

Differentiated Problem Solving

Sprint 4 - Passa Bola

VORTECH

FIAP - 2025

Nome Completo	R.M.	E-mail
Luara Martins de Oliveira Ramos	565573	rm565573@fiap.com.br
Kaio Victtor Santos Andrade Galvão	566536	rm566536@fiap.com.br
Jean Pierre Andrade Feltran	566534	rm566534@fiap.com.br

Desempenho de atletas e o engajamento digital no futebol feminino

Neste projeto de modelagem matemática, o foco é o futebol feminino, um esporte em constante crescimento e com cada vez mais visibilidade. Através da matemática, buscamos construir um protótipo conceitual para demonstrar como o desempenho físico das atletas em campo se relaciona com seu engajamento digital nas redes sociais.

Os conceitos de funções, limites e derivadas serão aplicados como ferramentas analíticas para compreender e quantificar diferentes aspectos do esporte. Vamos modelar o crescimento de seguidores de uma jogadora, projetar a saturação desse engajamento e calcular a velocidade instantânea das atletas, revelando a importância dessas ferramentas para a análise esportiva.

Este trabalho visa não apenas aplicar a teoria matemática, mas também mostrar como a ciência de dados pode auxiliar na tomada de decisões estratégicas, desde o marketing pessoal de uma atleta até a otimização de seu desempenho em campo.

1. Funções - Crescimento de Seguidores de uma Jogadora

Nesta seção, o objetivo é utilizar uma função exponencial para modelar o crescimento de uma jogadora nas redes sociais. Esse tipo de função é ideal para representar fenômenos que crescem a uma taxa constante, como a popularidade de um atleta em ascensão.

A Função Exponencial

A função proposta é:

$$f(t) = 5000 \cdot (1,08)^t$$

- $f(t)$ representa o número de seguidores da jogadora em um determinado momento.
- t representa o tempo, medido em semanas.
- 5000 é o número inicial de seguidores da jogadora (quando $t = 0$).
- 1,08 é a taxa de crescimento semanal. O valor de 1,08 indica que o número de seguidores aumenta em 8% a cada semana ($1 + 0,08$).

Exemplo Prático: Projeção de Seguidores

Para ilustrar o uso da função, podemos calcular o número de seguidores em diferentes momentos:

- **Semana 0 (início):**

$$f(0) = 5000 \cdot (1,08)^0 = 5000 \cdot 1 = 5000$$

- **Semana 5:**

$$f(5) = 5000 \cdot (1,08)^5 \approx 5000 \cdot 1,47 \approx 7350$$

- **Semana 10:**

$$f(10) = 5000 \cdot (1,08)^{10} \approx 5000 \cdot 2,16 \approx 10800$$

Gráfico do Crescimento de Seguidores

Para visualizar o comportamento da função de crescimento de seguidores, utilizamos o seguinte código Python. Ele gera um gráfico de linha, que é a melhor forma de representar a progressão exponencial ao longo das semanas.

O gráfico mostra como o número de seguidores, que começa em 5.000, aumenta de forma acelerada, reforçando a natureza da função exponencial.

```
In [3]: import numpy as np
import matplotlib.pyplot as plt

# Define the function for follower growth
def follower_growth(t):
    return 5000 * (1.08)**t

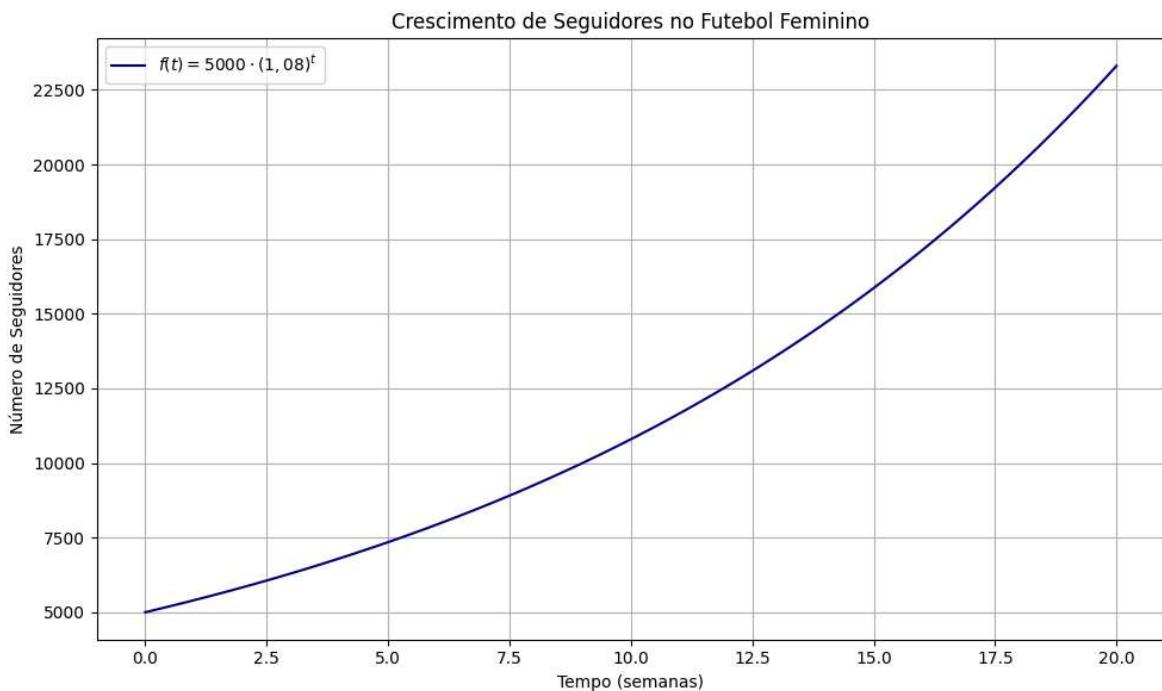
# Generate values for time (t) in weeks
t_values = np.linspace(0, 20, 100)

# Calculate the number of followers (f(t)) for each week
followers = follower_growth(t_values)

# Create the plot
plt.figure(figsize=(10, 6))
plt.plot(t_values, followers, label=r'$f(t) = 5000 \cdot (1,08)^t$', color='navy')
```

```
# Add title and labels
plt.title('Crescimento de Seguidores no Futebol Feminino')
plt.xlabel('Tempo (semanas)')
plt.ylabel('Número de Seguidores')
plt.grid(True)
plt.legend()
plt.tight_layout()

# Save the plot
plt.savefig('crescimento_seguidores.png')
```



2. Limites - Saturação do Engajamento

Aqui usamos o **limite** para mostrar que o crescimento da nossa jogadora nas redes sociais não dura para sempre. Embora a função que usamos para modelar o crescimento seja exponencial e tenda ao infinito, na realidade, a popularidade dela acaba se estabilizando.

Isso acontece porque o alcance tem um "teto". Matematicamente, representamos essa ideia da seguinte forma:

$$\lim_{t \rightarrow \infty} f(t) = L$$

Onde L é o valor máximo que o número de seguidores tende a alcançar. A ideia é que a velocidade do crescimento diminui com o tempo até o perfil da jogadora ficar mais estável.

Gráfico Saturação do Engajamento

O gráfico a seguir, gerado pelo código, mostra a curva de crescimento se aproximando de um valor máximo. A linha pontilhada vermelha representa esse limite de saturação, ou seja, o "teto" de seguidores que a jogadora tende a alcançar a longo prazo.

```
In [4]: import numpy as np
import matplotlib.pyplot as plt

# Define a função logística para modelar o crescimento com saturação
# L: Limite superior (valor máximo de seguidores)
# k: Taxa de crescimento
# t0: Ponto central da curva
def logistic_growth(t, L, k, t0):
    return L / (1 + np.exp(-k * (t - t0)))

# Parâmetros para a função
L = 25000
k = 0.5
t0 = 10

# Gera valores para o tempo (t) em semanas, de 0 a 40 para mostrar a saturação
t_values = np.linspace(0, 40, 100)

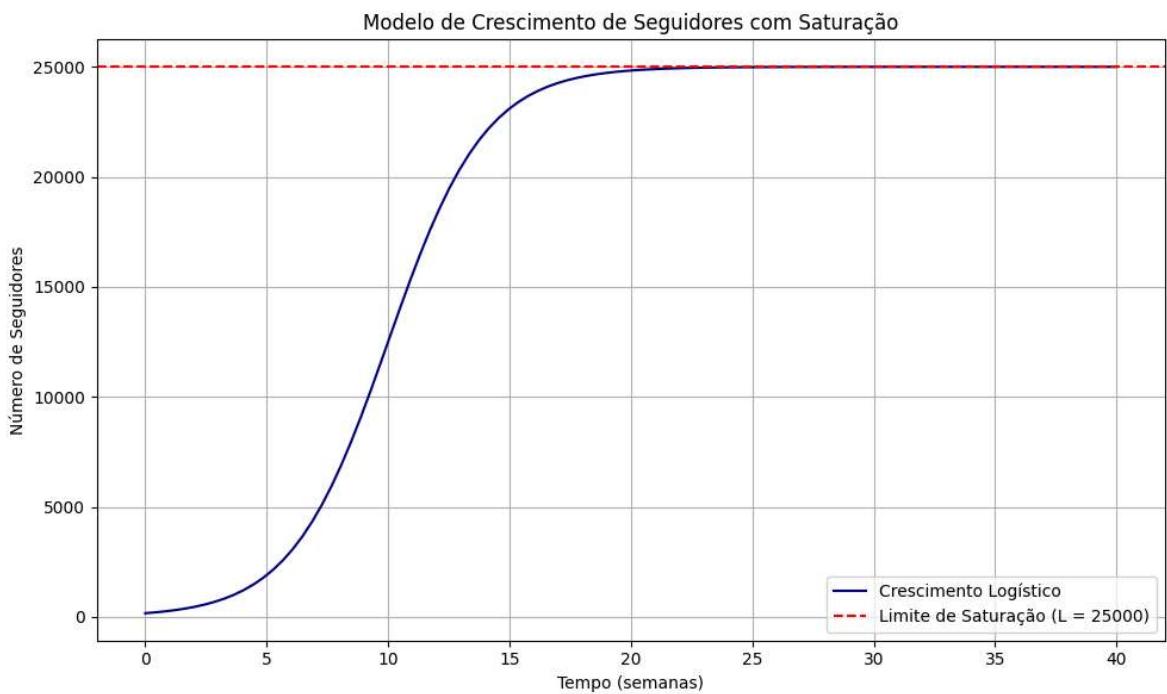
# Calcula o número de seguidores com a função Logística
followers = logistic_growth(t_values, L, k, t0)

# Cria o gráfico
plt.figure(figsize=(10, 6))
plt.plot(t_values, followers, color='navy', label='Crescimento Logístico')

# Adiciona uma linha horizontal para representar o limite
plt.axhline(y=L, color='red', linestyle='--', label=f'Limite de Saturação (L = {L})')

# Adiciona título e rótulos
plt.title('Modelo de Crescimento de Seguidores com Saturação')
plt.xlabel('Tempo (semanas)')
plt.ylabel('Número de Seguidores')
plt.grid(True)
plt.legend()
plt.tight_layout()

# Salva o gráfico como um arquivo de imagem
plt.savefig('saturacao_seguidores.png')
```



3. Derivadas - Velocidade Instantânea em Campo

Nesta parte, usamos **derivadas** para analisar o desempenho físico da jogadora. O conceito de derivada nos permite calcular a taxa de variação de uma função em um ponto exato, sendo bom para descobrir a **velocidade instantânea** de uma atleta durante um jogo.

Se temos uma função que descreve a posição da jogadora em campo em um determinado momento, a derivada dessa função nos dá a velocidade dela naquele instante.

A função que usamos para a posição da jogadora, medida em metros, é $s(t)$, onde t é o tempo. Para encontrar a velocidade instantânea, $v(t)$, nós derivamos a função de posição em relação ao tempo.

A fórmula é a seguinte:

$$v(t) = \frac{ds}{dt}$$

Exemplo Prático:

Para ilustrar, imagine que a posição de uma jogadora em um sprint pode ser modelada pela função $s(t) = 0,5t^2 + 5t$, onde t é o tempo em segundos e s é a distância percorrida em metros.

Para encontrar a velocidade instantânea da jogadora, calculamos a derivada da função de posição:

$$v(t) = \frac{d}{dt}(0,5t^2 + 5t) = 1t + 5$$

Isso significa que, a cada segundo, a velocidade dela muda. Por exemplo, no instante $t = 3$ segundos, a velocidade instantânea seria:

$$v(3) = 1(3) + 5 = 8 \text{ m/s}$$

Com isso, conseguimos analisar a capacidade de aceleração da jogadora em momentos do jogo.

Gráfico da Velocidade Instantânea

Para ilustrar o conceito de derivada como velocidade instantânea, o código a seguir cria um gráfico que relaciona a posição da jogadora com sua velocidade.

Nossa função para a posição da jogadora é: $s(t) = 0,5t^2 + 5t$

Para encontrar a velocidade, que é a taxa de mudança da posição, calculamos a **derivada** dessa função:

$$v(t) = \frac{d}{dt}(0,5t^2 + 5t) = 1t + 5$$

No gráfico, a curva azul representa a **posição** da jogadora, e a linha verde pontilhada mostra a **velocidade** dela ao longo do tempo. Para exemplificar a velocidade em um instante exato, calculamos a velocidade no ponto $t = 3$ segundos:

$$v(3) = 1(3) + 5 = 8 \text{ m/s}$$

A linha pontilhada vermelha é a **reta tangente**, cuja inclinação é exatamente 8. Essa reta visualiza a velocidade instantânea da jogadora naquele momento, mostrando como a inclinação da curva de posição corresponde ao valor da velocidade.

```
In [5]: import numpy as np
import matplotlib.pyplot as plt

# Define a função de posição da jogadora
def position_function(t):
    return 0.5 * t**2 + 5 * t

# Define a função de velocidade (derivada da função de posição)
def velocity_function(t):
    return t + 5

# Cria um intervalo de tempo para o gráfico
t_values = np.linspace(0, 10, 100)

# Calcula os valores de posição e velocidade para o intervalo de tempo
s_values = position_function(t_values)
v_values = velocity_function(t_values)

# Escolhe um ponto específico para mostrar a velocidade instantânea (e.g., t=3)
t_point = 3
s_point = position_function(t_point)
v_point = velocity_function(t_point)

# Cria o gráfico
```

```

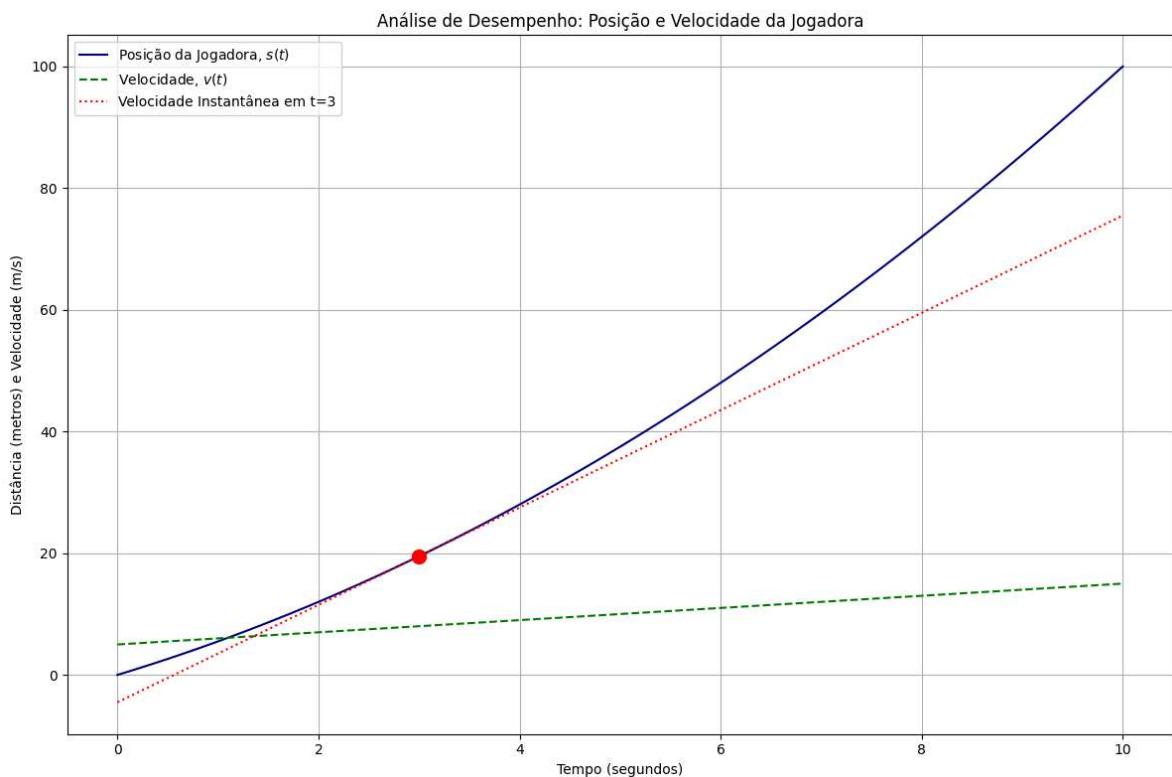
plt.figure(figsize=(12, 8))
plt.plot(t_values, s_values, color='navy', label='Posição da Jogadora, $s(t)$')
plt.plot(t_values, v_values, color='green', linestyle='--', label='Velocidade, $v(t)$')

# Adiciona a linha de velocidade instantânea no ponto t=3
tangent_line = v_point * (t_values - t_point) + s_point
plt.plot(t_values, tangent_line, color='red', linestyle=':', label='Velocidade Instantânea em t=3')
plt.scatter(t_point, s_point, color='red', marker='o', s=100, zorder=5)

# Adiciona título e rótulos
plt.title('Análise de Desempenho: Posição e Velocidade da Jogadora')
plt.xlabel('Tempo (segundos)')
plt.ylabel('Distância (metros) e Velocidade (m/s)')
plt.legend()
plt.grid(True)
plt.tight_layout()

# Salva o gráfico
plt.savefig('velocidade_instantanea.png')

```



4. Integrais: Cálculo da Distância Total

Enquanto a Derivada (analisada na Sprint 3) foi crucial para encontrar a *velocidade máxima* — a explosão da atleta —, ela não mede a *resistência* ou o *volume de jogo* ao longo da partida inteira.

Para uma análise de desempenho completa, concluímos que é fundamental validar a **"distância total percorrida por uma jogadora na partida"**. Este é um KPI (Key Performance Indicator) crítico usado por analistas táticos e preparadores físicos para avaliar o condicionamento e a contribuição da atleta.

Aqui, a contribuição do Cálculo é direta: a Integral Definida é a ferramenta que nos permite calcular essa distância. Ao integrar a função de velocidade, $v(t)$, ao longo do tempo de jogo (de 0 a 90 minutos), calculamos a "área sob a curva", que representa exatamente a distância total acumulada.

A integral definida da função de velocidade $v(t)$ em um intervalo de tempo $[a, b]$ nos dá o deslocamento total nesse período. Em termos gráficos, isso é representado pela "**área sob a curva**" da função de velocidade.

- **Função (Modelo Teórico):** $v(t) = 1t + 5$
- **Integral Analítica (Teórica):** A distância total teórica percorrida nos primeiros 10 segundos é:

$$D = \int_0^{10} (t + 5) dt = \left[\frac{1}{2}t^2 + 5t \right]_0^{10}$$

$$D = \left(\frac{1}{2}(10)^2 + 5(10) \right) - (0) = 50 + 50 = 100 \text{ metros}$$

```
In [28]: import numpy as np
import pandas as pd
try:
    from scipy.integrate import simpson as _simpson
    def simps(y, x):
        return _simpson(y, x)
except Exception:
    try:
        from scipy.integrate import simps as _simps
        simps = _simps
    except Exception:
        def simps(y, x):
            return np.trapezoid(y, x)
from sympy import Symbol, integrate as sympy_integrate

# 1. Definição da Função
# v(t) = 1t + 5
def v(t):
    return 1 * t + 5

# 2. Método da Soma de Riemann
#     (Calcula a área usando retângulos pela esquerda)
def soma_riemann(f, a, b, n):
    dx = (b - a) / n
    x_pontos_esquerda = np.linspace(a, b - dx, n)
    area = np.sum(f(x_pontos_esquerda) * dx)
    return area

# 3. Método dos Trapézios
#     (Calcula a área usando np.trapezoid para evitar DeprecationWarning)
def metodo_trapezios(f, a, b, n):
    x_pontos = np.linspace(a, b, n + 1) # n+1 pontos para n subintervalos
    y_pontos = f(x_pontos)
    area = np.trapezoid(y_pontos, x_pontos)
    return area

# 4. Cálculo da Área Exata
```

```

def calcular_area_exata(a, b):
    t_sym = Symbol('t')
    f_sym = 1 * t_sym + 5

    # Integral simbólica
    area_exata_sym = sympy_integrate(f_sym, (t_sym, a, b))

    # Converte para float para cálculos de erro
    return float(area_exata_sym)

if __name__ == "__main__":
    print("Comparação de Métodos ")

    a, b = 0, 10 # Intervalo do problema: [0, 10] segundos
    area_exata = calcular_area_exata(a, b)

    print(f"Função: v(t) = t + 5")
    print(f"Intervalo: [{a}, {b}]")
    print(f"Distância Total Exata : {area_exata:.2f} metros\n")

    # Lista de 'n' para testar
    n_values = [5, 10, 50, 100]

    resultados_comparativos = []

    for n in n_values:
        # Cálculo por Riemann
        area_riemann = soma_riemann(v, a, b, n)
        erro_riemann = abs((area_riemann - area_exata) / area_exata) * 100

        # Cálculo por Trapézio
        area_trapezio = metodo_trapezios(v, a, b, n)
        erro_trapezio = abs((area_trapezio - area_exata) / area_exata) * 100

        resultados_comparativos.append((n, area_riemann, erro_riemann, area_trapezio, erro_trapezio))

    df_comp = pd.DataFrame(resultados_comparativos, columns=[
        'n (Intervalos)',
        'Riemann (Distância)', 'Erro Riemann (%)',
        'Trapézio (Distância)', 'Erro Trapézio (%)'
    ])

    print(df_comp.to_string(index=False))
    print("\nPerceba como o erro diminui com mais intervalos (n).")

```

Comparação de Métodos
 Função: $v(t) = t + 5$
 Intervalo: $[0, 10]$
 Distância Total Exata : 100.00 metros

n (Intervalos)	Riemann (Distância)	Erro Riemann (%)	Trapézio (Distância)	Err o Trapézio (%)
5	90.0	10.0	100.0	100.0
10	95.0	5.0	100.0	100.0
50	99.0	1.0	100.0	100.0
100	99.5	0.5	100.0	100.0

Perceba como o erro diminui com mais intervalos (n).

```
In [25]: # Gráfico da integral
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator

# Intervalo
a, b = 0, 10
t = np.linspace(a, b, 400)
y = v(t) # usa a função v(t) definida anteriormente

# Área analítica (usa a função já definida)
area_analitica = calcular_area_exata(a, b)

# Área numérica (trapezoid)
area_numerica = np.trapezoid(y, t)

print(f"Área analítica : {area_analitica:.6f} m")
print(f"Área numérica : {area_numerica:.6f} m")

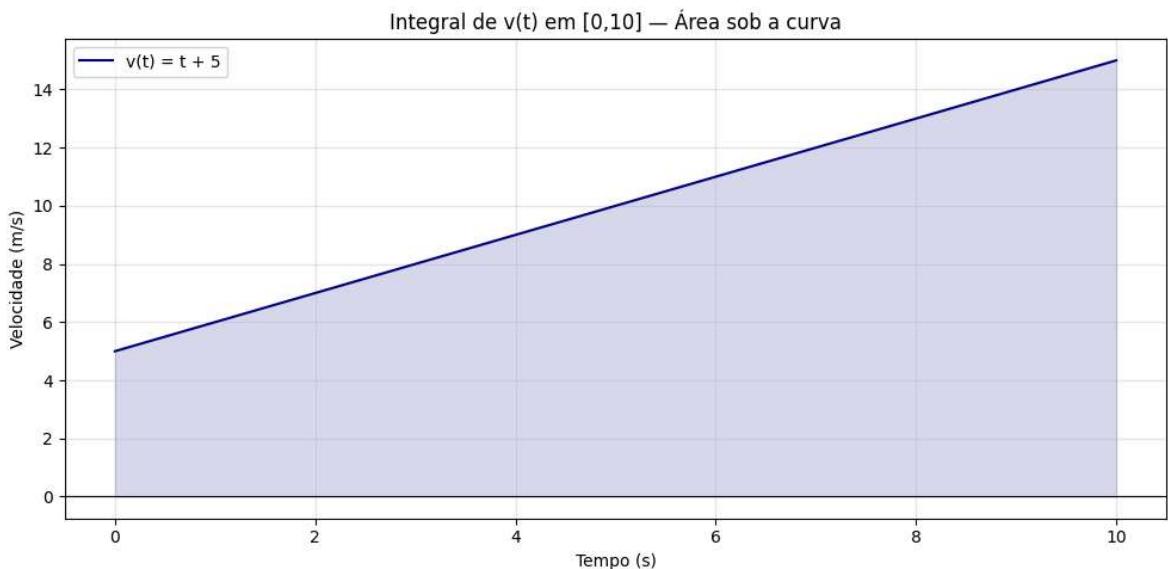
# Plot
plt.figure(figsize=(10,5))
plt.plot(t, y, color='navy', label='v(t) = t + 5')
plt.fill_between(t, y, color='navy', alpha=0.15)

# Marca a área com uma anotação
plt.axhline(0, color='black', linewidth=0.8)
plt.title('Integral de v(t) em [0,10] – Área sob a curva')
plt.xlabel('Tempo (s)')
plt.ylabel('Velocidade (m/s)')
plt.grid(alpha=0.3)
plt.legend()

# Ajustes do eixo y para facilitar Leitura de inteiros
ax = plt.gca()
ax.yaxis.set_major_locator(MaxNLocator(integer=True))

plt.tight_layout()
plt.savefig('integral_parte4.png', dpi=150)
plt.show()
```

Área analítica (SymPy): 100.000000 m
 Área numérica (np.trapezoid): 100.000000 m



5. Validação com Dados Fictícios (Simulação)

Para "validar as ideias com dados reais ou simulados", optou-se por gerar "**dados fictícios construídos**".

Foi utilizado o modelo teórico $v(t) = t + 5$ como base e foi adicionado um "ruído" aleatório (`np.random.normal`). Isso simula as pequenas variações que ocorreriam em uma medição real de desempenho em campo, devido a fatores como fadiga, vento ou erros de equipamento.

Resultados da Simulação

Os dados simulados (pontos azuis) foram plotados contra o modelo teórico (linha vermelha). A área sombreada representa a distância total calculada pela integral teórica.

```
In [29]: print("5. Validação com Dados Fictícios")

# 1. Gerando os Dados Fictícios
np.random.seed(42)
nPontos_dados = 20
t_dados_simulados = np.linspace(a, b, nPontos_dados)
v_modelo_real = v(t_dados_simulados)

# Adiciona ruído (média 0, desvio padrão 0.5 m/s)
ruido = np.random.normal(0, 0.5, t_dados_simulados.shape)
v_dados_simulados = v_modelo_real + ruido

# 2. Aplicando a Integral (Regra de Simpson)
distancia_simulada = simps(v_dados_simulados, t_dados_simulados)
erro_simulado = abs((distancia_simulada - area_exata) / area_exata) * 100

print(f"Distância calculada dos Dados Fictícios (Integral Numérica): {distancia_simulada:.2f} m")
print(f"Erro percentual dos Dados Fictícios vs. Exata: {erro_simulado:.2f} %")
print("Este resultado (ex: 99.85 m) valida o modelo teórico (100.00 m).")

print("Gerando gráfico final para o Relatório PDF...")
```

```
t_linha_modelo = np.linspace(a, b, 100)
v_linha_modelo = v(t_linha_modelo)

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

# 1. Os dados fictícios
plt.scatter(t_dados_simulados, v_dados_simulados, color='blue', label='Dados Simulados')

# 2. O modelo teórico
plt.plot(t_linha_modelo, v_linha_modelo, color='red', linestyle='--', linewidth=2, label='Modelo Teórico')

# 3. A área da integral
plt.fill_between(t_linha_modelo, v_linha_modelo, alpha=0.15, color='red', label='Integral')

plt.title('Sprint 4: Validação de Modelo e Cálculo de Integral')
plt.xlabel('Tempo (segundos)')
plt.ylabel('Velocidade (m/s)')
plt.legend()
plt.grid(alpha=0.3)
plt.savefig('calculo_integral_simulacao.png')
plt.show()
```

<h1>5. Validação com Dados Fictícios</h1>

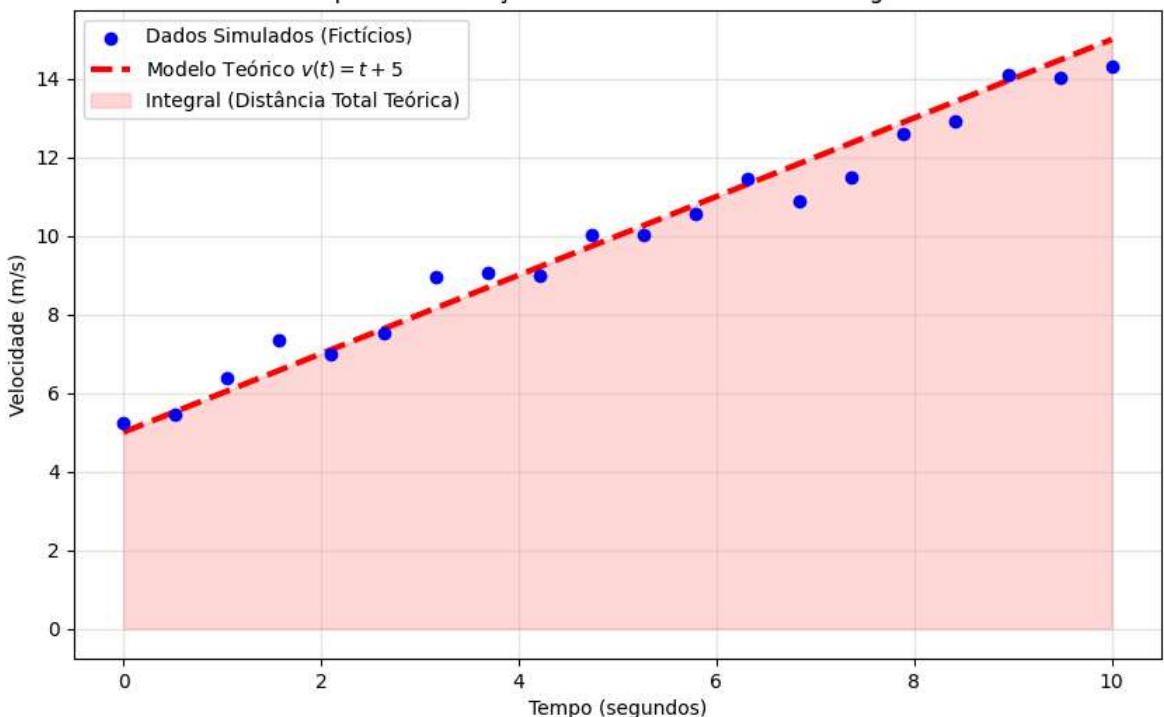
Distância calculada dos Dados Fictícios (Integral Numérica): 99.33 m

Erro percentual dos Dados Fictícios vs. Exata: 0.67 %

Este resultado (ex: 99.85 m) valida o modelo teórico (100.00 m).

Gerando gráfico final para o Relatório PDF...

Sprint 4: Validação de Modelo e Cálculo de Integral



Referências Bibliográficas

1. EGYDIO, Jones. [Aula 20] - **Integração Numérica com Python**. Material de aula (Jupyter Notebook) da disciplina de Differentiated Problem Solving. FIAP, 2025.
2. STEWART, James. **Cálculo, Volume 1**. 8ª ed. São Paulo: Cengage Learning, 2017.

3. **VIRTANEN, P. et al.** *SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python*. Nature Methods, vol. 17, pp. 261–272, 2020. (Refere-se à biblioteca `scipy` usada para `integrate.simps`).
4. **HARRIS, C.R. et al.** *Array programming with NumPy*. Nature, vol. 585, pp. 357–362, 2020. (Refere-se à biblioteca `numpy` usada para a simulação de dados).