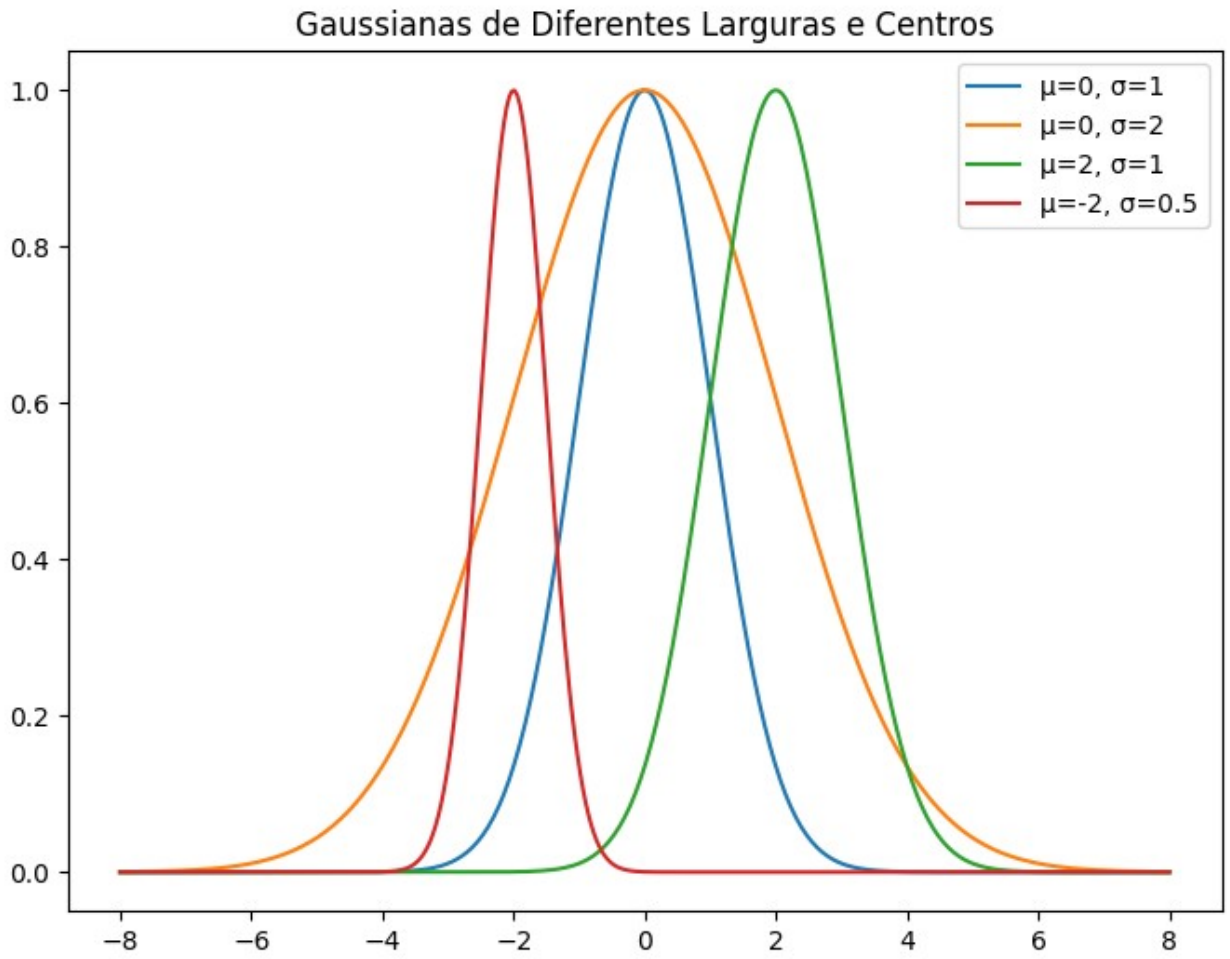
```
import numpy as np
import matplotlib.pyplot as plt
```

Gaussianas de diferentes larguras e centros

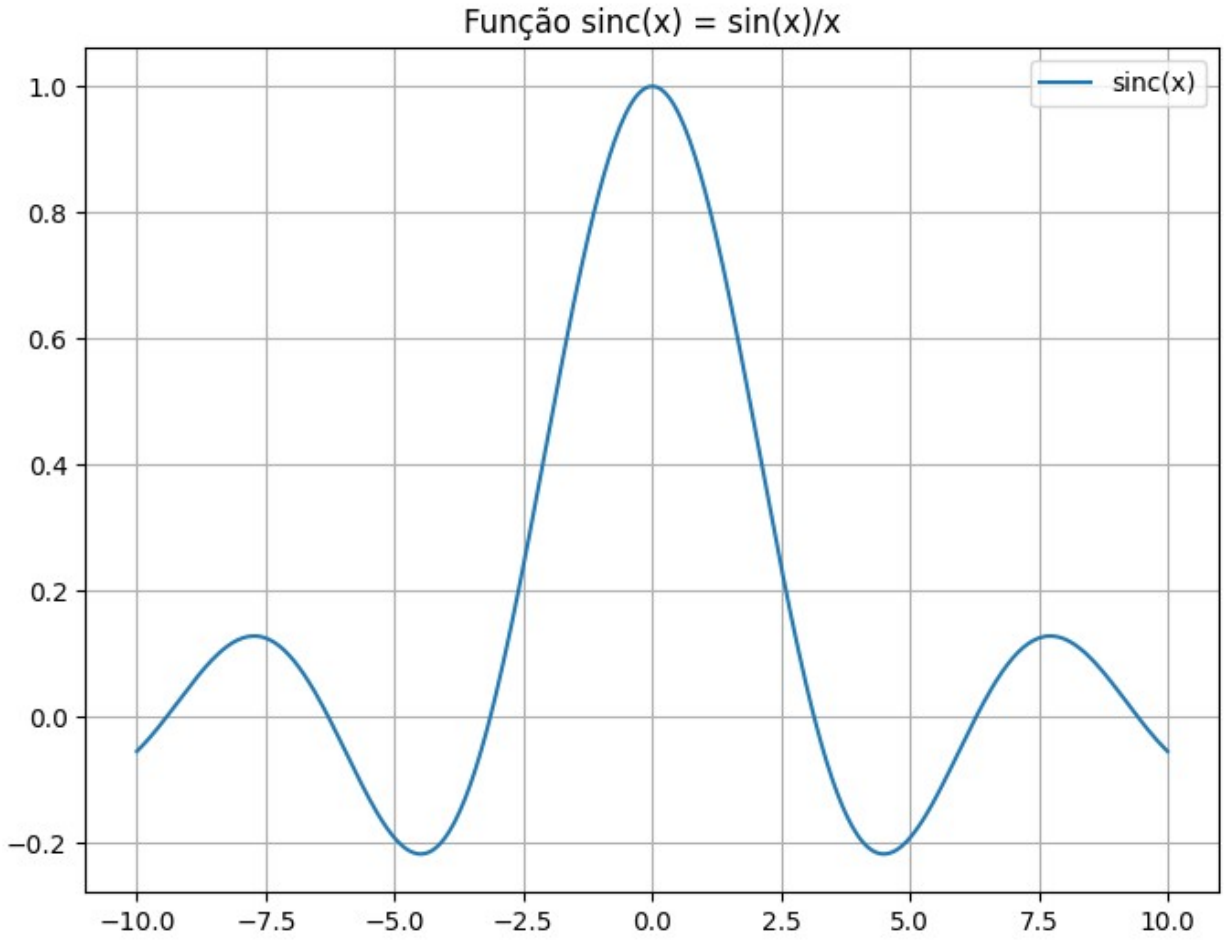
```
import numpy as np
import matplotlib.pyplot as plt
'''
 Plota curvas gaussianas com diferentes larguras e centros.

 A função gera e exibe gráficos de funções gaussianas
 com diferentes valores , ilustrando como esses parâmetros
 afetam a forma da distribuição.

 Exibe um gráfico com múltiplas curvas gaussianas.
'''

labelsize1='20'
fig = plt.figure(figsize=(8, 6))

x_gauss = np.linspace(-8, 8, 400)
parametros = ([(0, 1), (0, 2), (2, 1), (-2, 0.5)])
for mu, sigma in parametros:
    y_gauss = np.exp(-((x_gauss - mu) ** 2) / (2 * sigma ** 2))
    plt.plot(x_gauss, y_gauss, label=f"μ={mu}, σ={sigma}")


plt.title("Gaussianas de Diferentes Larguras e Centros")
plt.legend()
plt.show()
```



Função $sinc(x) = \frac{sin(x)}{x}$

```
labelsize1='20'
fig = plt.figure(figsize=(8, 6))
'''
Plota a função sinc(x), definida como sin(x)/x.

A função usa a implementação do NumPy para sinc, que já normaliza a
função
corretamente como sin(\pi.x)/(\pi.x).

O gráfico é gerado para x variando de -10 a 10 com 400 pontos.
'''


x_sinc = np.linspace(-10, 10, 400)

y_sinc = np.sinc(x_sinc / np.pi)
```

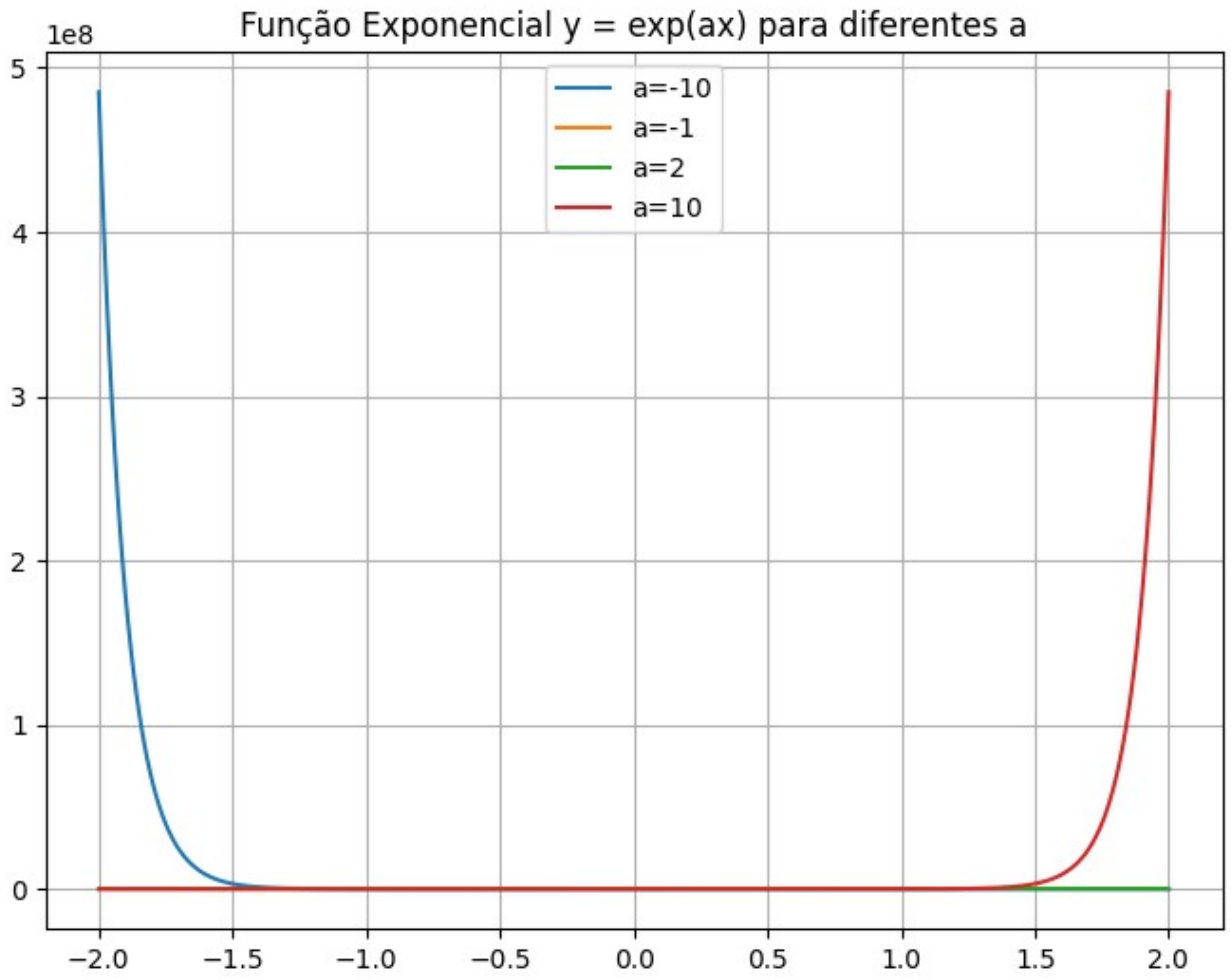
```
plt.plot(x_sinc, y_sinc, label="sinc(x)")
plt.title("Função sinc(x) = sin(x)/x")
plt.legend()
plt.grid()
plt.show()
```



Exponencial $y=e^{|ax|}$, para diferentes valores de a

```
labelsize1='20'
fig = plt.figure(figsize=(8, 6))
'''
Plota a função exponencial y = exp(a*x) para diferentes valores de a.

O gráfico é gerado para x variando de -2 a 2 com 400 pontos.
Os valores de 'a' utilizados são [-10, -1, 2, 10].
Cada curva representa uma exponencial diferente com um valor
específico de a.
'''
```

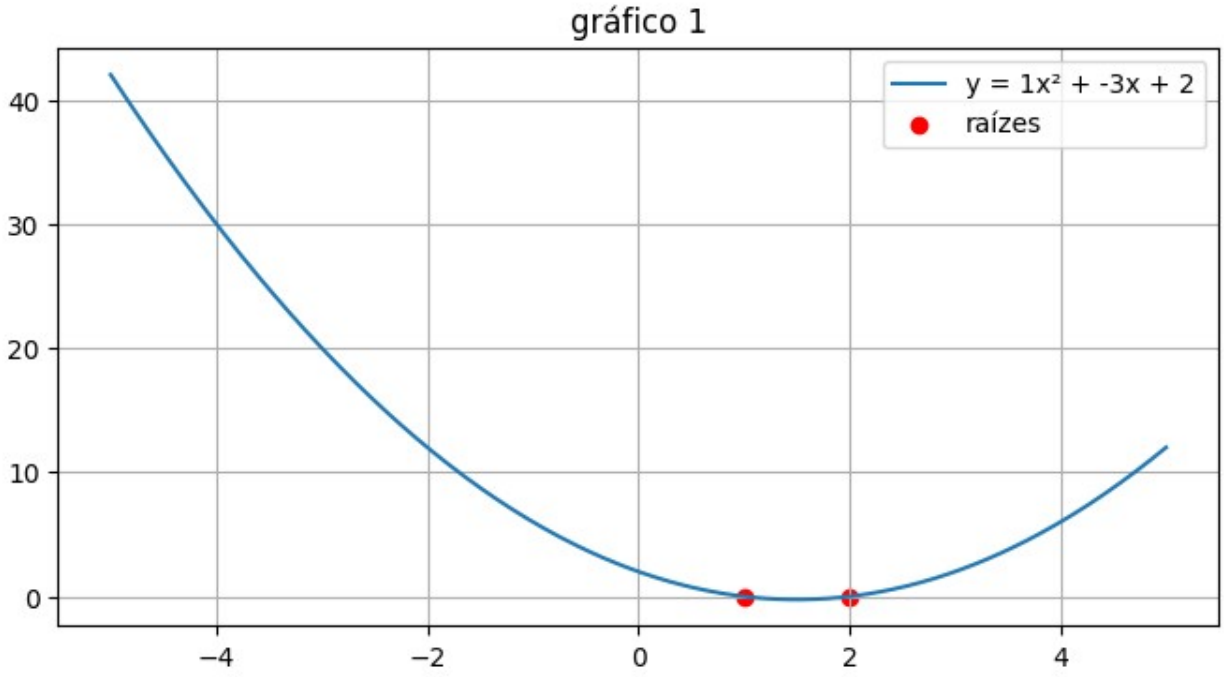
```python
#gráfico de -2 até 2 com 400 pontos
x_exp = np.linspace(-2, 2, 400)
#lista com valores de a
a_valores = [-10, -1, 2, 10]
for a in a_valores:
    y_exp = np.exp(a * x_exp)
    plt.plot(x_exp, y_exp, label=f"a={a}")

plt.title("Função Exponencial y = exp(ax) para diferentes a")
plt.legend()
plt.grid()
plt.show()
```



Considerando a função $y = a x^2 + b x + c$, achando raizes r_1 e r_2 , com gráfico indicando as raizes

```
'''
Calcula as raízes reais de uma equação quadrática da forma ax² + bx +
c = 0
usando a fórmula de Bhaskara.

Parâmetros:
 a (float): Coeficiente quadrático (deve ser diferente de zero).
 b (float): Coeficiente linear.
 c (float): Termo constante.

Retorno:
  tupla: um par de valores (r1, r2) representando as raízes da
equação.
  se delta > 0, retorna duas raízes reais distintas.
  se delta == 0, retorna uma raiz real dupla.
  se delta < 0, retorna (None, None), indicando ausência de raízes
reais
'''


def bhaskara(a, b, c):
    delta = b**2 - 4*a*c
    if delta > 0:
        r1 = (-b + np.sqrt(delta)) / (2*a)
        r2 = (-b - np.sqrt(delta)) / (2*a)
    elif delta == 0:
        r1 = r2 = -b / (2*a)
    else:
        r1 = r2 = None  # Sem raízes reais
    return r1, r2

# Casos com discriminante positivo, zero e negativo
casos = [(1, -3, 2), (1, -2, 1), (1, 1, 1)]

for i, (a, b, c) in enumerate(casos):
    x = np.linspace(-5, 5, 400)
    y = a*x**2 + b*x + c
    r1, r2 = bhaskara(a, b, c)

    plt.figure(figsize=(8, 4))
    plt.plot(x, y, label=f"y = {a}x² + {b}x + {c}")

    # Indicar raízes reais, se existirem
    if r1 is not None and r2 is not None:
        plt.scatter([r1, r2], [0, 0], color='red', marker='o',
label="raízes")
```

```
    elif r1 is not None:
        plt.scatter([r1], [0], color='red', marker='o', label="raiz
dupla")


    plt.title(f"gráfico {i+1}")
    plt.legend()
    plt.grid()
    plt.show()
```



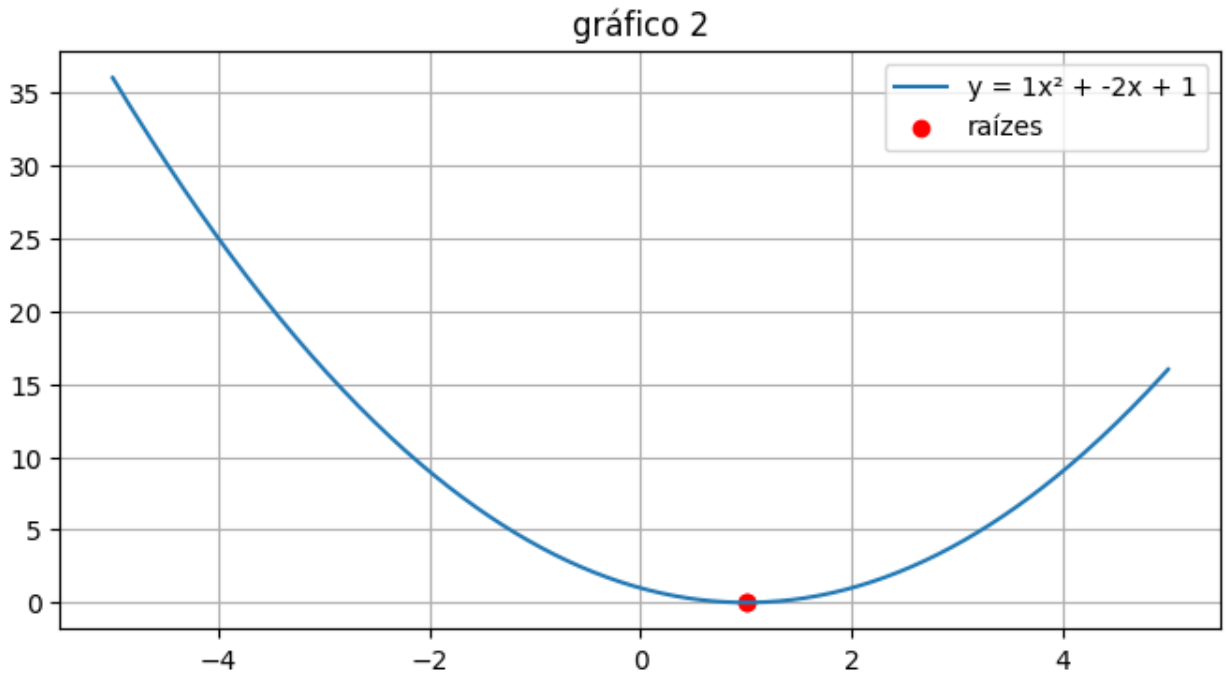


gráfico 2
y = 1x² + -2x + 1
raízes
35
30
25
20
15
10
5
0
−4
−2
0
2
4

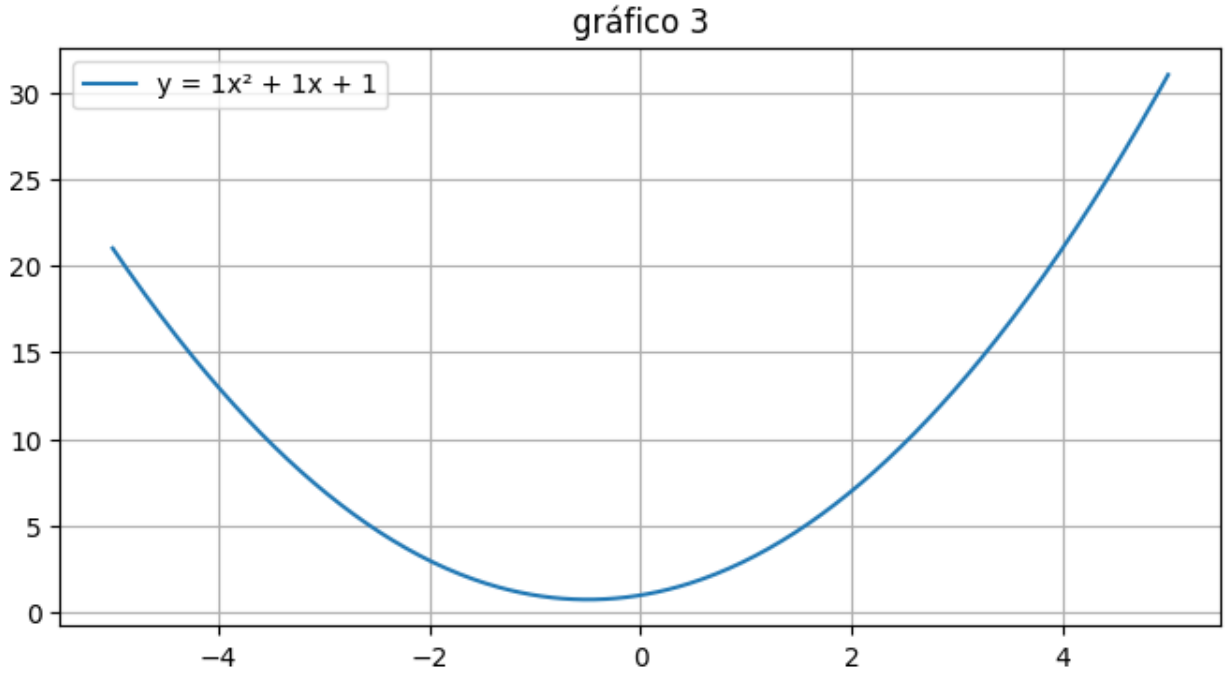


gráfico 3
y = 1x² + 1x + 1
30
25
20
15
10
5
0
−4
−2
0
2
4