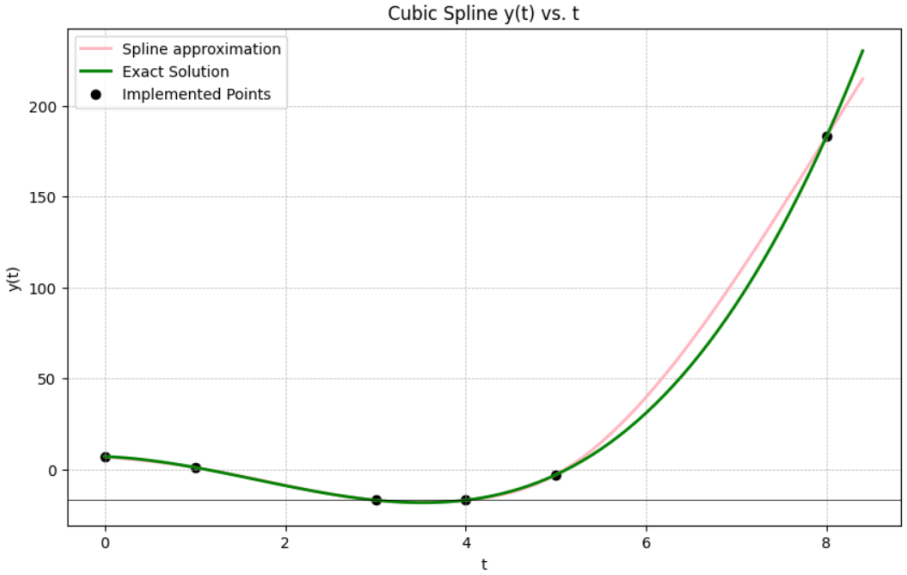

Figura 13: Interpolação por Splines e sua solução exata

O código completo utilizado na aplicação da solução manufaturada está apresentado abaixo. A demonstração de seu funcionamento ocorreu no Capítulo 4.1.

```
import numpy as np
import math
import matplotlib.pyplot as plt
from typing import Tuple, List
import bisect

##  Spline

def compute_changes(x: List[float]) -> List[float]:
    return [x[i+1] - x[i] for i in range(len(x) - 1)]

def create_tridiagonalmatrix(n: int, h: List[float]) ->
Tuple[List[float], List[float], List[float]]:
    A = [h[i] / (h[i] + h[i + 1]) for i in range(n - 2)] + [0]
    B = [2] * n
    C = [0] + [h[i + 1] / (h[i] + h[i + 1]) for i in range(n - 2)]
    return A, B, C
```

```
def create_target(n: int, h: List[float], y: List[float]):
    return [0] + [6 * ((y[i + 1] - y[i]) / h[i] - (y[i] - y[i - 1])
    / h[i - 1]) / (h[i] + h[i-1]) for i in range(1, n - 1)] + [0]

def solve_tridiagonalsystem(A: List[float], B: List[float], C:
List[float], D: List[float]):
    c_p = C + [0]
    d_p = [0] * len(B)
    X = [0] * len(B)

    c_p[0] = C[0] / B[0]
    d_p[0] = D[0] / B[0]
    for i in range(1, len(B)):
        c_p[i] = c_p[i] / (B[i] - c_p[i - 1] * A[i - 1])
        d_p[i] = (D[i] - d_p[i - 1] * A[i - 1]) / (B[i] - c_p[i - 1]
        * A[i - 1])

    X[-1] = d_p[-1]
    for i in range(len(B) - 2, -1, -1):
        X[i] = d_p[i] - c_p[i] * X[i + 1]

    return X

def compute_spline(x: List[float], y: List[float]):
    n = len(x)
    if n < 3:
        raise ValueError('Too short an array')
    if n != len(y):
        raise ValueError('Array lengths are different')

    h = compute_changes(x)
    if any(v < 0 for v in h):
        raise ValueError('X must be strictly increasing')

    A, B, C = create_tridiagonalmatrix(n, h)
    D = create_target(n, h, y)

    M = solve_tridiagonalsystem(A, B, C, D)

    coefficients = [[(M[i+1]-M[i])*h[i]*h[i]/6, M[i]*h[i]*h[i]/2,
    (y[i+1] - y[i] - (M[i+1]+2*M[i])*h[i]*h[i]/6), y[i]] for i in
    range(n-1)]

    def spline(val):
        idx = min(bisect.bisect(x, val)-1, n-2)
        z = (val - x[idx]) / h[idx]
        C = coefficients[idx]
        return (((C[0] * z) + C[1]) * z + C[2]) * z + C[3]
```

```
    return spline

## Plota o Spline-cubic

def testspline(test_x, test_y):
    spline = compute_spline(test_x, test_y)

    for i, x in enumerate(test_x):
        assert abs(test_y[i] - spline(x)) < 1e-8, f'Error at {x},
        {test_y[i]}'

    x_vals = [v / 10 for v in range(0, 105, 1)]
    y_vals = [spline(y) for y in x_vals]

    plt.figure(figsize=(10, 6))
    plt.plot(x_vals, y_vals, color='lightpink', linewidth=2,
    label='Spline approximation')  # Linha rosa clara
    plt.scatter(test_x, test_y, color='black', label='Points')  #
    Pontos pretos
    plt.xlabel('t')
    plt.ylabel('y(t)')
    plt.title('Cubic Spline y(t) vs. t')
    plt.legend()
    plt.grid(True)
    plt.show()

## RUNGE-KUTTA

## FUNÇÃO QUE SERÁ TRABALHADA
def f(t, y):
    return t + y

def phi(t, y, dt, f):
    k1 = f(t, y)
    k2 = f(t + dt/2, y + dt/2*k1)
    k3 = f(t + dt/2, y + dt/2*k2)
    k4 = f(t + dt, y + dt*k3)
    return 1/6*(k1 + 2*k2 + 2*k3 + k4)

def f2(t, y):
    f0 = y[1]
    f1 = 2 * math.exp(t) - 1
    return np.array([f0, f1])


def phi2(t, y, dt):
    k1 = f2(t, y)
    k2 = f2(t + dt/2, y + dt/2*k1)
    k3 = f2(t + dt/2, y + dt/2*k2)
    k4 = f2(t + dt, y + dt*k3)
```

```
    return 1/6*(k1 + 2*k2 + 2*k3 + k4)

def oneStepMethod(t0, y0, T, n):
    t_n = [t0]
    y_n = [y0[0]]
    y_erro = [np.array(y0)]
    h = (T - t0) / n

    while t_n[-1] < T:
        y_n.append(y_n[-1] + h*phi(t_n[-1], y_n[-1], h, f))
        y_erro.append(y_erro[-1] + h*phi2(t_n[-1], y_erro[-1], h))
        t_n.append(t_n[-1] + h)
        h = min(h, T - t_n[-1])
    y_erro = np.array(y_erro)

    return np.array(t_n), np.array(y_n), y_erro[-1]

def obter_valores_aproximacao(t_values, y_values):
    t_inteiros = np.arange(t_values[0][0], t_values[0][-1]+1)
    y_aproximados = np.interp(t_inteiros, t_values[0], y_values[0])
    y_inteiros = np.round(y_aproximados).astype(float)
    return t_inteiros, y_inteiros

def ye(t):
    # exact solution
    return 2 * math.exp(t) - t - 1

def main():
    t0 = 0
    y0 = [1,6]
    T = 10 ## TAMANHO DO MAXIMO DO INTERVALO DE TEMPO
    m = 9  ## NÚMERO DE ITERAÇÕES
    h = [0] * m
    yn = [y0] * m

    t_values = []
    y_values = []
    X_spline = []
    Y_spline = []

    print("   INTERAÇÃO | H | STEPSIZE | ERRO | ORDEM")

    sum_h = 0  # Inicializa a soma de r

    for i in range(1, m+1):
        n = 16 * 2**(i-1)
        t, y, yn[i-1] = oneStepMethod(t0, y0, T, n)
        if i == m:
            t_values.append(t)
            y_values.append(y)
```

```
        h[i-1] = (T - t0) / n

        e = p = q = r = 0
        if i > 1:
            q = abs((ye(T) - yn[i-2][0]) / (ye(T) - yn[i-1][0]))
            r = h[i-2] / h[i-1]
            p = math.log(q) / math.log(r)
            e = abs(ye(T) - yn[i-1][0])

        sum_h += h[i-1]

        print("%5d & %9.3e & %9.3e & %9.3e & %9.3e  \\\\" % (n, h[i-
        1], sum_h, e, p))

    print(" ")

    X_spline, Y_spline = obter_valores_aproximacao(t_values, y_values)
    return t_values, y_values, X_spline, Y_spline

## Plota o runge-kutta

def plot_solution(t_values, y_values):
    plt.figure(figsize=(10, 6))
    ## FUNÇÃO CORRETA
    x_values = np.linspace(min(t_values[0]), max(t_values[0]), 1000)
    y_resultado = 2 * np.exp(x_values) - x_values - 1
    plt.plot(x_values, y_resultado, color='red', linestyle='-',
    label='Solução Real')

    for t, y in zip(t_values, y_values):
        plt.plot(t, y, marker='o', markersize=2, linestyle='None',
        label='Aproximação')

    plt.xlabel('t')
    plt.ylabel('y(t)')
    plt.title('Approximate Solution y(t) vs. t')
    plt.legend()
    plt.grid(True)
    plt.show()

t_values, y_values, X_spline, Y_spline = main()
plot_solution(t_values, y_values)
testspline(X_spline, Y_spline)
```

O código utilizado para gerar as aplicações das equações de Friedmann discretizadas é o seguinte. Mais detalhes sobre ele estão na seção 4.2 do relatório.

```
import numpy as np
```

```
import math
import matplotlib.pyplot as plt
from typing import Tuple, List
import bisect

##  Spline

def compute_changes(x: List[float]) -> List[float]:
    return [x[i+1] - x[i] for i in range(len(x) - 1)]

def create_tridiagonalmatrix(n: int, h: List[float]) ->
Tuple[List[float], List[float], List[float]]:
    A = [h[i] / (h[i] + h[i + 1]) for i in range(n - 2)] + [0]
    B = [2] * n
    C = [0] + [h[i + 1] / (h[i] + h[i + 1]) for i in range(n - 2)]
    return A, B, C

def create_target(n: int, h: List[float], y: List[float]):
    return [0] + [6 * ((y[i + 1] - y[i]) / h[i] - (y[i] - y[i - 1])
    / h[i - 1]) / (h[i] + h[i-1]) for i in range(1, n - 1)] + [0]

def solve_tridiagonalsystem(A: List[float], B: List[float], C:
List[float], D: List[float]):
    c_p = C + [0]
    d_p = [0] * len(B)
    X = [0] * len(B)

    c_p[0] = C[0] / B[0]
    d_p[0] = D[0] / B[0]
    for i in range(1, len(B)):
        c_p[i] = c_p[i] / (B[i] - c_p[i - 1] * A[i - 1])
        d_p[i] = (D[i] - d_p[i - 1] * A[i - 1]) / (B[i] - c_p[i - 1]
        * A[i - 1])

    X[-1] = d_p[-1]
    for i in range(len(B) - 2, -1, -1):
        X[i] = d_p[i] - c_p[i] * X[i + 1]

    return X

def compute_spline(x: List[float], y: List[float]):
    n = len(x)
    if n < 3:
        raise ValueError('Too short an array')
    if n != len(y):
        raise ValueError('Array lengths are different')

    h = compute_changes(x)
    if any(v < 0 for v in h):
        raise ValueError('X must be strictly increasing')
```

```
    A, B, C = create_tridiagonalmatrix(n, h)
    D = create_target(n, h, y)

    M = solve_tridiagonalsystem(A, B, C, D)

    coefficients = [[(M[i+1]-M[i])*h[i]*h[i]/6, M[i]*h[i]*h[i]/2,
    (y[i+1] - y[i] - (M[i+1]+2*M[i])*h[i]*h[i]/6), y[i]] for i in
    range(n-1)]

    def spline(val):
        idx = min(bisect.bisect(x, val)-1, n-2)
        z = (val - x[idx]) / h[idx]
        C = coefficients[idx]
        return (((C[0] * z) + C[1]) * z + C[2]) * z + C[3]

    return spline

## Plota o Spline-cubic

def testspline(test_x, test_y):
    spline = compute_spline(test_x, test_y)

    for i, x in enumerate(test_x):
        assert abs(test_y[i] - spline(x)) < 1e-8, f'Error at {x},
        {test_y[i]}'

    x_vals = [v / 10 for v in range(0, 145, 1)]
    y_vals = [spline(y) for y in x_vals]

    plt.figure(figsize=(10, 6))
    plt.plot(x_vals, y_vals, color='lightpink', linewidth=2,
    label='Spline approximation')  # Linha rosa clara
    plt.scatter(test_x, test_y, color='black', label='Points')  #
    Pontos pretos
    plt.xlabel('t')
    plt.ylabel('a(t)')
    plt.title('Cubic Spline a(t) vs. t')
    plt.legend()
    plt.grid(True)
    plt.show()

# Aproximação

def obter_valores_aproximacao(t_values, y_values):
    t_inteiros = np.arange(t_values[0][0], t_values[0][-1]+1)
    y_aproximados = np.interp(t_inteiros, t_values[0], y_values[0])
    y_inteiros = np.round(y_aproximados).astype(float)
    return t_inteiros, y_inteiros
```

```
# RUNGE-KUTTA

# Constants
Lambda = 1.1  # Cosmological Constant
c = 1       # Speed of light in meters per second

  # a' = v
def f_a(a, v):
  return v

  # v' = (lambda * c²) * a / 3
def f_v(a, v):
  return  (Lambda * c**2 / 3) * a

def phi_deSitter(a, v, dt):

    k1 = f_a(a, v)
    l1 = f_v(a, v)

    k2 = f_a(a + dt/2 * k1, v + dt/2 * l1)
    l2 = f_v(a + dt/2 * k1, v + dt/2 * l1)

    k3 = f_a(a + dt/2 * k2, v + dt/2 * l2)
    l3 = f_v(a + dt/2 * k2, v + dt/2 * l2)

    k4 = f_a(a + dt * k3, v + dt * l3)
    l4 = f_v(a + dt * k3, v + dt * l3)

    phi_a = (k1 + 2 * k2 + 2 * k3 + k4)/6
    phi_v = (l1 + 2 * l2 + 2 * l3 + l4)/6

    return phi_a,  phi_v

def methodForSecondOrder(t0, a0, v0, T, n):
    t_n = [t0]
    a_n = [a0[0]]
    v_n = [v0[0]]
    h = (T - t0) / n

    while t_n[-1] < T:
      phi_a, phi_v = phi_deSitter(a_n[-1], v_n[-1], h)

                # list[-1] gets the last element in list
      a_n.append(a_n[-1] + h * phi_a)
      v_n.append(v_n[-1] + h * phi_v)

      t_n.append(t_n[-1] + h)
      h = min(h, T - t_n[-1])

    return np.array(t_n), np.array(a_n), np.array(v_n)
```

```
def plot_solution(t_values, y_values):
    plt.figure(figsize=(10, 6))

    ## FUNÇÃO CORRETA
    plt.plot(t_values, y_values, marker='o', markersize=2,
    linestyle='None', label='Aproximação')


    plt.xlabel('t')
    plt.ylabel('a(t)')
    plt.title('Approximate Solution a(t) vs. t')
    plt.legend()
    plt.grid(True)
    plt.show()


def main():
    t0 = 0
    a0 = [1.0, 0]  # Initial scale factor
    v0 = [1.0, 0]  # Initial rate of change of the scale factor
    T = 14  # Total time
    m = 7   # Number of steps1.2248684953 * 10 ³
    h = [0] * (m+1)
    erro = []
    order = []
    results_all = []
    steps = []
    t_values = []
    y_values = []
    X_spline = []
    Y_spline = []

    sum_h = []  # Inicializa a soma de r

    n = 2**2
    t_1, a_1, v_1 = methodForSecondOrder(t0, a0, v0, T, n)

    #i = 3
    n = 2**3
    t_n, a_n, v_n = methodForSecondOrder(t0, a0, v0, T, n)
    erro.append(np.linalg.norm(np.array(a_1 - a_n[-1])))
    order.append('-----------')
    steps.append(n)
    h[0] = (T - t0) / n
    sum_h.append(h[0])
    results_all.append(a_n)

    # print(int(64),'&',"&",f'{erro[3]:e}','&','-----------')
```

```
    for i in range(1, m+1):
        n = 16 * 2**(i-1)
        t_n, a_n, v_n = methodForSecondOrder(t0, a0, v0, T, n)
        if i == m:
            t_values.append(t_n)
            y_values.append(a_n)

        h[i] = (T - t0) / n

        erro.append(np.linalg.norm(np.array(results_all[-1][-1] -
        a_n[-1])))
        order.append((np.log(erro[-2]/erro[-1]))/np.log(2))
        steps.append(n)
        sum_h.append(sum_h[-1] + h[i])
        results_all.append(a_n)


    print("INTERAÇÃO | H | STEPSIZE | ERRO | ORDEM")

    print(int(steps[1]),'&',f'{h[1]:.3e}',"&",f'{sum_h[1]:.3e}',"&",f
    '{erro[1]:.3e}','&','-----------', '\\\\')
    for i in range(2,m+1):
      print(int(steps[i]), '&', f'{h[i]:.3e}', "&",
      f'{sum_h[i]:.3e}', "&", f'{erro[i]:.3e}', '&',
      f'{order[i]:.3e}', '\\\\')

    X_spline, Y_spline = obter_valores_aproximacao(t_values,
    y_values)
    return t_n, a_n, v_n, X_spline, Y_spline

t_values, a_values ,v_values, X_spline, Y_spline = main()
plot_solution(t_values, a_values)
testspline(X_spline, Y_spline)
```

Todos os códigos foram desenvolvidos no Google Colab. O link para o Colab está disponível aqui.