Comparação de métodos de integração numérica de Euler, Heun e Runge-Kutta 4

Resumo Este projeto implementa a simulação do Sistema de Lorenz, um exemplo clássico da teoria do caos, utilizando métodos numéricos para resolver equações diferenciais ordinárias (EDOs). Foram aplicados os métodos de Euler, Heun e Runge-Kutta de 4ª ordem (RK4), além de uma solução refinada com RK4 utilizando um passo de tempo menor. As trajetórias geradas pelos métodos foram comparadas e os erros globais calculados. Além disso, uma pequena perturbação foi adicionada ao sistema a fim de investigar como a sensibilidade às condições iniciais geram resultados inesperados ao longo do tempo.

Keywords: Sistema de Lorenz, simulação numérica

1 Introdução

O estudo de sistemas dinâmicos caóticos tem aplicação em diversas áreas da ciência e engenharia, incluindo previsões climáticas, modelagem de turbulência e sistemas não-lineares. Um dos exemplos mais emblemáticos é o sistema de Lorenz, que descreve o comportamento caótico de um sistema tridimensional simplificado para convecção atmosférica e dá origem ao famoso atrator caótico de Lorenz. Neste trabalho, serão implementados três métodos numéricos para a resolução das EDO's associadas ao sistema de Lorenz: Método de Euler, Método de Heun (Euler Modificado) e Runge-Kutta de Quarta Ordem (RK4). Para avaliar a eficácia desses métodos na captura das características caóticas do sistema, utilizaremos métricas comparativas como erro global, precisão das trajetórias, tempo de execução e estabilidade numérica. O objetivo é comparar como a solução obtida por cada um desses métodos impacta nas métricas de avaliação adotadas.

2 Problema proposto

A proposta de problema está descrita no arquivo HW2.pdf no Github com as soluções implementadas nesse trabalho. Acesse: Repositório do GitHub

3 Método de Euler

O Método de Euler é o método numérico mais simples para resolver equações diferenciais ordinárias. A fórmula geral para o Método de Euler é:

$$y_{n+1} = y_n + hf(t_n, y_n) \tag{1}$$

onde.

- y_{n+1}: valor estimado da solução no próximo ponto.
- y_n: valor atual da solução.
- h: passo de tempo.
- $f(t_n, y_n)$: é a função que define a derivada de $\frac{dy}{dt}$ no ponto (t_n, y_n).

3.1 Aplicação no sistema de Lorenz

$$\begin{cases} \frac{dx}{dt} = \sigma(y - x) \\ \frac{dy}{dt} = x(\rho - z) - y \\ \frac{dz}{dt} = xy - \beta z \end{cases} \tag{2}$$

Assim, o método de Euler atualiza os próximos passos da seguinte maneira:

$$x_{n+1} = x_n + h\sigma(y_n - x_n) \tag{3}$$

$$y_{n+1} = y_n + h(x_n(\rho - z_n) - y_n) \tag{4}$$

$$z_{n+1} = z_n + h(x_n y_n - \beta z_n) \tag{5}$$

3.2 Implementação do método de Euler

Para o caso do atrator de Lorenz, o método de Euler é implementado em Python da seguinte forma:

```python
#Metodo de Euler
def metodo_euler(estado, h, num_passos):
    x, y, z = estado
    trajetoria = [estado]
    for _ in range(num_passos):
        dx = sigma * (y - x)
        dy = x * (rho - z) - y
        dz = x * y - beta * z
        x += h * dx
        y += h * dy
        z += h * dz
        trajetoria.append([x, y, z])
    return np.array(trajetoria)
```

4 Método de Heun

O Método de Heun, também conhecido como **método do trapézio explícito**, é um método numérico de segunda ordem para resolver equações diferenciais ordinárias. Ele refina o Método de Euler utilizando uma estimativa inicial para calcular um valor mais preciso da solução.

A fórmula geral para o Método de Heun é:

$$y_{n+1} = y_n + \frac{h}{2}\left[f(t_n, y_n) + f(t_{n+1}, y^*_{n+1})\right] \tag{6}$$

onde:

– y_{n+1}: valor estimado da solução no próximo ponto;
– y_n: valor atual da solução;
– h: passo de tempo;
– $f(t_n, y_n)$: função que define a derivada $\frac{dy}{dt}$ no ponto (t_n, y_n);
– y^*_{n+1}: predição inicial do próximo valor, calculada pelo Método de Euler:

$$y^*_{n+1} = y_n + h \cdot f(t_n, y_n). \tag{7}$$

4.1 Aplicação no sistema de Lorenz

O sistema de Lorenz é definido pelas seguintes equações diferenciais descritas em 2

No Método de Heun, cada variável do sistema é atualizada utilizando uma predição inicial seguida por uma correção.

Predição inicial (Método de Euler)

$$x^*_{n+1} = x_n + h \cdot \sigma(y_n - x_n), \tag{8}$$

$$y^*_{n+1} = y_n + h \cdot [x_n(\rho - z_n) - y_n], \tag{9}$$

$$z^*_{n+1} = z_n + h \cdot [x_n y_n - \beta z_n]. \tag{10}$$

Correção (média dos gradientes)

$$x_{n+1} = x_n + \frac{h}{2}\left[\sigma(y_n - x_n) + \sigma(y^*_{n+1} - x^*_{n+1})\right], \tag{11}$$

$$y_{n+1} = y_n + \frac{h}{2}\left[x_n(\rho - z_n) - y_n + x^*_{n+1}(\rho - z^*_{n+1}) - y^*_{n+1}\right], \tag{12}$$

$$z_{n+1} = z_n + \frac{h}{2}\left[x_n y_n - \beta z_n + x^*_{n+1} y^*_{n+1} - \beta z^*_{n+1}\right]. \tag{13}$$

4.2 Implementação do Método de Heun

Para o caso do atrator de Lorenz, o Método de Heun pode ser implementado em Python da seguinte forma:

```
#Metodo de Heun
def metodo_heun(estado, h, num_passos):
    x, y, z = estado
    trajetoria = [estado]
    for _ in range(num_passos):
        # Predicao inicial (Euler)
        dx_euler = sigma * (y - x)
        dy_euler = x * (rho - z) - y
        dz_euler = x * y - beta * z
        x_pred = x + h * dx_euler
```

```
        y_pred = y + h * dy_euler
        z_pred = z + h * dz_euler

        # Correcao (Heun)
        dx_corr = sigma * (y_pred - x_pred)
        dy_corr = x_pred * (rho - z_pred) - y_pred
        dz_corr = x_pred * y_pred - beta * z_pred
        x += h / 2 * (dx_euler + dx_corr)
        y += h / 2 * (dy_euler + dy_corr)
        z += h / 2 * (dz_euler + dz_corr)
        trajetoria.append([x, y, z])
    return np.array(trajetoria)
```

5 Método de Runge-Kutta de Quarta Ordem

O Método de Runge-Kutta de quarta ordem (RK4) é amplamente utilizado devido à sua alta precisão e simplicidade relativa. Ele calcula a solução de equações diferenciais ordinárias estimando o valor da próxima iteração com base em quatro aproximações do gradiente.

A fórmula geral para o RK4 é:

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{14}$$

onde:

$$k_1 = f(t_n, y_n), \tag{15}$$

$$k_2 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right), \tag{16}$$

$$k_3 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2\right), \tag{17}$$

$$k_4 = f(t_n + h, y_n + hk_3). \tag{18}$$

Os termos k_1, k_2, k_3 e k_4 correspondem a diferentes estimativas do gradiente da função:

- k_1: gradiente no início do intervalo;
- k_2: gradiente no ponto médio do intervalo, usando k_1;
- k_3: outra estimativa no ponto médio do intervalo, usando k_2;
- k_4: gradiente no final do intervalo, usando k_3.

Esses valores são combinados de forma ponderada para calcular y_{n+1}, como mostrado na Equação 14.

5.1 Aplicação no sistema de Lorenz

O sistema de Lorenz é descrito pelas equações em 2

No Método RK4, as atualizações das variáveis x, y e z seguem o mesmo princípio geral, utilizando as fórmulas para k_1, k_2, k_3 e k_4 para cada equação diferencial.

Cálculo de k_1

$$k_{1x} = \sigma(y_n - x_n), \tag{19}$$
$$k_{1y} = x_n(\rho - z_n) - y_n, \tag{20}$$
$$k_{1z} = x_n y_n - \beta z_n. \tag{21}$$

Cálculo de k_2

$$k_{2x} = \sigma\left[(y_n + \frac{h}{2}k_{1y}) - (x_n + \frac{h}{2}k_{1x})\right], \tag{22}$$
$$k_{2y} = \left(x_n + \frac{h}{2}k_{1x}\right)\left(\rho - \left(z_n + \frac{h}{2}k_{1z}\right)\right) - \left(y_n + \frac{h}{2}k_{1y}\right), \tag{23}$$
$$k_{2z} = \left(x_n + \frac{h}{2}k_{1x}\right)\left(y_n + \frac{h}{2}k_{1y}\right) - \beta\left(z_n + \frac{h}{2}k_{1z}\right). \tag{24}$$

Cálculo de k_3

$$k_{3x} = \sigma\left[(y_n + \frac{h}{2}k_{2y}) - (x_n + \frac{h}{2}k_{2x})\right], \tag{25}$$
$$k_{3y} = \left(x_n + \frac{h}{2}k_{2x}\right)\left(\rho - \left(z_n + \frac{h}{2}k_{2z}\right)\right) - \left(y_n + \frac{h}{2}k_{2y}\right), \tag{26}$$
$$k_{3z} = \left(x_n + \frac{h}{2}k_{2x}\right)\left(y_n + \frac{h}{2}k_{2y}\right) - \beta\left(z_n + \frac{h}{2}k_{2z}\right). \tag{27}$$

Cálculo de k_4

$$k_{4x} = \sigma\left[(y_n + hk_{3y}) - (x_n + hk_{3x})\right], \tag{28}$$
$$k_{4y} = (x_n + hk_{3x})\left(\rho - (z_n + hk_{3z})\right) - (y_n + hk_{3y}), \tag{29}$$
$$k_{4z} = (x_n + hk_{3x})(y_n + hk_{3y}) - \beta\left(z_n + hk_{3z}\right). \tag{30}$$

Atualizações finais As atualizações para x, y e z são então calculadas como:

$$x_{n+1} = x_n + \frac{h}{6}\left(k_{1x} + 2k_{2x} + 2k_{3x} + k_{4x}\right), \tag{31}$$
$$y_{n+1} = y_n + \frac{h}{6}\left(k_{1y} + 2k_{2y} + 2k_{3y} + k_{4y}\right), \tag{32}$$
$$z_{n+1} = z_n + \frac{h}{6}\left(k_{1z} + 2k_{2z} + 2k_{3z} + k_{4z}\right). \tag{33}$$

5.2 Implementação do Método RK4

A implementação do Método RK4 para o atrator de Lorenz pode ser escrita em Python como segue:

```python
# --- Metodo RK4 ---
def sistema_lorenz(estado, sigma, rho, beta):
    x, y, z = estado
    dx = sigma * (y - x)
    dy = x * (rho - z) - y
    dz = x * y - beta * z
    return np.array([dx, dy, dz])

def runge_kutta_4(estado, h, num_passos):
    trajetoria = [estado]
    for _ in range(num_passos):
        k1 = h * sistema_lorenz(estado, sigma, rho, beta)
        k2 = h * sistema_lorenz(estado + 0.5 * k1, sigma, rho, beta)
        k3 = h * sistema_lorenz(estado + 0.5 * k2, sigma, rho, beta)
        k4 = h * sistema_lorenz(estado + k3, sigma, rho, beta)
        estado = estado + (k1 + 2 * k2 + 2 * k3 + k4) / 6
        trajetoria.append(estado)
    return np.array(trajetoria)
```

6 Resultados

6.1 Resultado das implementações

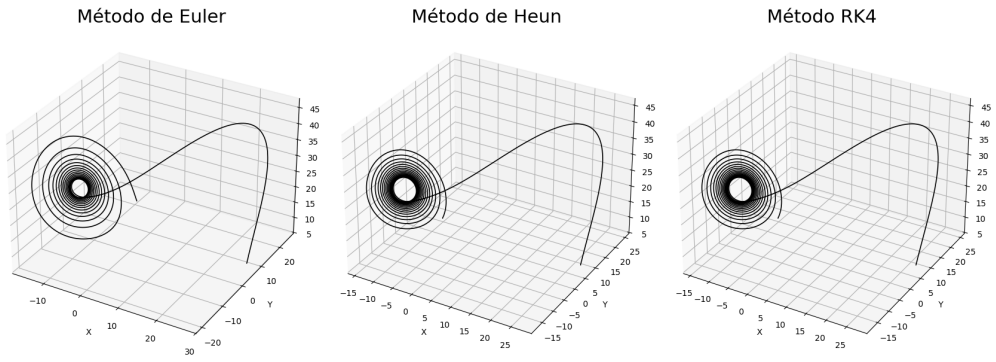

Figura 1. Comparação gráfica dos métodos

6.2 Teste: diferença entre trajetórias

A norma da diferença entre as trajetórias é uma métrica que quantifica a diferença entre duas trajetórias, uma representando solução de referência (RK4 com passo de 10e-5) e a outra representando a solução aproximada obtida pelos métodos de Euler, Heun e RK4 com passo de 10e-4. Essa métrica fornece uma medida global do erro entre as duas trajetórias ao longo do tempo.

Seja $\mathbf{y}_{\text{referência}}(t)$ a trajetória de referência e $\mathbf{y}_{\text{aproximada}}(t)$ a trajetória obtida pelo método de integração numérica. A norma da diferença entre as trajetórias é dada por:

$$\text{Erro} = \|\mathbf{y}_{\text{verdadeira}} - \mathbf{y}_{\text{aproximada}}\|_2$$

onde $\|\cdot\|_2$ representa a norma L^2 (ou norma Euclidiana) e é calculada como:

$$\|\mathbf{y}_{\text{verdadeira}} - \mathbf{y}_{\text{aproximada}}\|_2 = \sqrt{\sum_{i=1}^{n} (y_{\text{verdadeira},i} - y_{\text{aproximada},i})^2}$$

A norma da diferença fornece uma única medida que resume a discrepância entre as trajetórias em todos os pontos no tempo. Um valor menor indica que as soluções aproximadas estão mais próximas da solução verdadeira, enquanto um valor maior indica um maior erro.

Os resultados obtidos nesse teste foram:

- Erro Euler: 2.538788
- Erro Heun: 0.005781
- Erro RK4 coarse: 0.000062

Ainda relacionado a diferença nas trajetórias, foi adicionado uma pequena perturbação dos valores iniciais do problema. Cada valor do ponto inicial foi incrementado em 0.01. O resultado do comportamento caótico do sistema é mostrado na figura 2.

É possível notar que nas 10 primeiras medidas de tempo as trajetórias são bastante coincidentes, divergindo fortemente das trajetórias originais a partir disso. Esse fenômento é conhecido como sensibilidade às condições iniciais[1].

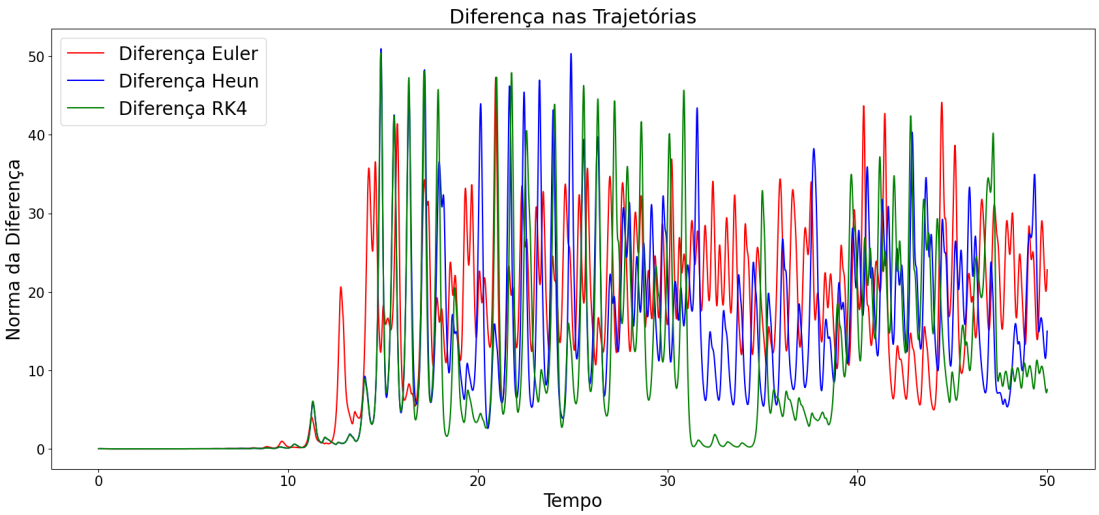

Figura 2. Diferenças nas trajetórias após perturbação

Referências Bibliográficas

[1] João Victor Carriello Celes. Simulação computacional de sistemas dinâmicos no contexto da teoria do caos, Setembro 2022.