MAP214 -Cálculo Numérico com Aplicações em Física
EP4

Guilherme Pacheco Paredes N°USP: 12693754

Instituto de Física, Universidade de São Paulo - SP

29/11/2024

Os exercícios computacionais deste EP foram feitos utilizando a linguagem de prgramação Python. As bibliotecas Numpy e matplotlib.pyplot foram utilizadas para operações matemáticas e plots dos gráficos.

1

Para obter a equação de 2ª ordem $\ddot{y} = \dot{y} + y - t^3 + 7t + 1$ pelos métodos de Euler e Runge-Kutta Clássico (4ª ordem-RK4), foi implementado o seguinte programa em precisão dupla:

```
import numpy as np


def g(t, y, z):
    return z + y - t ** 3 - 3 * t ** 2 + 7 * t + 1


def euler(t_ini, y_ini, z_ini, h, tf):
    t, y, z = t_ini, y_ini, z_ini
    result = [(t, y, z)]

    for _ in range(tf):
        z_next = z + h * g(t, y, z)
        y_next = y + h * z
        y, z, t = y_next, z_next, t + h
        result.append((t, y, z))

    return np.array(result)


def rk4(t_ini, y_ini, z_ini, h, n):
    t, y, z = t_ini, y_ini, z_ini
    result = [(t, y, z)]

    for _ in range(n):
        k1y = h * z
        k1z = h * g(t, y, z)

        k2y = h * (z + k1z / 2)
        k2z = h * g(t + h / 2, y + k1y / 2, z + k1z / 2)

        k3y = h * (z + k2z / 2)
        k3z = h * g(t + h / 2, y + k2y / 2, z + k2z / 2)

        k4y = h * (z + k3z)
        k4z = h * g(t + h, y + k3y, z + k3z)

        y_next = y + (k1y + 2 * k2y + 2 * k3y + k4y) / 6
        z_next = z + (k1z + 2 * k2z + 2 * k3z + k4z) / 6

        y, z, t = y_next, z_next, t + h
```

```
        result.append((t, y, z))

    return np.array(result)


t0 = 0
y0 = 0
z0 = -1
step = 0.01
N = 500

euler_result = euler(t0, y0, z0, step, N)
rk4_result = rk4(t0, y0, z0, step, N)

print(f"Método de Euler: y(5) = {euler_result[-1, 1]:.8f}, y'(5) = {euler_result[-1, 2]:.8f}")
print(f"Método de rk4: y(5) = {rk4_result[-1, 1]:.8f}, y'(5) = {rk4_result[-1, 2]:.8f}")
```

E a saída com 8 dígitos:

```
Método de Euler: y(5) = 84.84606752, y'(5) = 17.34498804
Método de rk4: y(5) = 119.99999704, y'(5) = 73.99999521
```

Para precisão simples, implementando NP.FLOAT32 nas variáveis utilizadas paras as operações, obteve-se a seguinte saída:

```
Método de Euler: y(5) = 84.84841156, y'(5) = 17.34827614
Método de rk4: y(5) = 120.00251007, y'(5) = 74.00354767
```

Para fins de comparação, a partir da solução analítica $y = t^3 - t$ e $dy/dt = 3t^2 - t$, temos que a solução exata em $t = 5$ é $y(5) = 120$ e $\dot{y}(5) = 74$. Pode se observar que o método de Euler, tanto por precisão simples quanto por precisão dupla possui grandes imprecisões devido ao seu erro global proporcional a $\mathcal{O}(h)$, apresentou soluções significativamente distante da solução exata. Esse desvio resulta da combinação de erros de truncamento, que se acumulam ao longo dos 500 passos, e de roundoff, amplificado pelo número elevado de iterações. $N = 500$ representa um equilíbrio entre essas duas fontes de erro, oferecendo a melhor aproximação dentro das limitações do método. No entanto, a simplicidade de Euler é insuficiente para problemas que exigem maior precisão, sendo preferíveis métodos de ordem superior, como o Runge-Kutta em que o erro absoluto foi de $y(5)$ for de $|\text{erro}| = |119.99999704 - 120| \approx 2.96 \times 10^{-06}$ e para $\dot{y}(5)$ com $|\text{erro}| = |73.99999521 - 74| \approx 4.8 \times 10^{-06}$ em precisão dupla.

2

A fim de se obter o espaço de fase para a equação do poço de potencial duplo, do poço de potencial amortecido e do poço de potencial forçado, foi reutilizada a função definida no exercício anterior para a rotina do método de Runge-Kutta acrescentando os parâmetros necessários para cada caso, alterando a função $g(t, y, z)$ conforme requisitado. Também, foi utilizado um passo (h = 0.1) e t evoluindo de 0 a 1000. Por fim, foi definida uma função para remover os 500 primeiros do item (c) referentes ao transiente da evolução do espaço de fase. A seguir o programa referente ao item (c), dado que os itens anteriores foram realizados os plots dos gráficos com seus respectivos parâmetros bem como suas respectivas varições:

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

```python
# Potencial poço duplo com forca externa
def g(t, x, v, f, w):
    return 0.5 * x * (1 - 4 * x ** 2) + f * np.cos(w * t) - 0.25 * v


def rk4(t_ini, x_ini, v_ini, h, t_fin, f, w):
    t, x, v = t_ini, x_ini, v_ini
    result = [(t, x, v)]

    while t < t_fin:
        k1x = h * v
        k1v = h * g(t, x, v, f, w)

        k2x = h * (v + k1v / 2)
        k2v = h * g(t + h / 2, x + k1x / 2, v + k1v / 2, f, w)

        k3x = h * (v + k2v / 2)
        k3v = h * g(t + h / 2, x + k2x / 2, v + k2v / 2, f, w)

        k4x = h * (v + k3v)
        k4v = h * g(t + h, x + k3x, v + k3v, f, w)

        x_next = x + (k1x + 2 * k2x + 2 * k3x + k4x) / 6
        v_next = v + (k1v + 2 * k2v + 2 * k3v + k4v) / 6

        x, v, t = x_next, v_next, t + h
        result.append((t, x, v))

    return np.array(result)

# Parâmetros iniciais
t0 = 0
x0 = -0.5
v0 = 0.5
F_values = [0.11, 0.115, 0.14, 0.35]
omega = 1
step = 0.01
tf = 1000


# Função para remover o transiente (os primeiros 500 dos pontos)
def remove_transient(data):
    transient_cutoff = int(0.5 * len(data))
    return data[transient_cutoff:]


plt.figure(figsize=(8, 6))


for F in F_values:
    results = rk4(t0, x0, v0, step, tf, F, omega)
    result_no_transient = remove_transient(results)
    plt.plot(result_no_transient[:, 1], result_no_transient[:, 2], label=f'F = {F}')

plt.xlabel('x(t)')
plt.ylabel('v(t)')
plt.title('Espaço de Fase - Potencial com Força Externa e Amortecimento')
plt.legend()
plt.show()
```

E a saída com os diagramas de espaço de fase para cada um dos itens:

a) Espaço de Fase para o potencial de poço duplo:

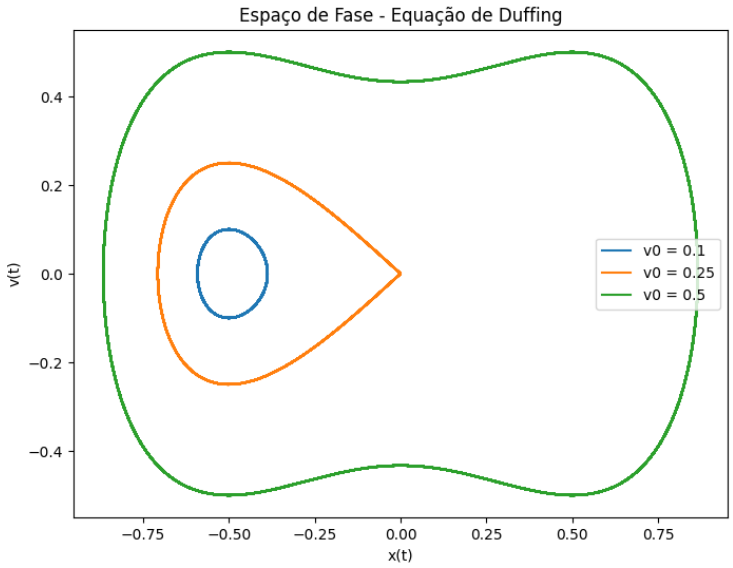

Figura 1: Gráfico do espaço de fase para o potencial poço duplo $\ddot{x} - \frac{1}{2}x(1 - 4x^2) = 0$.

O atrator neste caso foi o ciclo limite, dado que o sistema oscila indefinidamente dentro de uma trajetória fechada no espaço de fase. Isso ocorre porque não há perda de energia (sem amortecimento), então o sistema se mantém em movimento periódico ao longo do ciclo.

b) Espaço de Fase para o potencial de poço duplo amortecido:

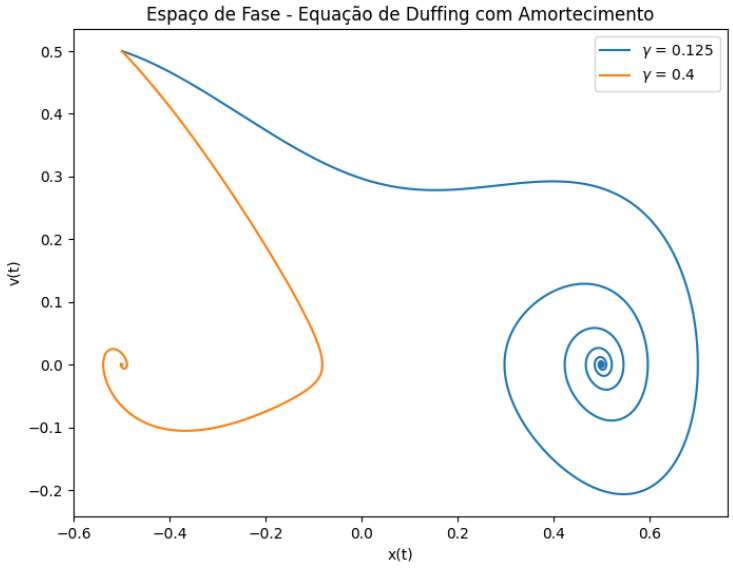

Figura 2: Gráfico do espaço de fase para o potencial poço duplo amortecido $\ddot{x} + 2\gamma\dot{x} - \frac{1}{2}x(1 - 4x^2) = 0$.

Para este caso, o atrator é um ponto fixo. O amortecimento remove energia do sistema, fazendo com que as oscilações diminuam gradualmente até o sistema atingir o equilíbrio no ponto de menor energia do potencial (o atrator estável). Dependendo da intensidade do amortecimento (γ), o sistema pode oscilar antes de se estabilizar (amortecimento fraco) ou simplesmente decair

sem oscilar (amortecimento forte).

c) Espaço de Fase para o potencial de poço duplo forçado:

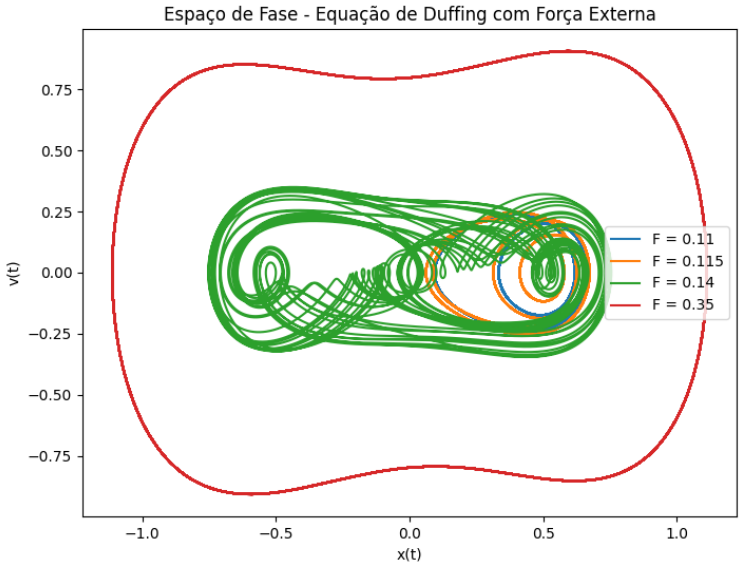

Figura 3: Gráfico do espaço de fase para o potencial poço duplo forçado $\ddot{x} + 0.25\dot{x} + -\frac{1}{2}x(1 - 4x^2) = F\cos(\omega t)$.

Por fim, o atrator também é o ciclo limite. A força externa aplicada periodicamente impede que o sistema alcance o equilíbrio. Em vez disso, o sistema oscila continuamente em resposta à força externa. No espaço de fase, isso se traduz em uma trajetória fechada estável que corresponde a oscilações periódicas no sistema, mas essas oscilações agora dependem da frequência (ω) e amplitude (F) da força externa.

3

Agora queremos obter o Diagrama de Bifurcação construindo um novo programa e mantendo as condições iniciais do exercício anterior, obtendo para um dado F as seções de Poincaré e determinando os valores de x correspondentes. Para isso, implementamos o seguinte programa:

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

omega = 1


# Potencial poço duplo com força externa
def g(t, x, v, f, w):
    return 0.5 * x * (1 - 4 * x ** 2) + f * np.cos(w * t) - 0.25 * v


def rk4(t_ini, x_ini, v_ini, h, t_fin, f, w):
    t, x, v = t_ini, x_ini, v_ini

    while t < t_fin:
```

```
        k1x = h * v
        k1v = h * g(t, x, v, f, w)

        k2x = h * (v + k1v / 2)
        k2v = h * g(t + h / 2, x + k1x / 2, v + k1v / 2, f, w)

        k3x = h * (v + k2v / 2)
        k3v = h * g(t + h / 2, x + k2x / 2, v + k2v / 2, f, w)

        k4x = h * (v + k3v)
        k4v = h * g(t + h, x + k3x, v + k3v, f, w)

        x_next = x + (k1x + 2 * k2x + 2 * k3x + k4x) / 6
        v_next = v + (k1v + 2 * k2v + 2 * k3v + k4v) / 6

        x, v, t = x_next, v_next, t + h

    return x, v, t


# Função principal para calcular o diagrama de bifurcação
def bifurcation_diagram(omega=1.0, max_F=0.35, delta_F=0.00025, h=0.001 * 2 * np.pi / omega,
                        trans_steps=200000, per_steps=1000, per_count=100):
    F_values = np.arange(0.1, max_F + delta_F, delta_F)
    x_vals = []

    for F in F_values:
        t = 0.0
        x = -0.5
        v = 0.5

        # Evoluindo o transiente
        for i in range(trans_steps):
            x, v, t = rk4(t, x, v, h, t + h, F, omega)

        # Coletando pontos após o transiente
        for i in range(per_count):
            for j in range(per_steps):
                x, v, t = rk4(t, x, v, h, t + h, F, omega)
            print(F, x)

            x_vals.append(x)

    x_vals = np.array(x_vals)

    plt.figure(figsize=(8, 6))
    for i, F in enumerate(F_values):
        plt.scatter([F] * per_count, x_vals[i * per_count: (i + 1) * per_count], s=0.1)
    plt.title("Diagrama de Bifurcação")
    plt.xlabel("F")
    plt.ylabel("x")
    plt.show()

bifurcation_diagram()
```

E a seguinte saída foi obtida:

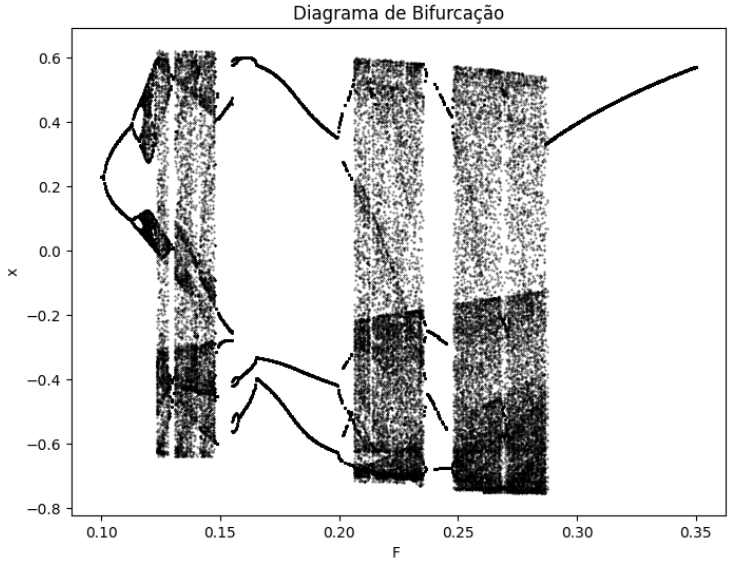

Figura 4: Diagrama de Bifurcação obtida para seções de Poincaré variando F e seus respectivos valores de x.

Para obtermos constante de Feigenbaum δ, limitando o eixo dos valores de x e os valores de F podemos observar mais claramente os pontos de bifurcação, como visto abaixo:

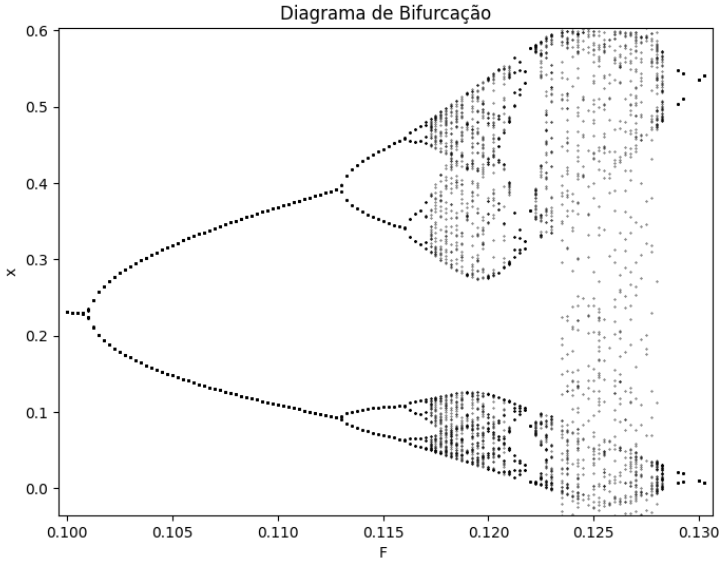

Figura 5: Diagrama de Bifurcação com eixos limitados para obtenção dos pontos de bifurcação.

A partir da figura acima, podemos observar que as bifurcações ocorrem em $F = 0.100749$, 0.113000 e 0.116002. Então temos que

$$\delta = \frac{F_3 - F_2}{F_2 - F_1} = \frac{0.100749 - 0.113000}{0.113000 - 0.116002} \approx 4.08094 \tag{3.1}$$

O valor nominal para a constante é de $\delta = 4.66920$, um erro em módulo de aproximadamente 0.6 em relação ao obtido.

Por fim, podemos obter o Mapa de Poincaré. Aplicando $F = 0.26$, obtemos a seguinte saída:

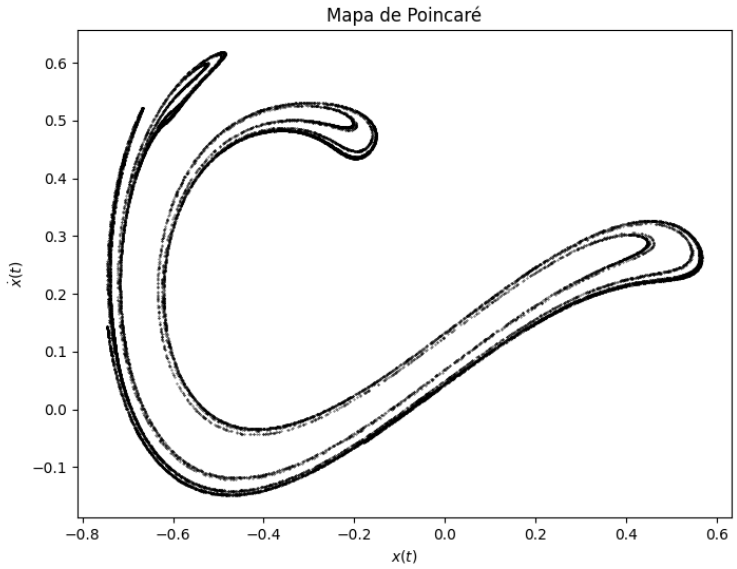

Figura 6: Mapa de Poincaré estabelecendo $F = 0.26$ e evoluindo i até 20000.