# Sistemas Dinâmicos: Simulações Numéricas e Visualizações em Python

Prof. Ana Isabel C.

June 23, 2025

# Sumário

Métodos Numéricos

# Sumário

# Sumário

# Sumário

# Sumário

# Sumário

# Métodos Numéricos

## Método de Euler

Para $\dot{x} = f(t, x)$, com passo $h$:

$$x_{n+1} = x_n + hf(t_n, x_n)$$

Simples, mas menos preciso.

# Métodos Numéricos

## Método de Euler

Para $\dot{x} = f(t, x)$, com passo $h$:

$$x_{n+1} = x_n + hf(t_n, x_n)$$

Simples, mas menos preciso.

## Runge-Kutta (RK4)

Mais preciso, usa quatro avaliações por passo:

$$k_1 = f(t_n, x_n), \quad k_2 = f(t_n + \frac{h}{2}, x_n + \frac{h}{2}k_1)$$

$$k_3 = f(t_n + \frac{h}{2}, x_n + \frac{h}{2}k_2), \quad k_4 = f(t_n + h, x_n + hk_3)$$

$$x_{n+1} = x_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

# Sumário

# Volatilidade Caótica em Mercados

## Modelo

Simule a volatilidade de retornos com o mapa logístico:

$$r_{n+1} = 3.9 r_n (1 - r_n), \quad r_n \in [0, 1]$$

Inspirado em flutuações de ativos como GOLL4.SA em 2020.

# Volatilidade Caótica em Mercados

## Modelo

Simule a volatilidade de retornos com o mapa logístico:

$$r_{n+1} = 3.9 r_n (1 - r_n), \quad r_n \in [0, 1]$$

Inspirado em flutuações de ativos como GOLL4.SA em 2020.

## Código Python

def logistic$_map(r, x0, n) : x = np.zeros(n) x[0] = x0 for i in range(n-1) : x[i+1] = r * x[i] * (1 - x[i]) return x$
r, x0, n = 3.9, 0.5, 50 x = logistic$_map(r, x0, n) plt.plot(range(n), x,$
'o-', color='0A2D50') plt.xlabel('Iterao (n)') plt.ylabel('Retorno ($r_n$)') plt.title('Volatilidade Catica') plt.grid(True)
plt.savefig('logistic$_volatility.png$ ', $dpi = 300$)

# Sumário

# Modelo SIR com RK4

## Modelo

Simule o modelo SIR (do Cap. 6):

$$\begin{cases} \dot{S} = -\beta S I \\ \dot{I} = \beta S I - \gamma I \\ \dot{R} = \gamma I \end{cases}$$

Usando RK4 com $\beta = 0.3$, $\gamma = 0.1$.

# Modelo SIR com RK4

## Modelo

Simule o modelo SIR (do Cap. 6):

$$\begin{cases} \dot{S} = -\beta S I \\ \dot{I} = \beta S I - \gamma I \\ \dot{R} = \gamma I \end{cases}$$

Usando RK4 com $\beta = 0.3$, $\gamma = 0.1$.

## Código Python

def sir$_model(t, state, beta, gamma) : S, I, R = state dS_dt = -beta * S * I dI_dt = beta * S * I - gamma * I dR_dt = gamma * I return np.array([dS_dt, dI_dt, dR_dt])$
def rk4(f, t0, tf, h, initial$_state, *args) : t = np.arange(t0, tf, h) n = len(t) states = np.zeros((n, len(initial_state))) states[0] = initial_state for i in range(n-1) : k1 = f(t[i], states[i], *args) k2 = f(t[i] + h/2, states[i] + h/2 * k1, *args) k3 = f(t[i] + h/2, states[i] + h/2 * k2, *args) k4 = f(t[i] + h, states[i] + h * k3, *args) states[i+1] = states[i] + h/6 * (k1 + 2 * k2 + 2 * k3 + k4) return t, states$
t, states = rk4(sir$_model, 0, 50, 0.1, [0.99, 0.01, 0], 0.3, 0.1) plt.plot(t, states[:, 0], label =$ 'S', color='0A2D50') plt.plot(t, states[:, 1], label='I', color='FFCC00') plt.plot(t, states[:, 2], label='R', color='323232') plt.xlabel('Tempo (t)')

# Sumário

# Visualizações em Python

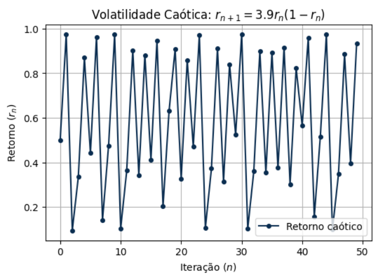
### Resultados

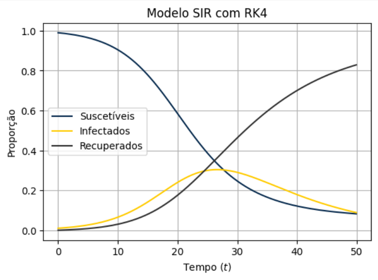
Gráficos gerados para o mapa logístico e o modelo SIR:

# Visualizações em Python

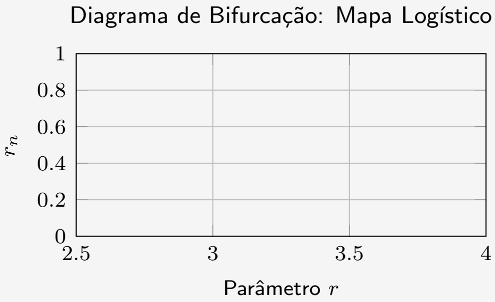
## Resultados

Gráficos gerados para o mapa logístico e o modelo SIR:

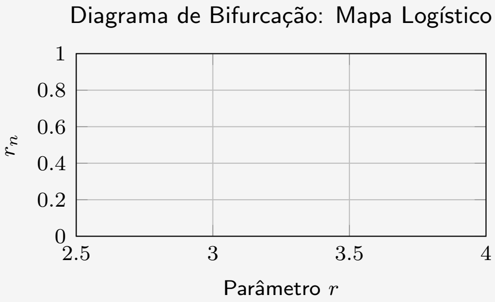


# Diagrama de Bifurcação



# Diagrama de Bifurcação




## Interpretação

O diagrama mostra a transição para o caos em $r = 3.9$, refletindo a imprevisibilidade de retornos financeiros.

# Sumário

# Conclusão

## Resumo

- Métodos numéricos (Euler, RK4) resolvem sistemas dinâmicos com precisão.
- Simulações em Python visualizam caos financeiro e dinâmicas biológicas.
- Aplicações reais: Previsão de volatilidade e epidemias.

# Conclusão

## Resumo

- Métodos numéricos (Euler, RK4) resolvem sistemas dinâmicos com precisão.
- Simulações em Python visualizam caos financeiro e dinâmicas biológicas.
- Aplicações reais: Previsão de volatilidade e epidemias.

### Próxima Sessão

Continue explorando sistemas dinâmicos no repositório e aplique em projetos reais!

Visite o repositório!