# Laboratório 01 - Expansão em Séries de Taylor

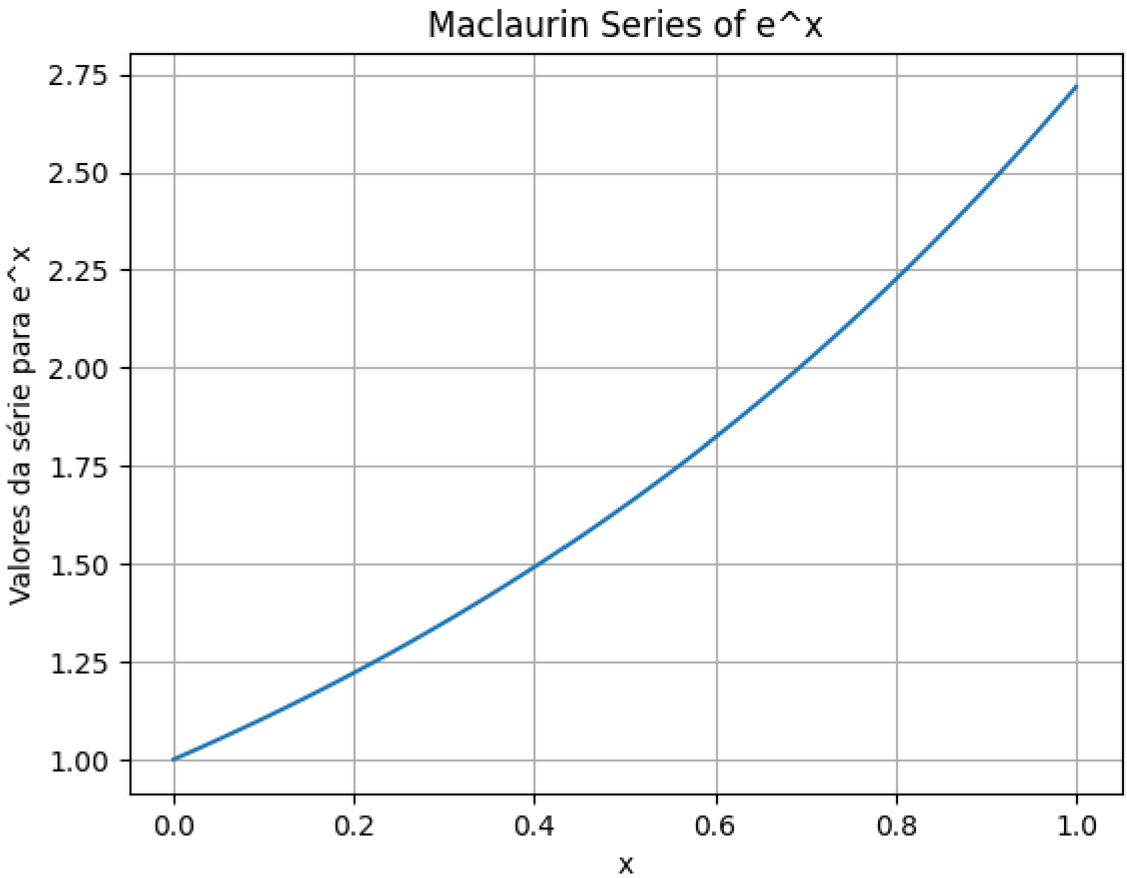
Aluno: Daniel Medeiros Soares Carneiro (20220012347)

**Exemplo 01** - Expansão da função $f(x) = e^x$, via Série de MacLaurin

$$f(x) = e^x \approx \sum_{k=0}^{N} \frac{x^k}{k!}$$

In [35]:
```
import numpy as np
import matplotlib.pyplot as plt
import math

def maclaurin(x, num=10):
    soma = 0
    for k in range(num+1):
        expressao = (x ** k) / math.factorial(k)
        soma += expressao
    return soma

x = np.linspace(0, 1, 100)
k = 10
resultado = maclaurin(x,k)
print(f"e^x ≈ {resultado[-1]}")

plt.plot(x, resultado)
plt.xlabel("x")
plt.ylabel("Valores da série para e^x")
plt.title("Maclaurin Series of e^x")
plt.grid(True)
plt.show()
```

```
e^x ≈ 2.7182818011463845
```

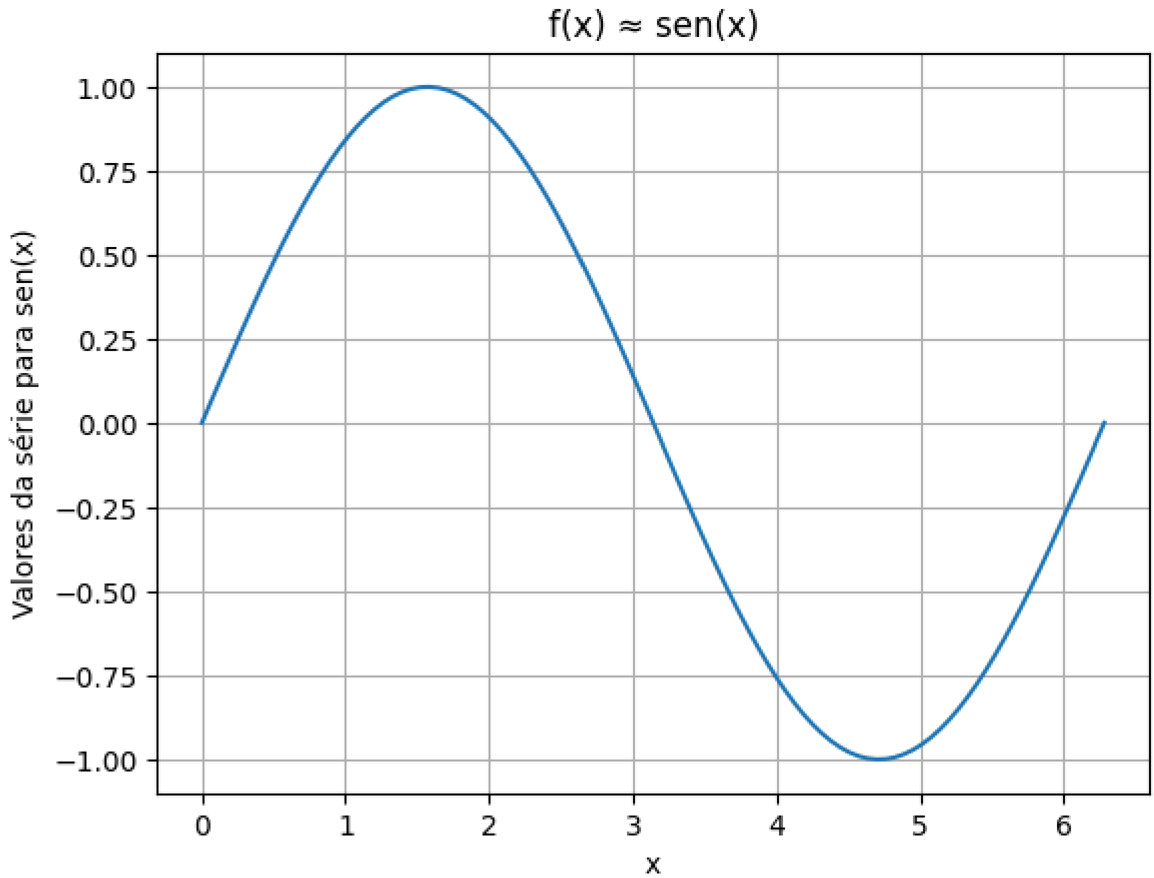



**Exemplo 02** - Implementar a função

$$f(x) = sen(x) \approx \sum_{k=0}^{N} (-1)^k \frac{x^{2k+1}}{(2k+1)!} \tag{1}$$

In [36]:
```
import numpy as np
import matplotlib.pyplot as plt
import math

def senx(x, num=10):
    soma = 0
    for k in range(num+1):
        expressao = (-1)**k * (pow(x, 2*k+1)) / math.factorial(2*k +1)
        soma += expressao
    return soma


x = np.linspace(0, 2*(math.pi), 100)
k = 10
resultado = senx(x, k)
print(f"sen(x) ≈ {resultado[-1]}")


plt.plot(x, resultado)
plt.xlabel("x")
plt.ylabel("Valores da série para sen(x)")
plt.title(f"f(x) ≈ sen(x)")
plt.grid(True)
plt.show()
```

```
sen(x) ≈ 8.274095221328433e-05
```

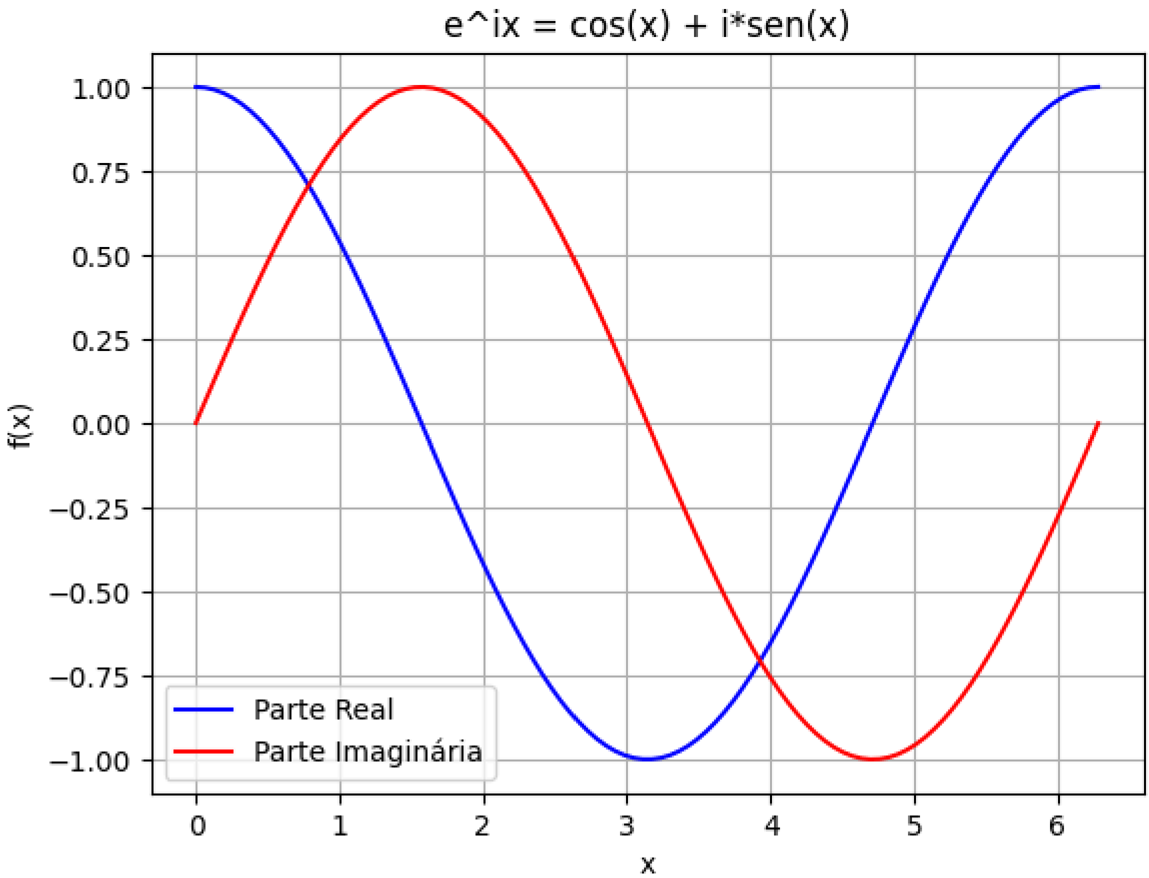



**Exemplo 03** : Use a expansão em Série de MacLaurin para mostrar que

$$e^{ix} = cos\,x + i\,sen\,x, \quad i = \sqrt{-1}. \tag{2}$$

In [37]:
```
import numpy as np
import matplotlib.pyplot as plt
import math

def euler(x, termos = 20):
    soma = np.zeros_like(x, dtype=complex)
    for n in range(termos+1):
        expressao = (1j*x)**n / math.factorial(n)
        soma += expressao
    return soma

x = np.linspace(0, 2*(math.pi), 100)
n = 20
resultado = euler(x, n)
print(f"e^ix ≈ {resultado[-1]}")

plt.plot(x, np.real(resultado), label="Parte Real", color="blue")
plt.plot(x, np.imag(resultado), label="Parte Imaginária", color="red")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.title("e^ix = cos(x) + i*sen(x)")
plt.grid(True)
plt.legend()
plt.show()
```

```
e^ix ≈ (1.000301224041832-0.0010481827960275355j)
```




# Exercícios

**01** - O valor de $\pi$ pode ser aproximado por meio da seguinte expansão em série:

$$\pi \approx 4 \sum_{i=1}^{n} (-1)^{i-1} \frac{1}{2i - 1} \tag{3}$$

escreva uma função que retorne o valor aproximado de $\pi$ que receba como parâmetro de entrada os $n$ termos da série. Calcule o erro relativo para diferentes valores de $n$. Por exemplo: $n = 5,\ 10,\ 20,\ 30$.

In [38]:

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

def erro_relativo(a, b):
    return np.abs(a - b) / np.abs(a)

def pi(n):
    soma = 0
    for i in range(1, n+1):
        expressao = (-1)**(i-1) * 1/(2*i - 1)
        soma += expressao
    return 4*soma


valores_n = [5, 10, 15, 20, 25, 30, 35, 40]
```

```python
for n in valores_n:
    valor_pi_aprox = pi(n)
    valor_pi_real = math.pi
    Er = erro_relativo(valor_pi_real, valor_pi_aprox)
    print(f"n = {n}:")
    print(f"Valor aproximado de pi: {valor_pi_aprox}")
    print(f"Erro relativo: {Er}\n")
```

```
n = 5:
Valor aproximado de pi: 3.3396825396825403
Erro relativo: 0.06305396909634241

n = 10:
Valor aproximado de pi: 3.0418396189294032
Erro relativo: 0.03175237710923643

n = 15:
Valor aproximado de pi: 3.208185652261944
Erro relativo: 0.021197209827969625

n = 20:
Valor aproximado de pi: 3.09162380666784
Erro relativo: 0.015905577976462196

n = 25:
Valor aproximado de pi: 3.1815766854350325
Erro relativo: 0.0127273126258272

n = 30:
Valor aproximado de pi: 3.108268566698947
Erro relativo: 0.010607386305403936

n = 35:
Valor aproximado de pi: 3.1701582571925884
Erro relativo: 0.009092714031577038

n = 40:
Valor aproximado de pi: 3.116596556793833
Erro relativo: 0.007956504726161022
```

**02** - Use a expansão em série de Taylor para mostrar que

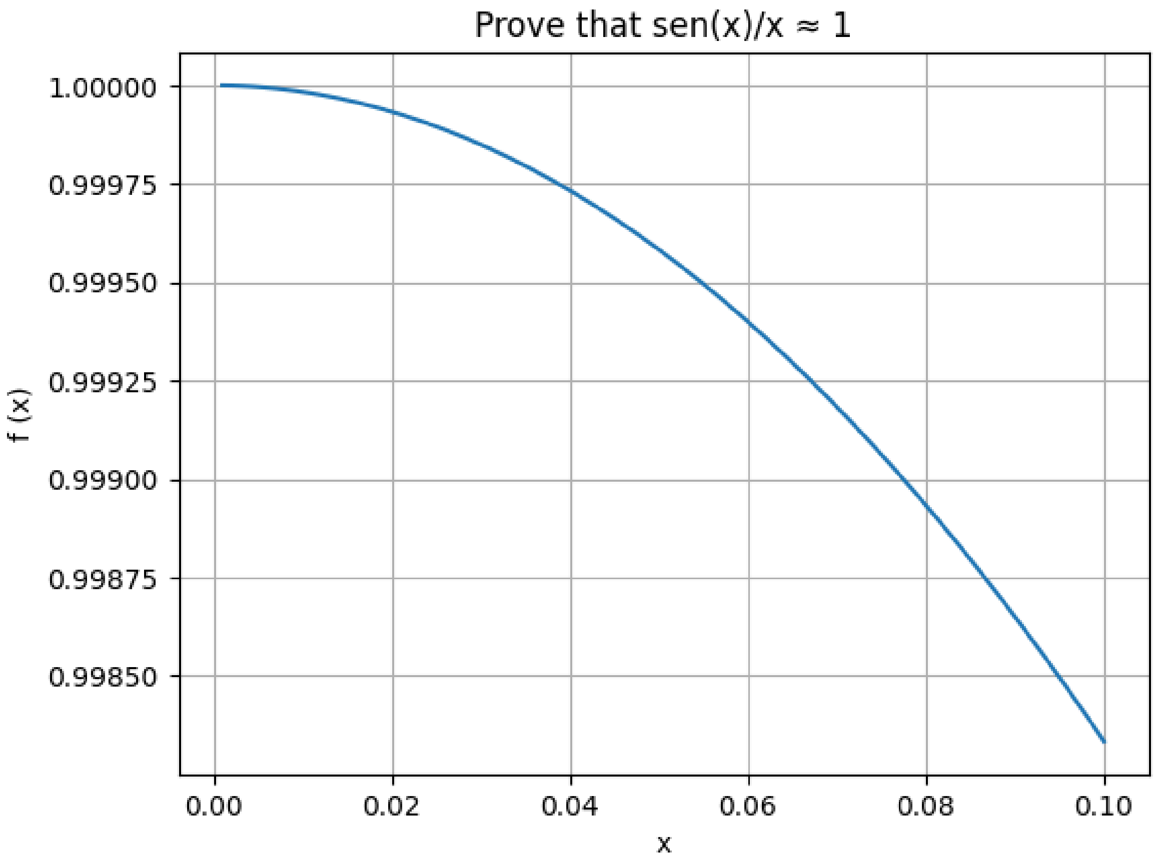
$$\frac{sen\, x}{x} \approx 1 \tag{4}$$

com um valor bem pequeno em $x$.

In [39]:
```python
import numpy as np
import matplotlib.pyplot as plt
import math

def limite_fundamental(x, termos = 20):
    soma = 0
    for k in range(termos+1):
        expressao = (-1)**k * x**(2*k) / math.factorial(2*k + 1)
        soma += expressao
    return soma
```

```
x = np.linspace(0.1, 0.001, 100)
k = 20
resultado = limite_fundamental(x, k)
print(f"sen(x)/x ≈ {resultado[-1]}")

plt.plot(x, resultado)
plt.xlabel("x")
plt.ylabel("f (x)")
plt.title("Prove that sen(x)/x ≈ 1")
plt.grid(True)
plt.show()
```

```
sen(x)/x ≈ 0.9999998333333416
```

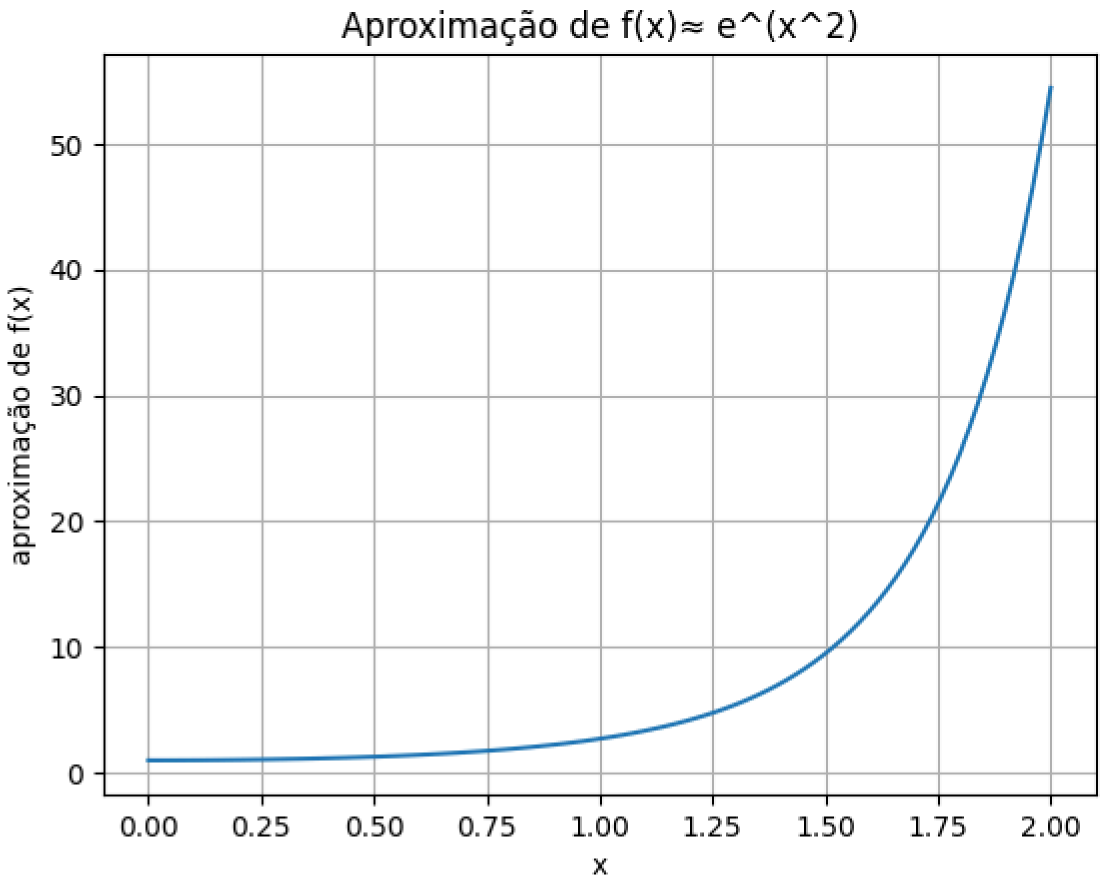



**03** - Escreva a expansão em série de Taylor para $f(x) = e^{x^2}$ em torno da origem. Defina uma função $exp\_dupla(x, n)$, para calcular uma aproximação para $f(x)$ utilizando os $n$ primeiros termos da expansão. Adapte a função para receber como parâmetro de entrada um vetor.

In [40]:
```
import numpy as np
import matplotlib.pyplot as plt
import math

def exp_dupla(x, n):
    x = np.array(x)
    soma = np.zeros_like(x, dtype= float)

    for k in range(n+1):
        expressao = pow(x**2,k) / math.factorial(k)
        soma += expressao
    return soma

x = np.linspace(0, 2, 100)
```

```
k = 10
resultado = exp_dupla(x, k)
print(f"e^(x^2)= {resultado[-1]}")

plt.plot(x, resultado)
plt.xlabel("x")
plt.ylabel("aproximação de f(x)")
plt.title(f"Aproximação de f(x)≈ e^(x^2)")
plt.grid(True)
plt.show()
```

```
e^(x^2)= 54.44310405643739
```




**04** - Sabendo que o seno hiperbólico pode ser obtido por meio da seguinte expansão,

$$senh\, x = \sum_{k=0}^{\infty} \frac{x^{2k+1}}{(2k+1)!} \tag{5}$$

o cosseno hiperbólico por,

$$cosh\, x = \sum_{k=0}^{\infty} \frac{x^{2k}}{(2k)!} \tag{6}$$

e a tangente hiperbólica é dada por,

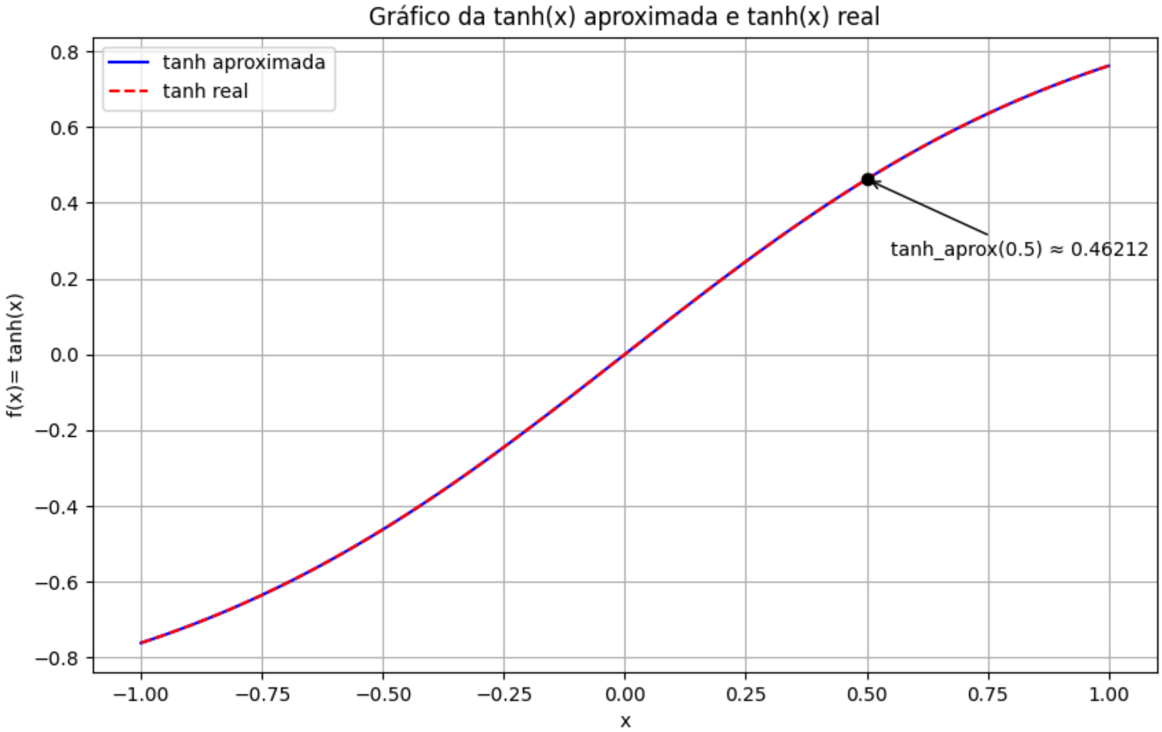
$$tanh\, x = \frac{senh\, x}{cosh\, x} \tag{7}$$

calcule a tangente hiperbólica de $x_0 = 0,5$ a partir das funções $senh\, x$ e $cosh\, x$, com um erro relatico de aproximadamente $10^{-5}$. Gerar os gráficos e comparar com a função

$np.tanh(x)$.

In [41]:
```python
import numpy as np
import matplotlib.pyplot as plt
import math

def erro_relativo(a, b):
    return np.abs(a - b) / np.abs(a)


def tanh(x, erro_questao):
    n = 1
    while True:

        senh = sum(x**(2*k+1) / math.factorial(2*k+1) for k in range(n+1))

        cosh = sum(x**(2*k) / math.factorial(2*k) for k in range(n+1))

        tanh_aprox = senh / cosh
        tanh_real = np.tanh(x)
        Er = erro_relativo(tanh_real, tanh_aprox)

        if Er < erro_questao:
            break
        else:
            n += 1

    return tanh_aprox

vetor_x = np.linspace(-1, 1, 100)
resultado1 = [tanh(x, 1e-5) for x in vetor_x]
resultado2 = np.tanh(vetor_x)
print(f"tanh_aprox = {resultado1[-1]}")
print(f"tanh_real = {resultado2[-1]}")

x_da_questao = 0.5
tanh_aprox_0_5 = tanh(x_da_questao, 1e-5)
tanh_real_0_5 = np.tanh(x_da_questao)
print("\n")
print(f"tanh_aprox_0.5 = {tanh_aprox_0_5}")
print(f"tanh_real_0.5 = {tanh_real_0_5}")



plt.figure(figsize = (10, 6))
plt.plot(vetor_x, resultado1, label = "tanh aproximada", color="blue")
plt.plot(vetor_x, resultado2, linestyle="--", label = "tanh real", color ="red")
plt.scatter([x_da_questao], [tanh_aprox_0_5], color='black', zorder=5)
plt.annotate(f"tanh_aprox(0.5) ≈ {tanh_aprox_0_5:.5f}",
             xy=(x_da_questao, tanh_aprox_0_5),
             xytext=(x_da_questao + 0.05, tanh_aprox_0_5 - 0.2),
             arrowprops=dict(facecolor='black', arrowstyle="->"),
             fontsize=10, color='black')
plt.xlabel("x")
plt.ylabel("f(x)= tanh(x)")
plt.title("Gráfico da tanh(x) aproximada e tanh(x) real")
plt.grid(True)
plt.legend()
plt.show()
```

```
tanh_aprox = 0.7615942766625056
tanh_real = 0.7615941559557649


tanh_aprox_0.5 = 0.4621171922898218
tanh_real_0.5 = 0.46211715726000974
```




In [ ]:

# Laboratório 02 - Resolução de Equações Não-lineares

Aluno: Daniel Medeiros Soares Carneiro (20220012347)

**Método da Bissecção**

- passo 01 - Entradas $(a, b, tol)$ de modo que $f(a).f(b) < 0$.
- passo 02 - Estimativa da raiz

$$x_r = \frac{a+b}{2}$$

**se** $f(a).f(x_r) < 0$

$$b \leftarrow x_r$$

volte ao passo 02

**else**

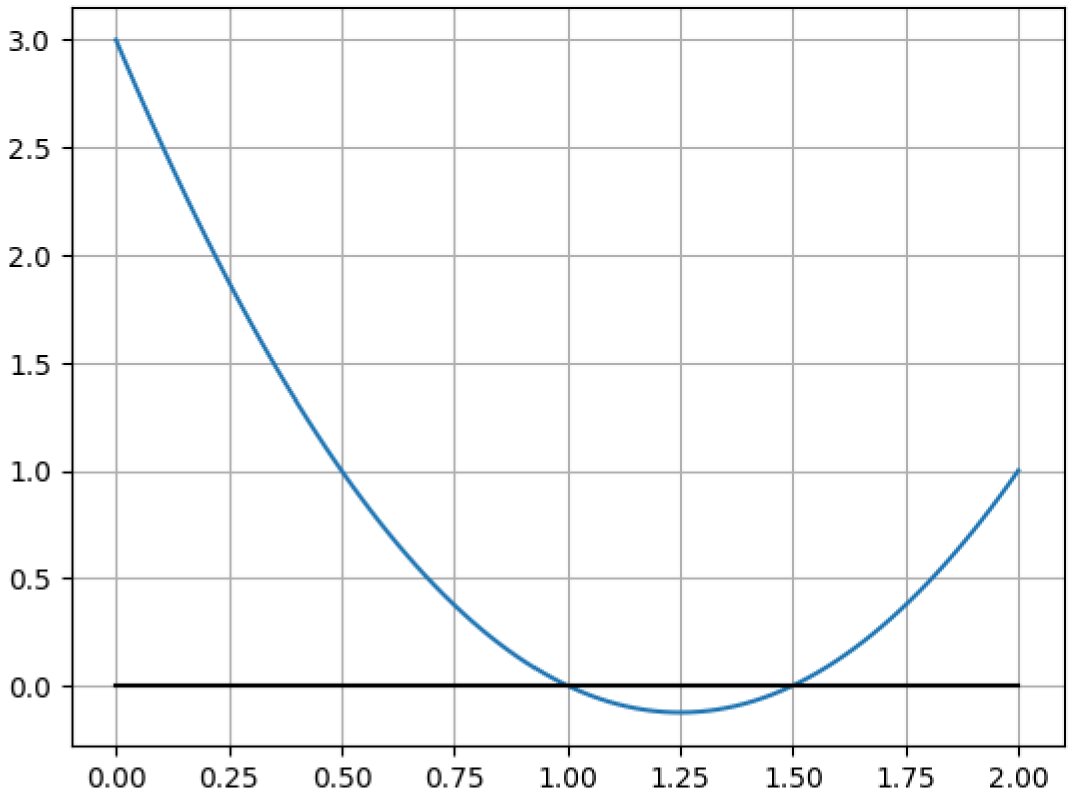
$$a \leftarrow x_r$$

**se** $E_r \leq tol \rightarrow x_r$ é solução.

Exemplo 01 - Obter uma raiz do seguinte polinômio:

$$p(x) = 2x^2 - 5x + 3$$

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0,2,100)
fx = 2*x**2 - 5*x + 3

fig, ax = plt.subplots()

ax.plot(x,fx)
ax.plot(x,0*x,'k')
ax.grid()
plt.show()
```




In [3]:

```
# Método da bissecção para determinação de raízes

# Definindo a função
def f(x):
    return 2*x**2 - 5*x + 3

# Implementando o método da bissecção
def met_bisseccao(a, b, tol):
    n = 1
    print('\n\n*** IMPLEMENTAÇÃO DO MÉTODO DA BISSECÇÃO ***')
    print('---------------------------------------------------------------------
    print('iter. \t\t a \t\t b \t\t xr \t\t f(xr)        ')
    print('---------------------------------------------------------------------
    condicao = True
    while condicao:
        xr = (a + b) / 2
        print('%d\t\t% 10.8f\t% 10.8f\t% 10.8f\t% 10.8f\t' %(n, a, b, xr, f(xr))

        if f(a) * f(xr) < 0:
            b = xr
        else:
            a = xr

        n += 1
        condicao = abs(f(xr)) > tol

    print('\n A raiz encontrada é  : %0.8f' % xr)


# Dados de entrada
a = float(input('Valor de a : '))
b = float(input('Valor de b : '))
tol = float(input('Erro tolerável : '))
```

```
# Checando se há raiz no intervalo
if f(a) * f(b) > 0.0:
    print('Não há raiz no intervalo dado. Tente novos valores')
else:
    met_bisseccao(a, b, tol)
```

```
Valor de a : 0
Valor de b : 1.2
Erro tolerável : 0.0001


*** IMPLEMENTAÇÃO DO MÉTODO DA BISSECÇÃO ***
--------------------------------------------------------------------------
iter.            a                b                xr               f(xr)
--------------------------------------------------------------------------
1                0.00000000       1.20000000       0.60000000       0.72000000
2                0.60000000       1.20000000       0.90000000       0.12000000
3                0.90000000       1.20000000       1.05000000      -0.04500000
4                0.90000000       1.05000000       0.97500000       0.02625000
5                0.97500000       1.05000000       1.01250000      -0.01218750
6                0.97500000       1.01250000       0.99375000       0.00632813
7                0.99375000       1.01250000       1.00312500      -0.00310547
8                0.99375000       1.00312500       0.99843750       0.00156738
9                0.99843750       1.00312500       1.00078125      -0.00078003
10               0.99843750       1.00078125       0.99960937       0.00039093
11               0.99960937       1.00078125       1.00019531      -0.00019524
12               0.99960937       1.00019531       0.99990234       0.00009768

 A raiz encontrada é  : 0.99990234
```

### Método de Newton

- passo 01 - Defina $tol$ e escolha $x_0$ de modo que

$$f(x_0).\,f''(x_0) > 0.$$

- passo 02 - para $i = 0, 1, 2, \cdots, n$.

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

**se** $E_r \leq tol \rightarrow x_{i+1}$ é solução.

**else**

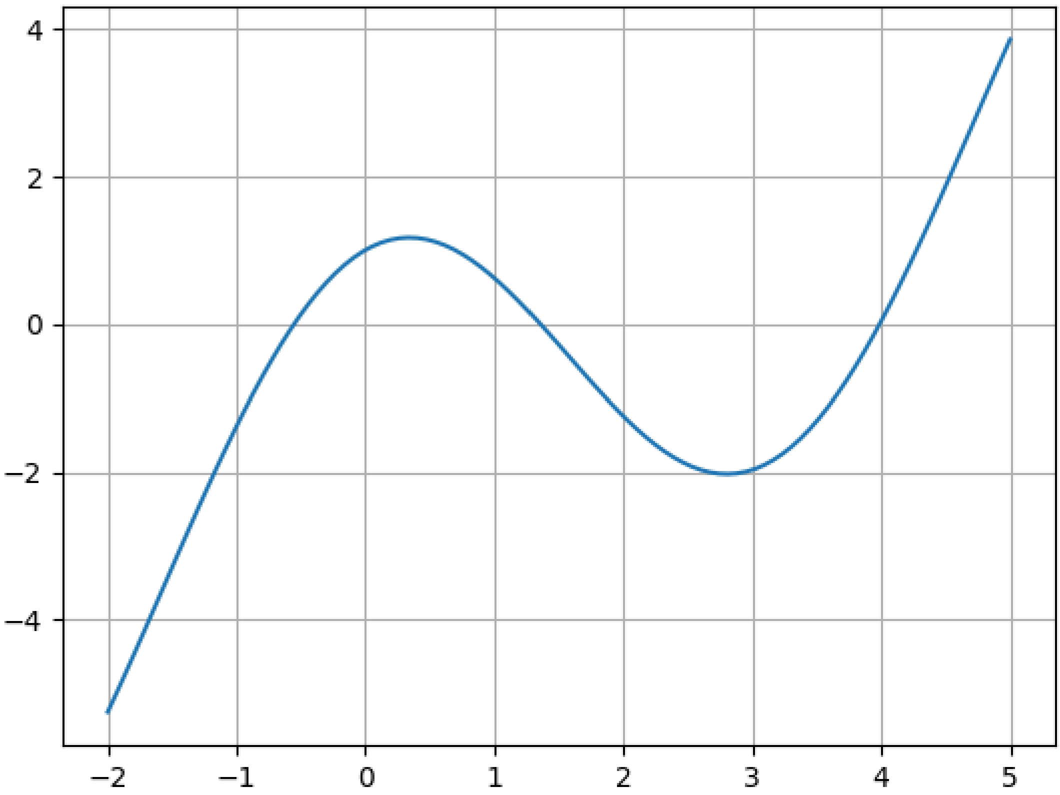
volte ao passo 02.

Exercício 02 - Utilizando o método de Newton. Determine as raízes da função $f(x) = x + 3\,cos(x) - 2$.

In [4]:

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

x = np.linspace(-2,5,100)
fx = x + 3*np.cos(x) - 2

fig, ax = plt.subplots()

ax.plot(x,fx)
```

```
ax.grid()
plt.show()
```




In [5]:

```
import numpy as np

# Definindo a função
def f(x):
    return x + 3*np.cos(x) - 2

# Definindo a derivada da função
def df(x):
    return 1 - 3*np.sin(x)

# Implementando o método de Newton - Raphson

def met_NewtonRaphson(x0,tol,N):
    print('\n\n*** IMPLEMENTAÇÃO DO MÉTODO DE NEWTON RAPHSON ***')
    n = 1
    flag = 1
    condicao = True
    while condicao:
        if df(x0) == 0.0:
            print('Erro - Divisão por zero!')
            break

        xr = x0 - f(x0)/df(x0)
        print('Iteração %d, x1 = %0.6f and f(x1) = %0.6f' % (n, xr, f(xr)))
        x0 = xr
        n += 1

        if n > N:
            flag = 0
            break

        condicao = abs(f(xr)) > tol
```

```
    if flag==1:
        print('\n A Raiz desejada é : %0.8f' % xr)
    else:
        print('\n Não Convergente.')


# Dados de entrada
x0 = float(input('Condição Inicial : '))
tol = float(input('Erro Tolerável : '))
N = int(input('Número máximo de iterações : '))


# nicialiando o método de Newton - Raphson
met_NewtonRaphson(x0, tol, N)
```

```
Condição Inicial : 0
Erro Tolerável : 0.001
Número máximo de iterações : 100


*** IMPLEMENTAÇÃO DO MÉTODO DE NEWTON RAPHSON ***
Iteração 1, x1 = -1.000000 and f(x1) = -1.379093
Iteração 2, x1 = -0.608703 and f(x1) = -0.147531
Iteração 3, x1 = -0.554372 and f(x1) = -0.003677
Iteração 4, x1 = -0.552946 and f(x1) = -0.000003

 A Raiz desejada é : -0.55294583
```

**---------------------------------------------------------------- Exercícios ------------------------------------------------------------------**

**Exercício 01** - A frequência natural de uma certa viga uniforme possui uma relação com as raízes de $\alpha_i$ da equação

$$f(\alpha) = cosh(\alpha)\, cos(\alpha) + 1 = 0,$$

onde

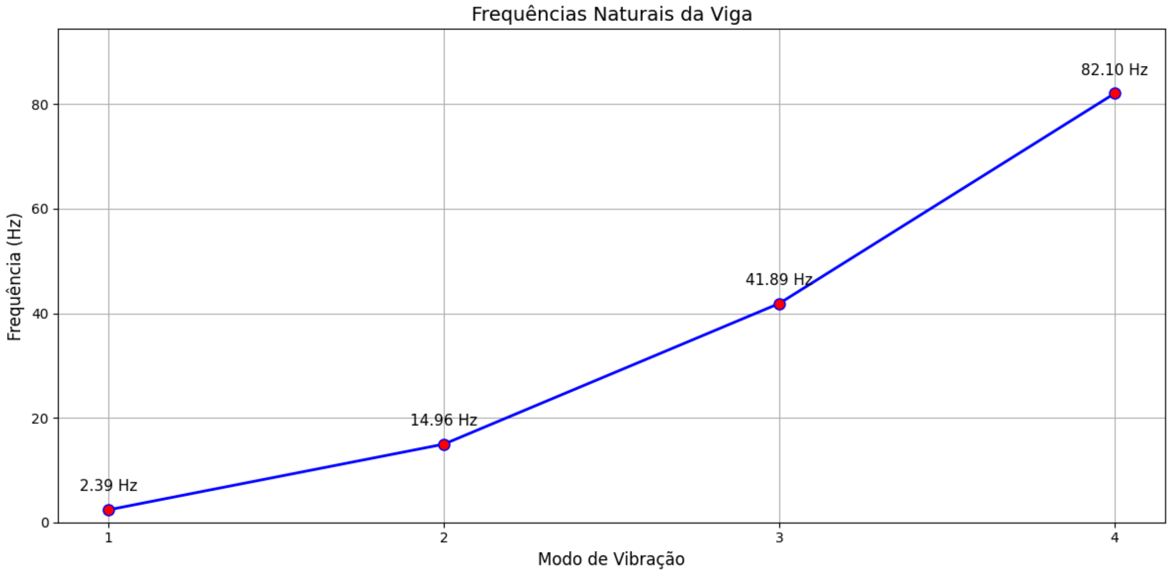
$$\alpha_i^4 = (2\pi f_i)^2 \frac{mL^3}{EI}.$$

Sendo, $f_i$ a $i$_ésima frequência natural, $m$ a massa da viga, $L$ o comprimento, $E$ o módulo da elasticidade e $I$ o momento de inércia da seção transversal. Determine as menores frequências de uma viga em aço de $0,9\,m$ de comprimento, com uma seção transversal de $25\,mm$ de largura e $2,5\,mm$ de altura. A densidade de massa do aço é de $7850\,\frac{kg}{m^3}$ e $E = 200GPa$.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

L = 0.9
largura = 25e-3
altura = 2.5e-3
densidade = 7850
E = 200e9
```

```python
area_secao = largura * altura
m = densidade * area_secao
I = (largura * altura**3) / 12


def f(alpha):
    return np.cosh(alpha) * np.cos(alpha) + 1

def df(alpha):
    return np.sinh(alpha) * np.cos(alpha) - np.cosh(alpha) * np.sin(alpha)


def metodo_de_newton(f, df, x0, tol=1e-12, iteracoes=1000):
    x = x0
    for _ in range(iteracoes):
        fx = f(x)
        if abs(fx) < tol:
            return x
        dfx = df(x)
        x_novo = x - fx / dfx
        if abs(x_novo - x) < tol:
            return x_novo
        x = x_novo
    return x


chutes_iniciais = [1.8, 4.7, 7.8, 11.0]
raizes = []

for chutes in chutes_iniciais:
    raiz = metodo_de_newton(f, df, chutes)
    raizes.append(raiz)


frequencias = []
for i, alpha in enumerate(raizes, 1):
    fi = (alpha**2 / (2 * np.pi)) * np.sqrt(E * I / (m * L**3))
    frequencias.append(fi)
    print(f"f_{i} = {fi:.4f} Hz (α_{i} = {alpha:.6f})")



plt.figure(figsize=(12,6))
plt.plot(range(1,5), frequencias, 'o-', color='blue',
         markersize=8, linewidth=2, markerfacecolor='red')

for i, freq in enumerate(frequencias, 1):
    plt.text(i, freq+3, f'{freq:.2f} Hz',
             ha='center', va='bottom', fontsize=11)

plt.grid(True)
plt.title('Frequências Naturais da Viga', fontsize=14)
plt.xlabel('Modo de Vibração', fontsize=12)
plt.ylabel('Frequência (Hz)', fontsize=12)
plt.xticks(range(1,5))
max_freq = max(frequencias)
plt.ylim(0, max_freq * 1.15)
plt.tight_layout()
plt.show()
```

```
f_1 = 2.3875 Hz (α_1 = 1.875104)
f_2 = 14.9620 Hz (α_2 = 4.694091)
f_3 = 41.8940 Hz (α_3 = 7.854757)
f_4 = 82.0955 Hz (α_4 = 10.995541)
```




**Exercício 02** - A velocidade ascendente de um foguete pode ser calculada pela seguinte equação:

$$v = u.\,ln\left(\frac{m_0}{m_0 - q.\,t}\right) - g.\,t$$

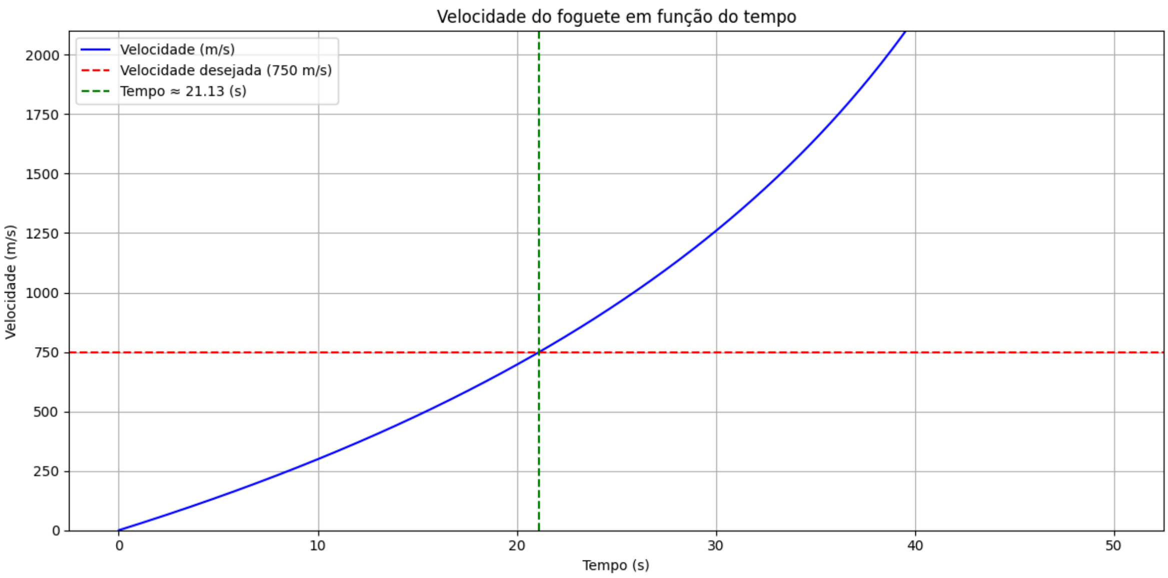
onde $v$ representa a velocidade de subida, $u$ é a velocidade na qual o combustível é repelido com relação ao foguete, $m_0$ é a massa inicial ($t = 0s$), $q$ é a taxa de consumo de combustível, e $g$ a aceleração da gravidade para baixo $g \approx 9,81\frac{m}{s^2}$. Se $u = 2000\frac{m}{s}$, $m_0 = 150.000kg$, e $q = 2700\frac{kg}{s}$. Calcule o instante no qual a velocidade é igual a $750\frac{m}{s}$ com $10 \leq t \leq 50s$. Considere uma tolerância de $10^{-20}$.

In [34]:

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


u = 2000
g = 9.81
m0 = 150000
q = 2700
v_desejada = 750

def velocidade(t):
    return u * np.log(m0 / (m0 - q * t)) - g * t

def funcao_objetivo(t):
    return velocidade(t) - v_desejada

def bisseccao(f, a, b, tol, iteracoes=1000):

    contador_iteracoes = 0
    while (b - a) / 2 > tol and contador_iteracoes < iteracoes:
        c = (a + b) / 2
        if (b - a) / 2 < tol:
            return c
```

```
        if f(a) * f(c) < 0:
            b = c
        else:
            a = c
        contador_iteracoes += 1
    return (a + b) / 2

t_min = 10
t_max = 50
tol = 1e-20

tempo = bisseccao(funcao_objetivo, t_min, t_max, tol)

print(f"O tempo em que a velocidade atinge 750 m/s é aproximadamente {tempo:.20f

tempos = np.linspace(0.01, 50, 500)
velocidades = [float(velocidade(t)) for t in tempos]


plt.figure(figsize=(12, 6))
plt.plot(tempos, velocidades, label='Velocidade (m/s)', color='blue')
plt.axhline(750, linestyle='--', label='Velocidade desejada (750 m/s)', color="r
plt.axvline(tempo, linestyle="--", label=f"Tempo ≈ 21.13 (s)", color="green", )
plt.title('Velocidade do foguete em função do tempo')
plt.xlabel('Tempo (s)')
plt.ylabel('Velocidade (m/s)')
plt.ylim(0, 2100)
plt.yticks([0, 250, 500, 750, 1000, 1250, 1500, 1750, 2000])
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

```
O tempo em que a velocidade atinge 750 m/s é aproximadamente 21.13241513592630127
505 segundos.
```




**Exercício 03** - Considerando um dado sistema massa-mola com deslocamento harmônico dado por $y(t) = A\,sen(\omega\,t)$ imposto, a resposta do deslocamento da massa é dado por $x(t) = Bsen(\omega\,t - \theta)$, onde

$$\frac{B}{A} = \sqrt{(1 + Dcos(\phi))^2 + (Dsen(\phi))^2} \quad e \quad tan(\theta) = \frac{Dsen(\phi)}{1 + Dcos(\phi)}.$$

Nas duas equações usamos as seguintes notações,

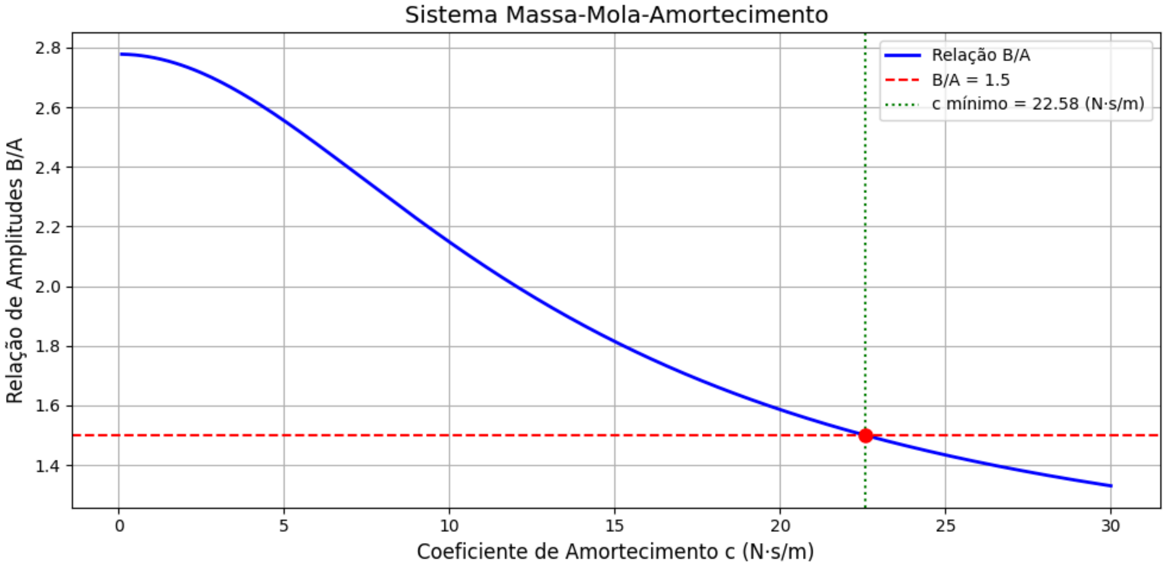
$$D = \frac{\left(\frac{\omega}{p}\right)^2}{\sqrt{\left[1-\left(\frac{\omega}{p}\right)^2\right]^2 + \left(\frac{2\zeta\omega}{p}\right)^2}} \quad e \quad tan(\phi) = \frac{\frac{2\zeta\omega}{p}}{1-\left(\frac{\omega}{p}\right)^2}$$

com $p = \sqrt{\frac{k}{m}}$ sendo a frequência natural do sistema e $\zeta = \frac{c}{2mp}$ o fator de amortecimento. Considerando, $m = 0,2\,kg$, $k = 2880\,\frac{N}{m}$, e $\omega = 96\,\frac{rad}{s}$. Determine o menor valor de $c$ (coeficiente de amortecimento) de modo que a relação $\frac{B}{A}$ seja menor ou igual a $1,5$.

In [37]:
```
import numpy as np
import matplotlib.pyplot as plt

m = 0.2
k = 2880
omega = 96
BA_questao = 1.5

p = np.sqrt(k / m)

def razao_BA(zeta):
    r = omega / p
    D = (r**2) / np.sqrt((1 - r**2)**2 + (2*zeta*r)**2)
    phi = np.arctan2(2*zeta*r, 1 - r**2)
    return np.sqrt((1 + D*np.cos(phi))**2 + (D*np.sin(phi))**2)


def bisseccao(f, a, b, tol=1e-10, iteracoes=100):
    if f(a) * f(b) >= 0:

        while f(b) < 0:
            b *= 2
        if f(a) * f(b) >= 0:
            raise ValueError("Não há mudança de sinal no intervalo")

    for _ in range(iteracoes):

        c = (a + b) / 2
        if abs(f(c)) < tol:
            return c
        if f(c) * f(a) < 0:
            b = c
        else:
            a = c

    return (a + b) / 2


try:

    def funcao_alvo(zeta):
        return razao_BA(zeta) - BA_questao

    zeta_final = bisseccao(funcao_alvo, 0.01, 1.0)
```

```
    c_final = 2 * m * p * zeta_final

    print(f"O menor valor do Coeficiente de Amortecimento (c), que satisfaz B/A


    valores_de_C = np.linspace(0.1, 30, 300)
    valores_BA = [razao_BA(c/(2*m*p)) for c in valores_de_C]

    plt.figure(figsize=(10, 5))
    plt.plot(valores_de_C, valores_BA, color='blue', linewidth=2, label='Relaçãc
    plt.axhline(y=BA_questao, color='red', linestyle='--', label=f'B/A = {BA_que
    plt.axvline(x=c_final, color='green', linestyle=':', label=f'c mínimo = {c_f
    plt.plot(c_final, BA_questao, color='red', marker="o", markersize=8)

    plt.title('Sistema Massa-Mola-Amortecimento', fontsize=14)
    plt.xlabel('Coeficiente de Amortecimento c (N·s/m)', fontsize=12)
    plt.ylabel('Relação de Amplitudes B/A', fontsize=12)
    plt.grid(True)
    plt.legend(fontsize=10)
    plt.tight_layout()
    plt.show()

except ValueError as e:
    print(f"Erro: {str(e)}")
```

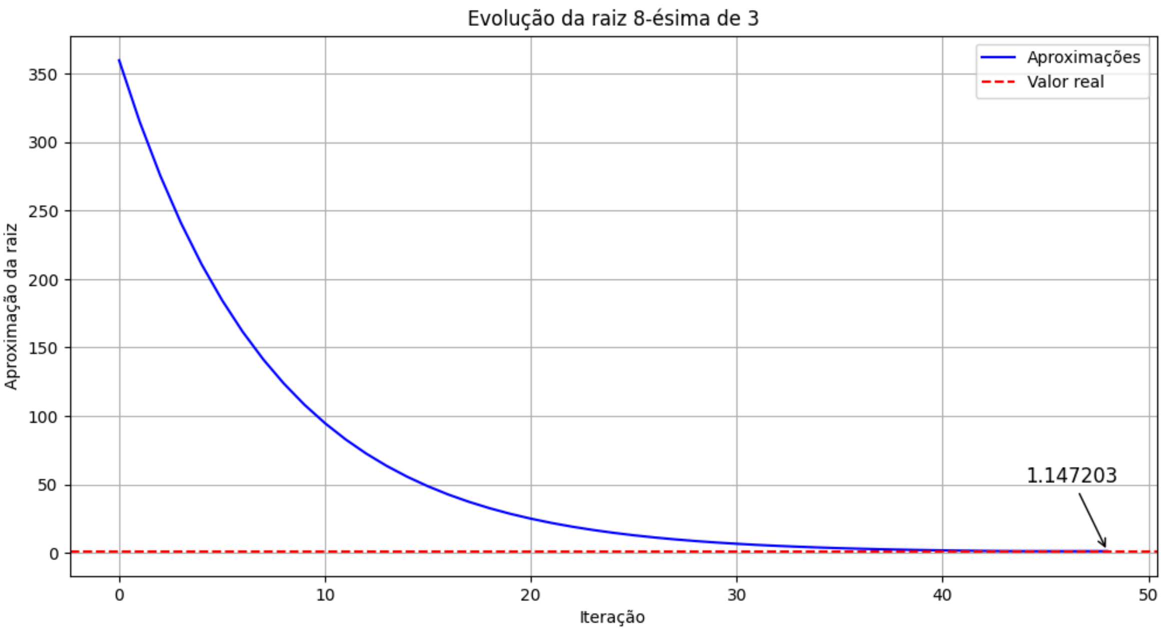
```
O menor valor do Coeficiente de Amortecimento (c), que satisfaz B/A <= 1.5 é: 22.
584242 N·s/m
```




**Exercício 04** - Escreva uma função $raiz\_\ enesima(x, n, tol)$ para calcular $r = \sqrt[n]{x}$, por meio do método de Newton-Raphson.

In [40]:
```
import numpy as np
import matplotlib.pyplot as plt

def raiz_enesima(x, n, tol=1e-10):

    r = x / n
    valores = []

    for _ in range(1000):
        novo_valor = r - (r**n - x) / (n * r**(n - 1))
        valores.append(novo_valor)
```

```
        erro_relativo = np.abs(novo_valor - r) / np.abs(novo_valor)

        if erro_relativo < tol:
            break
        else:
            r = novo_valor


    valor_real = x**(1/n)

    plt.figure(figsize=(12, 6))
    plt.plot(valores, label="Aproximações", color="blue")
    plt.axhline(y=valor_real, linestyle='--', label="Valor real", color="red")
    plt.annotate(f"{valor_real:.6f}",
                 xy=(len(valores)-1, valor_real),
                 xytext=(len(valores) - 5, valor_real + 50),
                 arrowprops=dict(arrowstyle="->", color='black'),
                 fontsize=12)
    plt.xlabel("Iteração")
    plt.ylabel("Aproximação da raiz")
    plt.title(f"Evolução da raiz {n}-ésima de {x}")
    plt.grid(True)
    plt.legend()
    plt.show()

    return r

x = 3
n = 8
resultado = raiz_enesima(x, n)
print(f"A raiz {n}-ésima de {x} é aproximadamente {resultado}")
```




```
A raiz 8-ésima de 3 é aproximadamente 1.1472026904497181
```