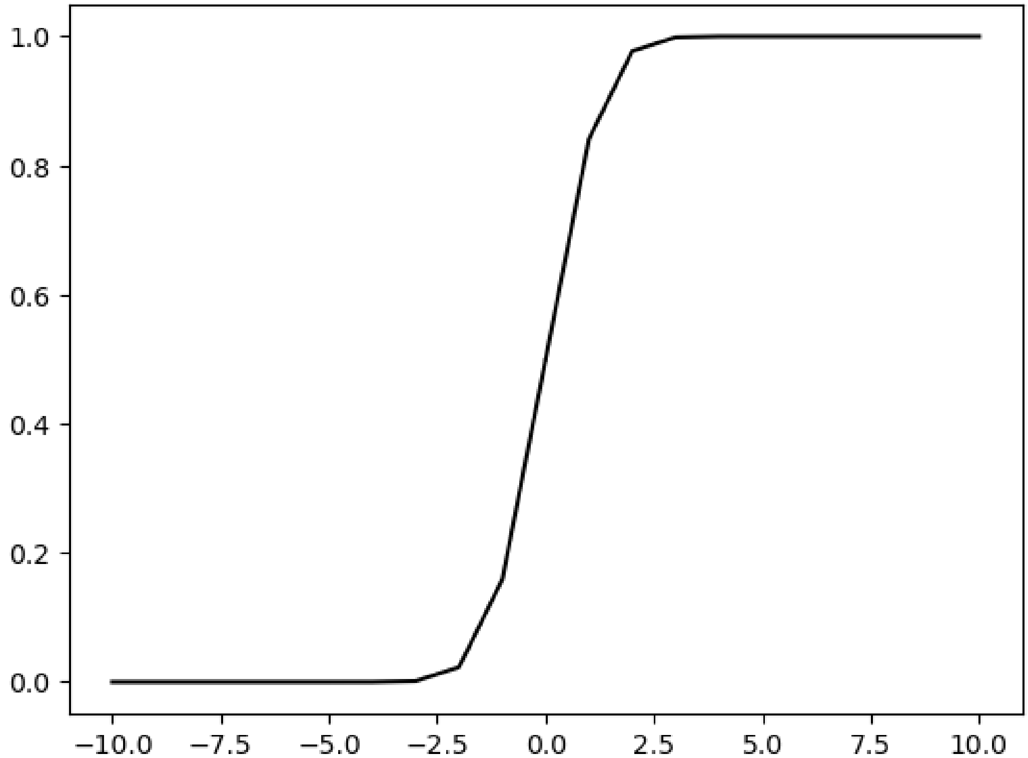
Lista_tarefas

October 7, 2025

1 Tarefa 1

[96]:
```
import numpy as np
import matplotlib.pyplot as plt

# Valores do parâmetro p_i
p_i = [0.231641900, 0.319381530, -0.356563782, 1.781477937, -1.821255978, 1.
 ↪330274429]

# Função w
def w (y):
    return 1 / (1 + (p_i[0] * abs(y)))

# Função z
def z (w):
    return (w * (p_i[1] + w * (p_i[2] + w * (p_i[3] + w * (p_i[4] + w *␣
 ↪p_i[5])))))

# Aproximação analítica para a função de probabilidade acumulada
# Para y negativo
def phi_neg (z , y):
    return (z / np.sqrt( 2 * np.pi)) * np.exp(- (y**2 / 2))

# Para y positivo
def phi_pos (z , y):
    return 1 - (z / np.sqrt( 2 * np.pi)) * np.exp(- (y**2 / 2))

# Vetores para guardar os resultados
phi_neg_results = []
phi_pos_results = []
y_neg = []
y_pos = []

# Verificação das funções
for y in range(-10 , 11, 1):
    if y < 0:
        w_calc = w(y)
```

```python
        z_calc = z(w_calc)
        phi_neg_calc = phi_neg(z_calc, y)
        phi_neg_results.append(phi_neg_calc)
        y_neg.append(y)
    elif y == 0:
        w_calc = w(y)
        z_calc = z(w_calc)
        phi_neg_calc = phi_neg(z_calc, y)
        phi_neg_results.append(phi_neg_calc)
        y_neg.append(y)
        phi_pos_calc = phi_pos(z_calc, y)
        phi_pos_results.append(phi_pos_calc)
        y_pos.append(y)
    else:
        w_calc = w(y)
        z_calc = z(w_calc)
        phi_pos_calc = phi_pos(z_calc, y)
        phi_pos_results.append(phi_pos_calc)
        y_pos.append(y)

# Plotagem dos resultados
fig, ax = plt.subplots()
ax.plot(y_neg, phi_neg_results, color='black')
ax.plot(y_pos, phi_pos_results, color='black')
plt.show()
```



[97]:

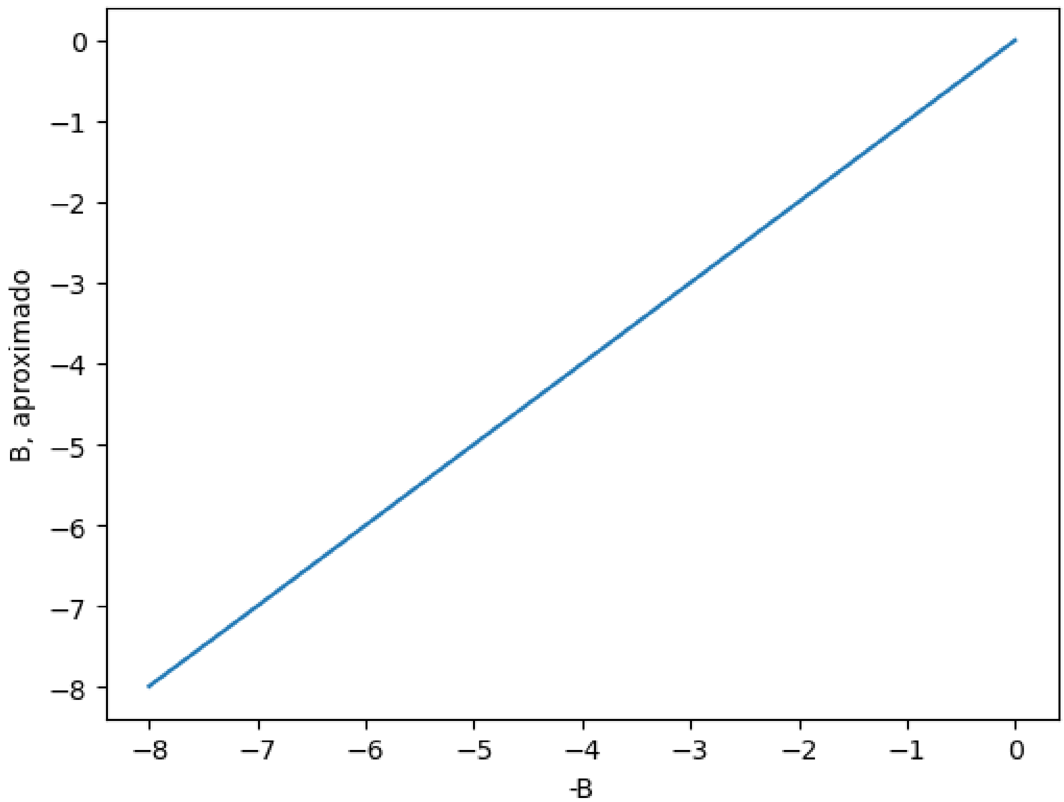
```
# Formulação da função inversa

# Valores do parâmetro p_i
p = [-0.3222324310880, -1.0000000000000, -0.3422422088547, -0.2042312102450e-1,␣
 ↪-0.4536422101480e-4]

# Valores do parâmetro q_i
q = [0.9934846260600e-1, 0.5885815704950, 0.5311034623660, 0.10353775285000, 0.
 ↪3856070063400e-2]

# Função inversa
# Para 0 < u <= 0.5
def y_1 (u):
    z = np.sqrt(np.log(1 / (u ** 2)))
    return -z - ((p[0] + z * (p[1] + z * (p[2] + z * (p[3] + z * (p[4]))))) /␣
 ↪(q[0] + z * (q[1] + z * (q[2] + z * (q[3] + z * (q[4]))))))

# Para 0.5 <= u < 1
def y_2 (u):
    z = np.sqrt(np.log (1 / ((1 - u) ** 2)))
```

```
    return z + ((p[0] + z * (p[1] + z * (p[2] + z * (p[3] + z * (p[4]))))) /
↪(q[0] + z * (q[1] + z * (q[2] + z * (q[3] + z * (q[4]))))))
```

[117]:

```
# Verificação da implementação

vetor_beta = []
vetor_beta_aprox = []

for i in np.arange (0, 9, 1):
    w_calc = w(i)
    z_calc = z(w_calc)
    vetor_beta.append(-1 * i)
    result = 1- phi_pos(z_calc, i)
    if result > 0:
        if result <= 0.5:
            inverse_result = y_1(result)
        else:
            inverse_result = y_2(result)
        vetor_beta_aprox.append(inverse_result)

print(vetor_beta)
print(vetor_beta_aprox)
plt.plot(vetor_beta, vetor_beta_aprox)
plt.xlabel('-B')
plt.ylabel('B, aproximado')
plt.show()
```

```
[np.int64(0), np.int64(-1), np.int64(-2), np.int64(-3), np.int64(-4),
np.int64(-5), np.int64(-6), np.int64(-7), np.int64(-8)]
[np.float64(8.160186326655605e-08), np.float64(-0.999999850223544),
np.float64(-2.0000011964816147), np.float64(-2.999984266933476),
np.float64(-3.999889395168855), np.float64(-4.9996951700643155),
np.float64(-5.999419242585658), np.float64(-6.999097624841499),
np.float64(-7.991573888288796)]
```


0
−1
−2
−3
−4
−5
−6
−7
−8
B, aproximado
−8
−7
−6
−5
−4
−3
−2
−1
0
-B