Universidade Estadual de Campinas

Departamento de Matemática Aplicada

Mestrado Profissional em Matemática Aplicada e Computacional

Cálculo Fracionário aplicado no Sistema de Lotka-Volterra: Uma análise didática e computacional

Aluno(s): Diogo Takamori Barbosa - RA 037382 e Framilson José Ferreira Carneiro - RA 230113

Professor orientador: Junior Cesar Alves Soares

Novembro

2023

Universidade Estadual de Campinas

Departamento de Matemática Aplicada

Mestrado Profissional em Matemática Aplicada e Computacional

Cálculo Fracionário aplicado no Sistema de Lotka-Volterra: Uma análise didática e computacional

Primeiro Relatório de Pesquisa apresentado ao Professor Junior Cesar Alves Soares do Curso de Mestrado Profissional em Matemática Aplicada e Computacional da Universidade Estadual de Campinas, como requisito parcial para nota da disciplina PM009 - Tópicos em Matemática I - Introdução ao Cálculo Fracionário e Aplicações.

Aluno(s): Diogo Takamori Barbosa - RA 037382 e Framilson José Ferreira Carneiro - RA 230113

Professor orientador: Junior Cesar Alves Soares

Novembro

2023

Conteúdo

1 Resumo

O trabalho analisado busca construir uma extensão do sistema de Lotka-Volterra a fim de incorporar derivadas de ordem não inteira. Nesse ínterim, tem-se como objetivo principal dessa generalização aprimorar a descrição do fenômeno de maneira semelhante ao que foi realizado em outros trabalhos da literatura da área. O modelo clássico que descreve as interações entre presa e predador, conhecido como "sistema de Lotka-Volterra", é abordado com duas derivadas de ordem inteira. Por meio de uma técnica de linearização, intenta-se obter uma solução em termos dos parâmetros constantes. Além disso, apresenta-se uma solução para o sistema assim denominado "Lotka-Volterra fracionário", que consiste em duas equações diferenciais não lineares com derivadas de ordem menor que a unidade. Tal solução é expressa em termos da função de Mittag-Leffler, por intermédio da aplicação do conceito de diferenças finitas, paralelamente à técnica de linearização utilizada.

2 Modelagem Sistema Clássico Lotka-Volterra em Pythom

O modelo Lotka-Volterra é uma ferramenta matemática que descreve a dinâmica de duas populações em um ecossistema: as presas e os predadores. As populações interagem entre si, e essas interações têm um impacto significativo na dinâmica das populações ao longo do tempo. As equações diferenciais que compõem o modelo permitem entender como a abundância de presas e predadores evolui e oscila no decorrer das estações e anos.

O modelo Lotka-Volterra é um exemplo clássico de um sistema dinâmico que pode ser aplicado em uma ampla gama de contextos, não apenas na ecologia, mas também em campos como economia, epidemiologia e engenharia. Essa versatilidade torna o entendimento do modelo e sua implementação prática em linguagens de programação, como Python, uma habilidade valiosa.

2.1 Modelos Reais e Suas Abordagens:

Neste artigo, o modelo Lotka-Volterra em cenários do mundo real. Alguns exemplos notáveis incluem:

Dinâmica de Populações de Presas e Predadores: O modelo Lotka-Volterra tem sido usado para entender as oscilações nas populações de lebres e linces, lobos e alces, e outros exemplos de predadores e suas presas na natureza. Esses estudos têm contribuído para a conservação de espécies em risco.

Economia: O modelo encontra aplicação na economia para descrever a competição entre empresas e consumidores. Pode ser usado para analisar a relação entre o estoque de recursos naturais, como peixes, e a pesca comercial.

Epidemiologia: Na epidemiologia, o modelo pode ser usado para modelar a propagação de doenças infecciosas. As presas podem representar indivíduos suscetíveis, e os predadores podem representar indivíduos infectados.

Cada uma dessas aplicações requer adaptações do modelo Lotka-Volterra, levando em consideração as dinâmicas específicas das populações e das interações em questão. Esses modelos adaptados são fundamentais para entender as dinâmicas de sistemas complexos.

2.2 Abordagem Computacional com Python:

Uma das vantagens do modelo Lotka-Volterra é a sua implementação computacional. Python é uma linguagem de programação amplamente utilizada para modelagem e simulação de sistemas ecológicos. As vantagens da implementação computacional do modelo Lotka-Volterra incluem simplicidade e legibilidade, o que torna a implementação das equações Lotka-Volterra mais acessível, mesmo para aqueles que não têm uma formação matemática avançada. Python oferece uma ampla gama de bibliotecas e ferramentas para análise de dados, simulação e visualização. Capacidade de criar gráficos e visualizações. Isso permite que se observem as oscilações das populações de presas e predadores ao longo do tempo, tornando a interpretação dos resultados mais clara. Com o uso de Python, é possível realizar análises de sensibilidade para entender como variações nos parâmetros afetam as dinâmicas do sistema. Isso é fundamental para prever o comportamento de populações em resposta a diferentes condições, permitindo a integração de dados reais, como dados de campo e séries temporais de populações, para validar e ajustar os modelos Lotka-Volterra. Isso torna o modelo mais robusto e aplicável a cenários do mundo real.

2.3 Modelo Matemático Lotka-Volterra para Dinâmica de Populações de Presas e Predadores

O modelo Lotka-Volterra descreve a interação entre duas espécies, no caso, coelhos (x) e raposas (y). As equações diferenciais do modelo são dadas por:

$$\frac{dx}{dt} = \alpha x - \beta xy$$
$$\frac{dy}{dt} = \delta xy - \gamma y$$

onde:

- x é a população de coelhos;
- y é a população de raposas;
- α é a taxa de crescimento dos coelhos na ausência de predadores;
- β é a taxa de predação dos coelhos pelas raposas;
- γ é a taxa de morte das raposas na ausência de presas;
- δ é a taxa de crescimento das raposas devido à predação dos coelhos.

A simulação numérica do modelo usando o método de Euler é feita pelas seguintes equações de atualização:

$$x_{i+1} = x_i + \Delta t \cdot (\alpha x_i - \beta x_i y_i)$$
$$y_{i+1} = y_i + \Delta t \cdot (\delta x_i y_i - \gamma y_i)$$

onde: i é o índice da iteração;

Δt é o intervalo de tempo;

x_i , y_i são as populações de coelhos e raposas no passo i, respectivamente.

2.4 Simulação Computacional em Python:

O Código em Python demonstra a evolução das populações de presas e predadores ao longo do tempo e plota um gráfico para visualização:

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

# Parâmetros do modelo Lotka-Volterra
alpha = 0.1 # Taxa de crescimento dos coelhos na ausência de predadores
beta = 0.02 # Taxa de predação dos coelhos pelas raposas
gamma = 0.1 # Taxa de morte das raposas na ausência de presas
delta = 0.01 # Taxa de crescimento das raposas devido à predação dos coelhos

# Condições iniciais
x0 = 40 # População inicial de coelhos
y0 = 9 # População inicial de raposas

# Configuração do tempo
T = 200
dt = 0.1
num_steps = int(T / dt)
```

```
# Arrays para armazenar resultados
x_values = np.zeros(num_steps)
y_values = np.zeros(num_steps)
time_values = np.zeros(num_steps)

# Inicialização das populações iniciais
x = x0
y = y0

# Simulação do modelo Lotka-Volterra usando o método de Euler
for i in range(num_steps):
x_values[i] = x
y_values[i] = y
time_values[i] = i * dt

# Equações de Lotka-Volterra usando o método de Euler
dx = dt * (alpha * x - beta * x * y)
dy = dt * (delta * x * y - gamma * y)

# Atualização das populações
x += dx
y += dy
# Plotagem do gráfico bidimensional
plt.figure(figsize=(10, 6))
plt.plot(x_values, y_values, label=’Dinâmica Populacional’)
plt.title(’Modelo Lotka-Volterra: Sistema Dinâmico Bidimensional’)
plt.xlabel(’População de Coelhos’)
plt.ylabel(’População de Raposas’)
plt.legend()
plt.grid(True)
plt.show()
```

Comentários sobre o código:

Parâmetros do Modelo: Os parâmetros alpha, beta, gamma e delta representam as taxas de crescimento e predação das populações de coelhos e raposas no modelo Lotka-Volterra.

As populações iniciais de coelhos (x0 = 40) e raposas (y0 = 9) são definidas para iniciar a simulação.

Define o tempo total da simulação (T = 200), o intervalo de tempo (dt = 0.1) e calcula o número total de passos de tempo (num_steps).

Arrays x_values, y_values e time_values são inicializados para armazenar os resultados da simulação.

As populações iniciais são atribuídas às variáveis x e y.

As equações de Lotka-Volterra são resolvidas usando o método de Euler para cada passo de tempo.

As populações de coelhos e raposas são atualizadas com base nas derivadas calculadas.

Um gráfico bidimensional é criado para visualizar a dinâmica populacional ao longo do tempo, mostrando as populações de coelhos e raposas. Este código Python realiza a simulação da dinâmica de populações de presas e predadores ao longo do tempo com base no modelo Lotka-Volterra e plota um gráfico que demonstra as oscilações características entre as duas populações.

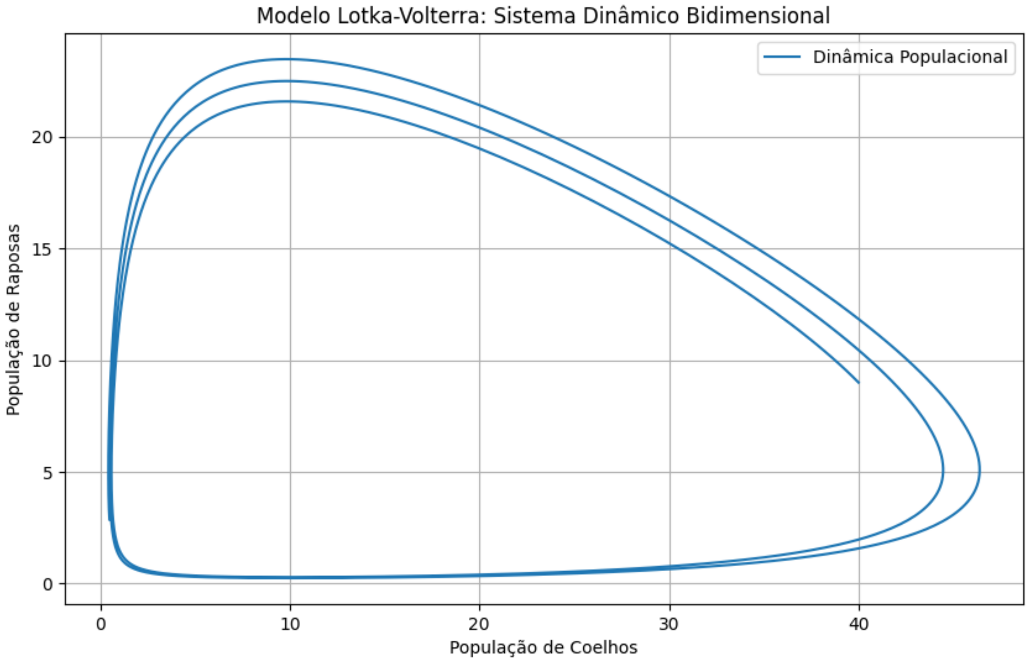

Figura 1: Plotagem do gráfico gerado atravéz da relação de entre a população de coelhos e raposas no sistema Lotka-Volterra

2.5 Cálculo Matemático do Código apresentado

O modelo Lotka-Volterra descreve a interação entre duas espécies, no seu caso, coelhos (x) e raposas (y). As equações diferenciais do modelo são dadas por:

Definição das Equações Diferenciais do Modelo:

$$\frac{dx}{dt} = \alpha x - \beta xy$$
$$\frac{dy}{dt} = \delta xy - \gamma y$$

Inicialização das Variáveis e Parâmetros:

- $\alpha = 0.1$,
- $\beta = 0.02$,
- $\gamma = 0.1$,
- $\delta = 0.01$
- $x_0 = 40$,

- $y_0 = 9$
- $T = 200$
- $\Delta t = 0.1$

Número de passos: num_steps $= \frac{T}{\Delta t}$
Inicialização das Arrays e Condições Iniciais:

- $x_{\text{values}}[0] = x_0,$
- $y_{\text{values}}[0] = y_0,$
- time_values$[0] = 0$

Simulação do Modelo usando o Método de Euler:

$$x_{i+1} = x_i + \Delta t \cdot (\alpha x_i - \beta x_i y_i)$$
$$y_{i+1} = y_i + \Delta t \cdot (\delta x_i y_i - \gamma y_i)$$

Para cada iteração i do código :
Calculado $dx = \Delta t \cdot (\alpha x - \beta xy)$
Calculado $dy = \Delta t \cdot (\delta xy - \gamma y)$
Atualizar x e y usando:
$x+ = dx$
e
$y+ = dy$
Plotagem do Gráfico Bidimensional:

- População de Coelhos (x) no eixo x
- População de Raposas (y) no eixo y

2.6 Gráfico em relação ao tempo

Podemos usar outro código para plotar um gráfico em função do tempo modificando a forma de plotar o gráfico. Demonstrando a populaç eos no eixo Y e o tempo no eixo X

Para esse caso alteramos ao código referente a plotagem do gráfico, ficando da seguinte form o código em Python:

———

```
# Plotagem
plt.figure(figsize=(10, 6))
plt.plot(time_values, x_values, label='Coelhos')
plt.plot(time_values, y_values, label='Raposas')
plt.title('Dinâmica Populacional: Modelo Lotka-Volterra')
plt.xlabel('Tempo')
plt.ylabel('População')
plt.legend()
plt.grid(True)
plt.show()
```

——— Plotagem: A dinâmica populacional é plotada ao longo do tempo para as populações de coelhos e raposas, com rótulos e legendas para melhor compreensão.

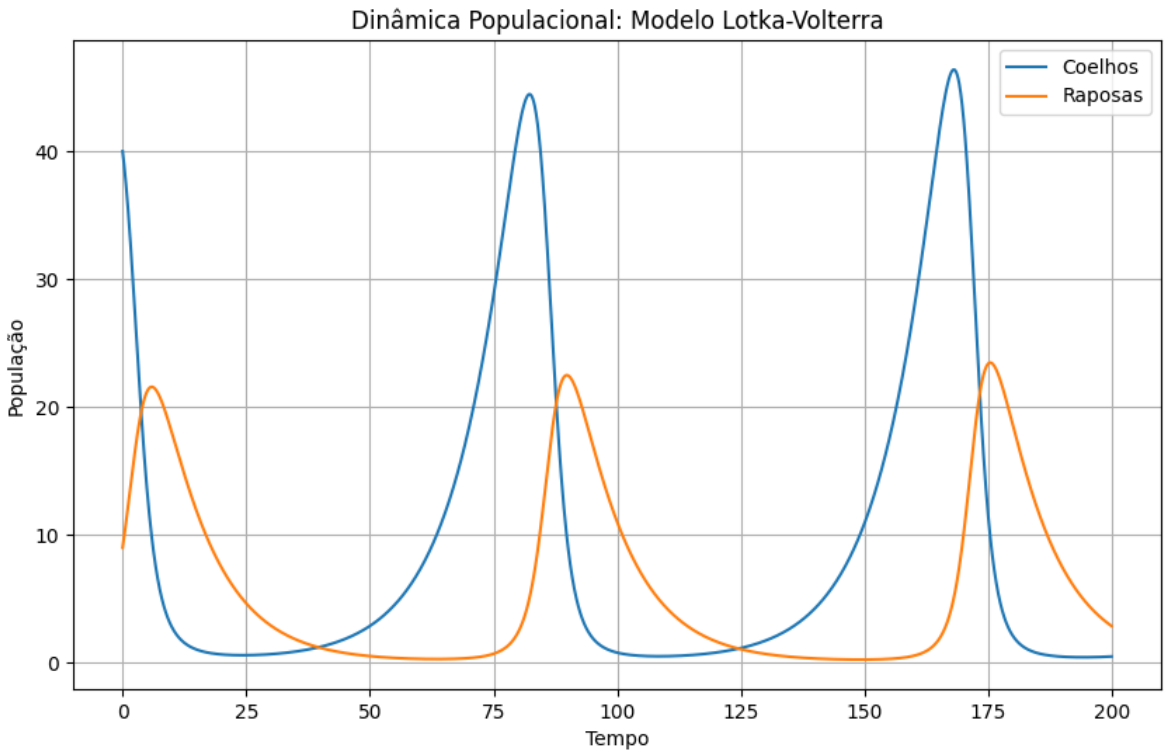

Figura 2: Plotagem do gráfico gerado atravéz da relação de entre a população de coelhos e raposas em relação ao tempo no sistema Lotka-Volterra

3 Modelo Matemático Lotka-Volterra para Dinâmica de Populações de Presas e Predadores com Cálculo Fracionário

Equações Diferenciais Fracionárias (EDFs) são uma generalização das Equações Diferenciais Ordinárias (EDOs), onde a ordem da derivada ou integral é um número não inteiro. Em outras palavras, elas envolvem derivadas ou integrais de ordens fracionárias. A EDF mais comum é a Equação Diferencial Fracionária (EDF) de Caputo, que é uma generalização da derivada comum para ordens não inteiras.

A forma geral de uma EDF de Caputo de ordem q para uma função $y(t)$ é dada por:

$$D_t^q y(t) = f(t, y(t), D_t^{\lceil q \rceil - 1} y(t), D_t^{\lceil q \rceil - 2} y(t), \ldots, y'(t), y(t))$$

onde D_t^q representa a derivada de ordem fracionária de Caputo, f é uma função que descreve a dinâmica do sistema, e $\lceil q \rceil$ denota o arredondamento para cima de q para o inteiro mais próximo.

As EDFs podem modelar fenômenos complexos em sistemas físicos, biológicos e outros, onde comportamentos não lineares e memória de longo prazo são relevantes. O cálculo fracionário oferece uma ferramenta matemática poderosa para descrever esses fenômenos.

Para resolver EDFs numericamente, podem ser usados métodos específicos, como o método de diferenças finitas fracionárias, a transformada de Laplace fracionária ou outros métodos adaptativos.

Lidar com Equações Diferenciais Fracionárias (EDFs) pode ser mais complexo do que lidar com EDOs tradicionais. Em Python pode-se recorrer a outras abordagens e bibliotecas especializadas para lidar com cálculos fracionários. Vamos apresentar uma abordagem geral usando o método de diferenças finitas para a integração numérica de EDFs que permitem uma abordagem matemática para o modelo Lotka-Volterra. A resolução de um modelo Lotka-Volterra por meio de Equações Diferenciais Fracionárias (EDFs) pode ser feita usando métodos específicos para cálculo fracionário. Vamos apresentar uma abordagem usando o método de diferenças finitas fracionárias. O modelo reformula Lotka-Volterra em termos de EDFs. O sistema de EDFs de Caputo para o modelo Lotka-Volterra, As equações Lotka-Volterra com derivadas fracionárias são representadas por:

$$D_t^q R(t) = \alpha R(t) - \beta R(t) F(t)$$

$$D_t^q F(t) = -\gamma F(t) + \delta R(t) F(t)$$

Aqui, D_t^q representa a derivada de Caputo de ordem fracionária.

Vamos derivar as equações que serão utilizadas no código para o modelo Lotka-Volterra com derivadas fracionárias usando o método de diferenças finitas fracionárias.

O modelo Lotka-Volterra com derivadas fracionárias é representado por:

$$\frac{dR}{dt} = \alpha R - \beta RF + \delta \mathcal{D}_q[R]$$
$$\frac{dF}{dt} = -\gamma F + \delta RF + \delta \mathcal{D}_q[F]$$

onde $\mathcal{D}_q$ representa a derivada fracionária e q é a ordem da derivada fracionária.

A derivada fracionária $\mathcal{D}_q$ é aproximada usando o método de diferenças finitas fracionárias, como será demonstrado no código pela função '$fractional_difference$':

$$\mathcal{D}_q[Y] = (1-q) \cdot Y_i + q \cdot Y_{i-1}$$

onde Y pode ser R ou F.

Agora, vamos realizar os cálculos dentro do loop de atualização populacional:

Cálculo de $\frac{dR}{dt}$:

$$\frac{dR}{dt} = \alpha R - \beta RF + \delta \mathcal{D}_q[R]$$

Substituindo a expressão para $\mathcal{D}_q[R]$:

$$\frac{dR}{dt} = \alpha R - \beta RF + \delta \left((1-q) \cdot R_i + q \cdot R_{i-1}\right)$$

Cálculo de $\frac{dF}{dt}$:

$$\frac{dF}{dt} = -\gamma F + \delta RF + \delta \mathcal{D}_q[F]$$

Substituindo a expressão para $\mathcal{D}_q[F]$:

$$\frac{dF}{dt} = -\gamma F + \delta RF + \delta\left((1-q)\cdot F_i + q\cdot F_{i-1}\right)$$

Atualização das Populações: Utilizando o método de diferenças finitas para atualizar R e F:

$$R_{i+1} = R_i + \Delta t\cdot\frac{dR}{dt}$$
$$F_{i+1} = F_i + \Delta t\cdot\frac{dF}{dt}$$

Esses cálculos são iterados ao longo do loop para simular a dinâmica populacional ao longo do tempo. O resultado é então plotado no gráfico final.

3.1 Modelo em Python Simulação da Dinâmica Populacional

A seguir, representaremos um código simplificado em Python para calcular a derivada de Caputo e o método de diferenças finitas fracionárias para a integração numérica.

O código em Python fica da seguinte forma:

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


def fractional_difference(y, alpha, dt):
"
Calcula a diferença fracionária usando o método de diferenças finitas fracionárias.
Parâmetros:
- y: Lista contendo os valores da série temporal.
- alpha: Ordem fracionária para a derivada.
- dt: Incremento de tempo.
Retorna:
- Valor fracionário calculado.
" if len(y) >= 2:
# Se houver pelo menos dois elementos em y, calcula a diferença fracionária.
return (1 - alpha) * y[-1] + alpha * y[-2]
else:
# Se houver menos de dois elementos, retorna o último elemento.
return y[-1]
 # Parâmetros do modelo Lotka-Volterra
alpha = 0.1 # Taxa de crescimento das presas na ausência de predadores
beta = 0.02 # Taxa de predação (interação entre presas e predadores)
gamma = 0.1 # Taxa de diminuição dos predadores na ausência de presas
delta = 0.01 # Taxa de crescimento dos predadores em função das presas
q = 0.5 # Ordem fracionária para a derivada (pode ser ajustada conforme necessário)
 # Condições iniciais
R0 = 40 # População inicial de coelhos (presas)
F0 = 9 # População inicial de raposas (predadores)
```

Configuração do tempo

t_max = 200 # Tempo máximo de simulação

dt = 0.1 # Incremento de tempo (passo)

time_points = np.arange(0, t_max, dt) # Lista de pontos temporais

Inicialização das populações

R = np.zeros(len(time_points)) # Lista para armazenar a população de coelhos

F = np.zeros(len(time_points)) # Lista para armazenar a população de raposas

R[0] = R0 # População inicial de coelhos

F[0] = F0 # População inicial de raposas

Método de diferenças finitas fracionárias para resolver EDFs

for i in range(1, len(time_points)):

Equações Lotka-Volterra discretizadas com derivadas fracionárias

dRdt = alpha * R[i-1] - beta * R[i-1] * F[i-1] + delta * fractional_difference(R[:i], q, dt)

dFdt = -gamma * F[i-1] + delta * R[i-1] * F[i-1] + delta * fractional_difference(F[:i], q, dt)

Atualização das populações usando o método de diferenças finitas fracionárias

R[i] = R[i-1] + dt * dRdt

F[i] = F[i-1] + dt * dFdt

Criação de campos vetoriais

R_vec = np.linspace(min(R), max(R), 20)

F_vec = np.linspace(min(F), max(F), 20)

R_grid, F_grid = np.meshgrid(R_vec, F_vec)

dRdt_grid = alpha * R_grid - beta * R_grid * F_grid

dFdt_grid = -gamma * F_grid + delta * R_grid * F_grid # Normalizando os vetores para melhor visualização

magnitude = np.sqrt(dRdt_grid**2 + dFdt_grid**2)

dRdt_grid /= magnitude

dFdt_grid /= magnitude

Plotagem do gráfico bidimensional com campos vetoriais

plt.figure(figsize=(12, 6))

Plotagem do campo vetorial

plt.quiver(R_grid, F_grid, dRdt_grid, dFdt_grid, scale=40, color='pink')

Plotagem da evolução temporal de Coelhos e Raposas

plt.plot(R, F, label='Dinâmica Populacional', color='blue')

plt.title('Modelo Lotka-Volterra: Sistema Dinâmico Bidimensional com Campos Vetoriais')

plt.xlabel('População de Coelhos')

plt.ylabel('População de Raposas')

plt.legend()

plt.grid(True)

plt.show()

———

Neste código, as equações do modelo Lotka-Volterra são discretizadas usando o método de diferenças finitas para aproximar as derivadas. O resultado é uma simulação temporal das populações de coelhos e raposas ao longo do tempo. O gráfico final mostra como as populações evoluem de acordo com o modelo. O código fornecido implementa o modelo Lotka-Volterra com a adição de cálculos fracionários, representados pela ordem fracionária q . As equações diferenciais discretizadas usando o método de diferenças finitas são:

$$R_{i+1} = R_i + \Delta t \cdot (\alpha R_i - \beta R_i F_i)$$
$$F_{i+1} = F_i + \Delta t \cdot (-\gamma F_i + \delta R_i F_i)$$

onde i é o índice da iteração, Δt é o intervalo de tempo, e R_i, F_i são as populações de coelhos e raposas no passo i, respectivamente.

3.2 Cálculos Matemáticos:

Vamos demonstrar os cálculos para o modelo Lotka-Volterra com derivadas fracionárias usando o método de diferenças finitas fracionárias, com os parâmetros fornecidos no código:

$$\alpha = 0.1$$

$$\beta = 0.02$$

$$\gamma = 0.1$$

$$\delta = 0.01$$

$$q = 0.5 \quad \text{(ordem fracionária)}$$

Condições Iniciais:

$R_0 = 40$ e $F_0 = 9$.

Tempo e Passos:

- $t_{\max} = 200$;

- $\Delta t = 0.1$;

- time_points = np.arange$(0, t_{\max}, \Delta t)$.

Inicialização das Populações:

- $R[0] = R_0$ e $F[0] = F_0$

Método de Diferenças Finitas:

Para cada iteração i:

- $R_{i+1} = R_i + \Delta t \cdot (\alpha R_i - \beta R_i F_i)$;

- $F_{i+1} = F_i + \Delta t \cdot (-\gamma F_i + \delta R_i F_i)$.

Cálculo de $\frac{dR}{dt}$:

$$\frac{dR}{dt} = \alpha R - \beta RF + \delta \left((1-q) \cdot R_i + q \cdot R_{i-1}\right)$$

Cálculo de $\frac{dF}{dt}$:

$$\frac{dF}{dt} = -\gamma F + \delta RF + \delta \left((1-q) \cdot F_i + q \cdot F_{i-1}\right)$$

Atualização das Populações:

$$R_{i+1} = R_i + \Delta t \cdot \frac{dR}{dt}$$
$$F_{i+1} = F_i + \Delta t \cdot \frac{dF}{dt}$$

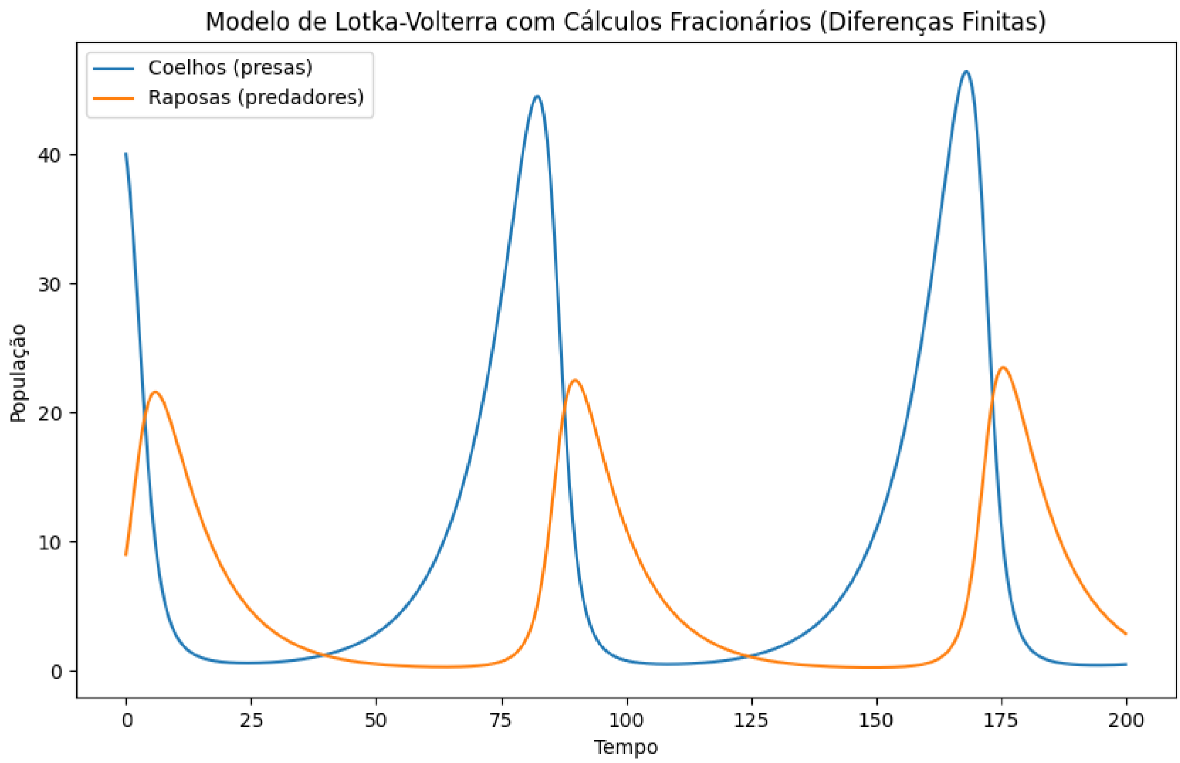

Figura 3: Enter Caption

Agora, vamos iterar esses cálculos para simular a dinâmica populacional ao longo do tempo. Considere os seguintes valores iniciais:

$$R_0 = 40$$

$$F_0 = 9$$

e um incremento de tempo $\Delta t = 0.1$ com um tempo máximo de simulação $t_{\max} = 200$.

Esses cálculos e simulações são realizados para cada ponto no tempo, atualizando as populações de coelhos e raposas de acordo com o modelo Lotka-Volterra com derivadas fracionárias. O gráfico final mostra a evolução das populações ao longo do tempo.

Plotagem dos Resultados:

- Gráfico das populações de coelhos e raposas ao longo do tempo.

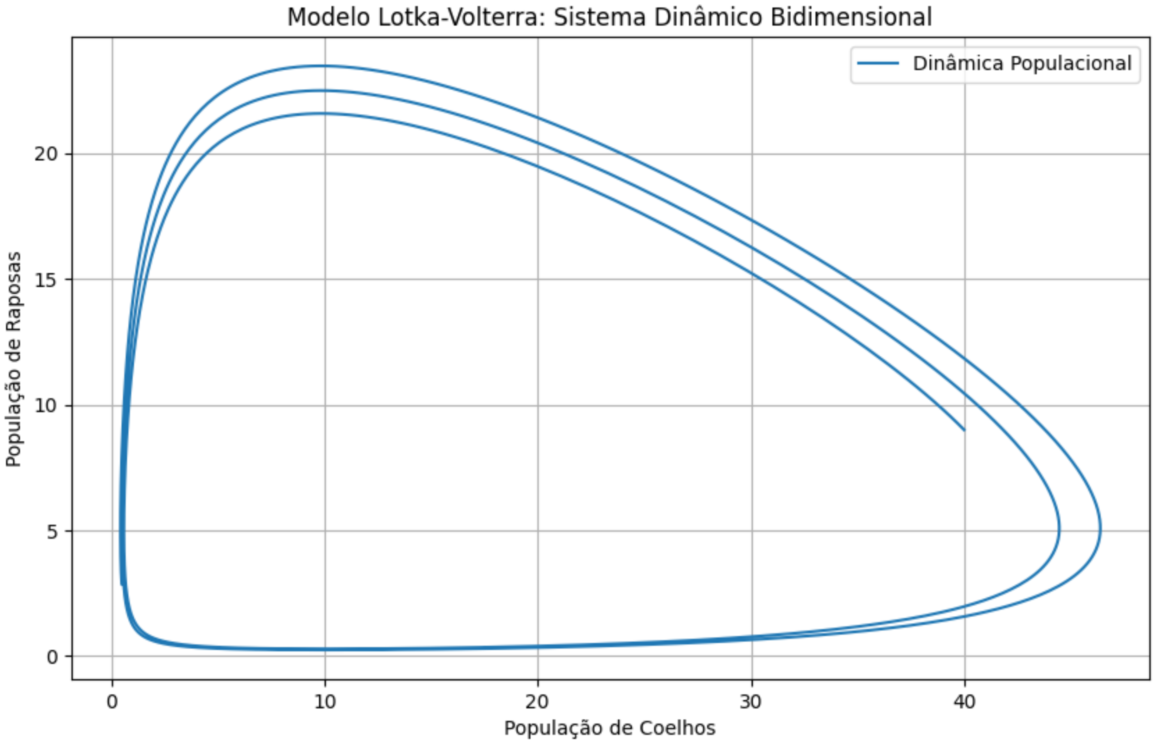

Figura 4: Enter Caption

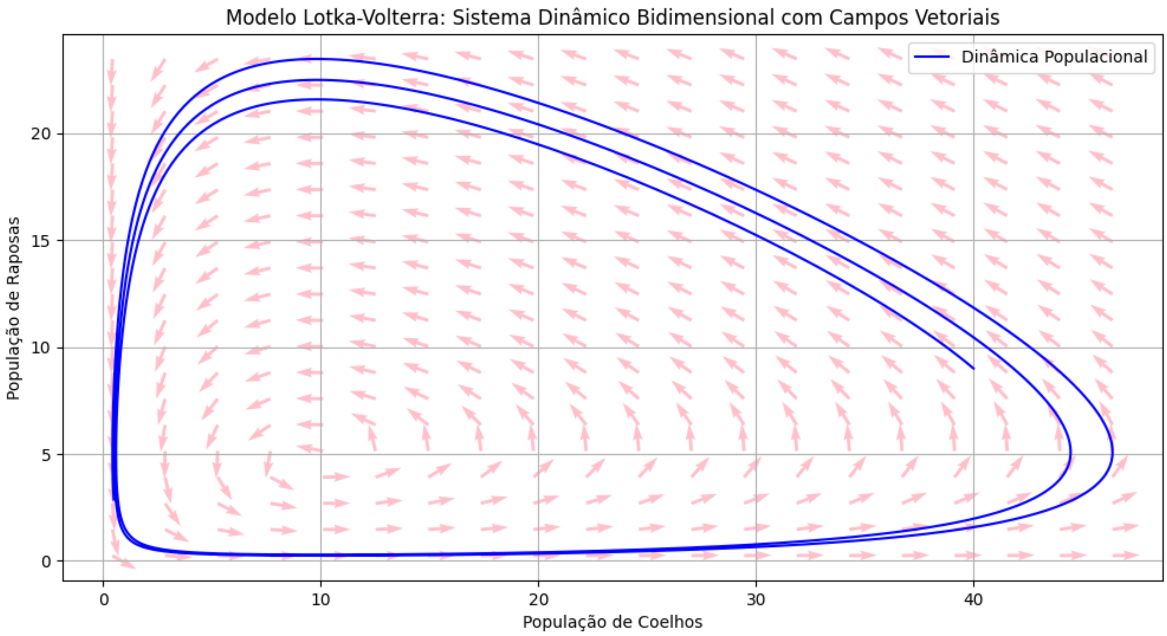

Figura 5: Enter Caption

4 Conclusão

A introdução da função fractional_difference(código Python), ou Calculo Fracionário, no cálculo das derivadas fracionárias representa uma abordagem alternativa para incorporar efeitos fracionários no modelo Lotka-Volterra em comparação com o cálculo por Equações Diferenciais Ordinárias (EDOs) padrão. A diferença fundamental está na forma como as derivadas fracionárias são tratadas. O método de diferenças finitas fracionárias é uma técnica numérica que discretiza a derivada fracionária usando diferenças finitas, levando em consideração a ordem fracionária q. Isso pode ser especialmente útil, quando se lida com sistemas complexos ou comportamentos não lineares que podem ser capturados de maneira mais precisa por derivadas fracionárias.

Em contraste, o cálculo por EDO padrão utiliza derivadas ordinárias, que assumem que as mudanças nas variáveis são proporcionais às próprias variáveis. Para modelos mais simples ou situações em que a linearidade é uma boa aproximação, as EDOs podem ser suficientes e são geralmente mais simples de resolver analiticamente.

O efeito específico da função fractional_difference no modelo Lotka-Volterra dependerá do valor escolhido para a ordem fracionária q e das características específicas do sistema que está sendo modelado. Experimentar diferentes valores de q pode ajudar a entender como as derivadas fracionárias afetam a dinâmica do sistema.

Portanto, a introdução da função fractional_difference, ou do Cálculo Fracionário traz uma abordagem numérica para incorporar derivadas fracionárias, permitindo maior flexibilidade na modelagem de sistemas dinâmicos complexos em comparação com a abordagem tradicional de EDOs.

Bibliografia

1 - CAPUTO, M. Linear Model of Dissipation Whose Q is Almost Frequency Independent – II, Geophys. J. R. Astron. Soc., 13, 529-539, (1967).

2 - CAMARGO, R. F. Cálculo Integro diferencial de Ordem Arbitrária. Tese de doutorado em Matemática, 2008.

3 - LORENZO, C. F., HARTLEY, T. T. Initialized Fractional Calculus, NASA/PT-2000- 209943, 2000.

4 - OLIVEIRA, E. C. The Green's Function and the Lotka-Volterra System, 2007.

5 - OLIVEIRA, E. C., RODRIGUES, W. A. Funções Analíticas com Aplicações. São Paulo: Editora Livraria da Física, 2005.

6 - Python Software Foundation. (https://www.python.org/).

7 - Murray, J.D. (2002). Mathematical Biology I: An Introduction. Springer.

8 - Strogatz, S.H. (2018). Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering. CRC Press.

9 - Soetaert, K., Petzoldt, T., Setzer, R.W. (2010). Solving Differential Equations in R: Package deSolve. Journal of Statistical Software, 33(9), 1-25.

10 - Podlubny, I. (1999). "Fractional Differential Equations: An Introduction to Fractional Derivatives, Fractional Differential Equations, to Methods of Their Solution and Some of Their Applications."

11 - Diethelm, K., Ford, N. J. (2002). "Analysis of fractional differential equations."Journal of Mathematical Analysis and Applications, 265(2), 229-248.

12 - Podlubny, I., Chechkin, A., Skovranek, T., Chen, Y. Q., Jara, B. M. (1999). "Matrix approach to discrete fractional calculus."Fractional Calculus and Applied Analysis, 2(3), 359-386.