1.

$$X_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \nabla\phi(x) = \begin{bmatrix} 2x_1 - 1 \\ x_2 - 1 \end{bmatrix}$$

$$P_0 = -\nabla\phi(x_0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$P_0 \neq 0 \Rightarrow \alpha_0 = \frac{\nabla\phi(x_0)^T P_0}{P_0^T A P_0} = \frac{\begin{bmatrix} 1 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix}}{\begin{bmatrix} 1 & 1 \end{bmatrix}\begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix}} = \frac{2}{3}$$

$$X_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \frac{2}{3}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2/3 \\ 2/3 \end{bmatrix}$$

$$\nabla\phi(x_1) \neq 0$$

$$P_1 = \frac{\begin{bmatrix} 1 & 1 \end{bmatrix}\begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}\left(\begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 2/3 \\ 2/3 \end{bmatrix}\right)}{\begin{bmatrix} 1 & 1 \end{bmatrix}\begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix}} = \frac{1}{9}$$

$$P_1 = \begin{bmatrix} -\frac{1}{3} \\ \frac{1}{3} \end{bmatrix} + \frac{1}{9}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -\frac{2}{9} \\ \frac{4}{9} \end{bmatrix}$$

$$\alpha_1 = \frac{\frac{2}{9}}{\left(\frac{2}{9}\right)\left(-\frac{4}{9}\right) + \left(\frac{4}{9}\right)\left(\frac{4}{9}\right)} = \frac{3}{4}$$

$$X_2 = \begin{bmatrix} 2/3 \\ 2/3 \end{bmatrix} + \begin{bmatrix} -\frac{2}{9} \\ \frac{4}{9} \end{bmatrix}\begin{bmatrix} \frac{3}{4} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ 1 \end{bmatrix}$$

$$\nabla\phi\left(\begin{bmatrix} \frac{1}{2} \\ 1 \end{bmatrix}\right) = 0$$

Answer: $X^* = \begin{bmatrix} \frac{1}{2} \\ 1 \end{bmatrix}$

2
a)

$\nabla_y q(y) = \nabla F(x) + C(y-x)$

$\nabla q(y^*) = 0 \Rightarrow \nabla F(x) + C(y^*-x) = 0$

$\Rightarrow y^* = x - \frac{1}{C} \cdot \nabla F(x)$

$\nabla_y^2 q(y) = C \cdot I > 0 \Rightarrow$ minimizer

b) $q(y) = F(x) + \nabla F(x)^T (y-x) + \frac{C}{2} \| x-y \|$

$q(y^*) = q\left(x - \frac{1}{C} \nabla F(x)\right) = F(x) + \nabla F(x)\left(x - \frac{1}{C} \nabla F(x) - x\right)$

$+ \frac{C}{2} \| x - x + \frac{1}{C} \nabla F(x) \|$

$= F(x) - \frac{1}{C} \| \nabla F(x) \|^2 + \frac{1}{2C} \| \nabla F(x) \|^2$

$= F(x) - \frac{1}{2C} \| \nabla F(x) \|^2$

c) Strong convexity implies

$\forall x, y \in \mathbb{R}^d, F(y) \geq q(y)$

$\Rightarrow F(y^*) \geq F(x) - \frac{1}{2C} \| \nabla F(x) \|^2$

$\Rightarrow f(x) - F(y^*) \leq \frac{1}{2C} \| \nabla F(x) \|^2$

x^* is the global minimizer

which means:

$F(x) - F(y^*) \leq F(x) - F(x^*) \leq \frac{1}{2C} \| \nabla F(x) \|^2$

3. A differentiable function $f: \mathbb{R}^d \mapsto \mathbb{R}$ is strongly convex if:

$\exists\, c > 0$ s.t. $\forall\, x, y \in \mathbb{R}^d$

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) + \frac{1}{2} c \|y - x\|^2$$

---

a) $$y - x - (y - x) \geq \frac{c}{2} \|y - x\|^2$$

$$-2x \geq \frac{c}{2} \|y - x\|^2$$

This inequality doesn't hold for any $c > 0$ when $x > 0$.

Answer: $f(x) = x$ is not strongly convex

3. A twice differentable function

b) $f: \mathbb{R}^d \mapsto \mathbb{R}$ is strongly convex with constant c if:

$$v^T \nabla^2 f(x)\, v \geq c \cdot \|v\|^2$$

$$\forall\, x, y \in \mathbb{R}^d \quad \text{and} \quad \exists\, c > 0$$

$$f(x) = x^2 \rightarrow f''(x) = 2$$

$\Rightarrow$ $f(x)$ is strongly convex with constant $c = 2$.

3 c)

Theorem: A twice differentiable function $f: \mathbb{R}^d \mapsto \mathbb{R}$ is strongly convex with positive constant c if:

(A) $v^T \nabla^2 f(x) v \geq c \|v\| \quad \forall x, y \in \mathbb{R}^d$

$$f(x) = \log(1+e^x) \rightarrow f''(x) = \frac{e^x}{(1+e^x)^2}$$

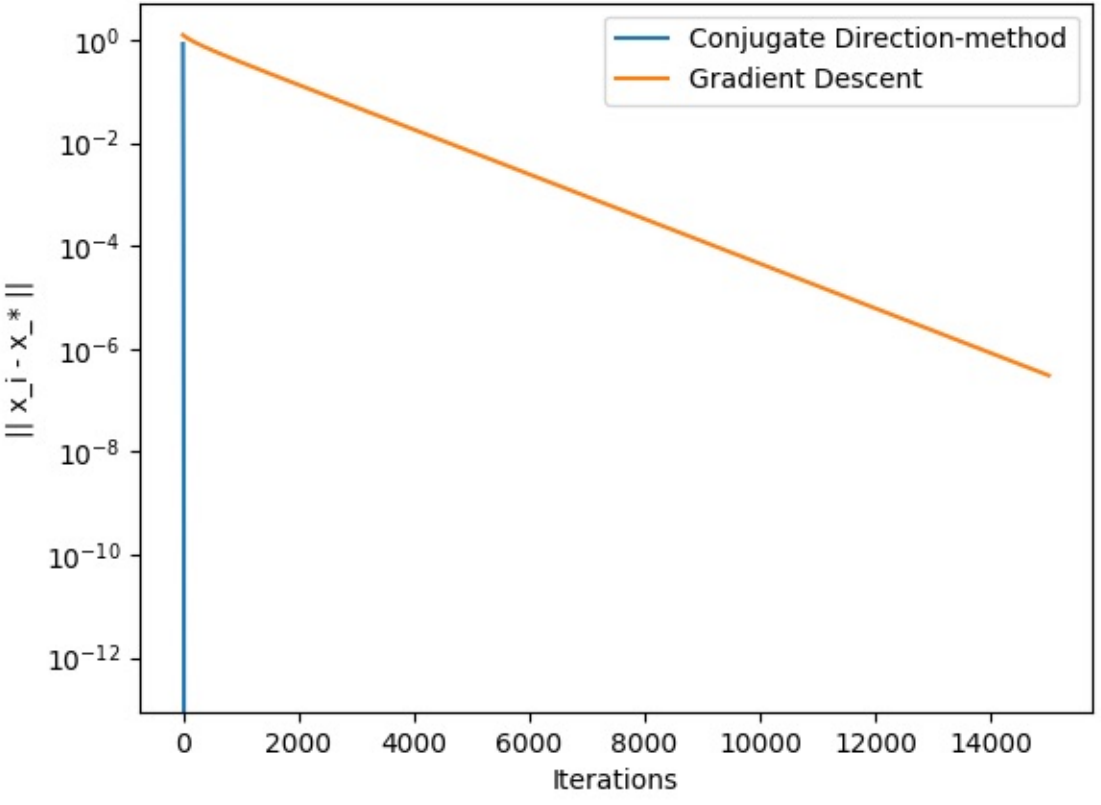
$$\lim_{x \to \infty} f''(x) = 0$$

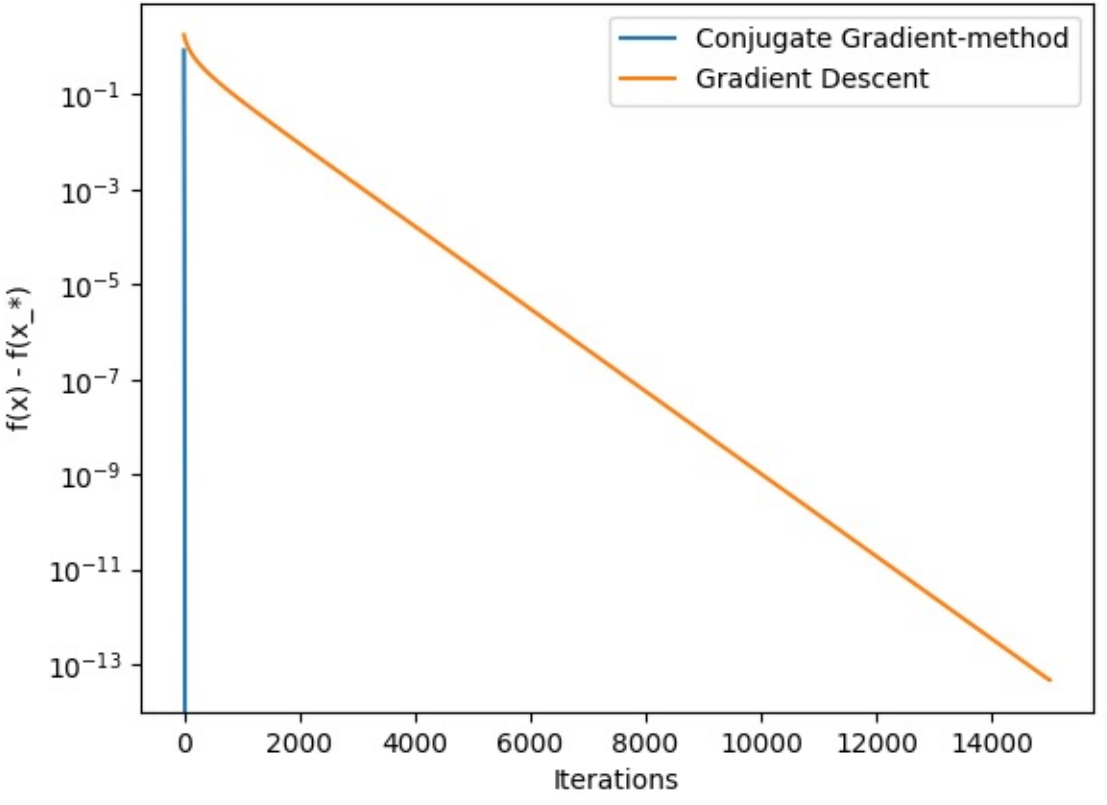
$\Rightarrow$ There is no positive constan $c > 0$ that fulfills inequality (A) above

Answer: $f(x) = \log(1+e^x)$ is not strongly convex.

```python
import numpy as np
from scipy.sparse.linalg import cg

def create_system(n):
    A = np.diag(np.arange(1, n + 1))
    b = np.ones(n)
    return A, b

def solve_direct(A, b):
    return np.linalg.solve(A, b)

for n in [20, 100]:
    print(f"\nRunning for n = {n}")
    A, b = create_system(n)
    x0 = np.zeros(n)

    x_star = solve_direct(A, b)
    print("Optimal x*:", x_star)
```

```python
from typing import Callable, List, Tuple
import numpy as np
from scipy.optimize import minimize_scalar
import matplotlib.pyplot as plt


dim = 20
iterations = 15000


twenty =  np.array([1.0, 0.5, 0.33333333, 0.25, 0.2, 0.16666667,
                    0.14285714, 0.125, 0.11111111, 0.1, 0.09090909, 0.08333333,
                    0.07692308, 0.07142857, 0.06666667, 0.0625, 0.05882353, 0.05555556,
                    0.05263158, 0.05])
hundred = np.array([
    1.,         0.5,        0.33333333, 0.25,       0.2,        0.16666667,
    0.14285714, 0.125,      0.11111111, 0.1 ,       0.09090909, 0.08333333,
    0.07692308, 0.07142857, 0.06666667, 0.0625 ,    0.05882353 ,0.05555556,
    0.05263158, 0.05 ,      0.04761905, 0.04545455, 0.04347826, 0.04166667,
    0.04,       0.03846154, 0.03703704, 0.03571429, 0.03448276, 0.03333333,
    0.03225806, 0.03125 ,   0.03030303, 0.02941176, 0.02857143, 0.02777778,
    0.02702703, 0.02631579, 0.02564103, 0.025     , 0.02439024, 0.02380952,
    0.02325581, 0.02272727, 0.02222222, 0.02173913, 0.0212766 , 0.02083333,
    0.02040816, 0.02      , 0.01960784, 0.01923077, 0.01886792, 0.01851852,
    0.01818182, 0.01785714, 0.01754386, 0.01724138, 0.01694915, 0.01666667,
    0.01639344, 0.01612903, 0.01587302, 0.015625  , 0.01538462, 0.01515152,
    0.01492537, 0.01470588, 0.01449275, 0.01428571, 0.01408451, 0.01388889,
    0.01369863, 0.01351351, 0.01333333, 0.01315789, 0.01298701, 0.01282051,
    0.01265823, 0.0125    , 0.01234568, 0.01219512, 0.01204819, 0.01190476,
    0.01176471, 0.01162791, 0.01149425, 0.01136364, 0.01123596, 0.01111111,
    0.01098901, 0.01086957, 0.01075269, 0.0106383 , 0.01052632, 0.01041667,
    0.01030928, 0.01020408, 0.01010101, 0.01])
optimal_solutions = [twenty, hundred]

def function(x:np.ndarray) -> np.ndarray:
    n = x.shape[0]
    diagonal_elements = np.arange(1, n + 1)
    matrix = np.diag(diagonal_elements)
    return 0.5 * np.dot(x.T, np.dot(matrix, x)) - np.dot(np.array([1 for _ in range(n)]).T, x)

def function_of_a(x: np.ndarray, conjugate_vector:np.ndarray):
    def inner_function(a: float) -> float:
        n = x.shape[0]
        diagonal_elements = np.arange(1, n + 1)
        matrix = np.diag(diagonal_elements)
        return 0.5 * np.dot((x + a * conjugate_vector).T, np.dot(matrix, (x + a * conjugate_vector))) - np.dot(np.ones(n), (x + a * conjugate_vector))
    return inner_function

def gradient(x:np.ndarray) -> np.ndarray:
    n = x.shape[0]
    diagonal_elements = np.arange(1, n + 1)
    matrix = np.diag(diagonal_elements)
    return np.dot(matrix, x) - np.ones(n).T


def gradient_descent(optimal_solution,step_size:float=1e-3) -> Tuple[List[np.ndarray], List[np.ndarray]]:
    x = np.array([0 for _ in range(dim)], dtype=float)
    xlist = []
    ylist = []
    for _ in range(iterations):
        x -= step_size * gradient(x)
        xlist.append(np.linalg.norm(x - optimal_solution))
        ylist.append(function(x)-function(optimal_solution))
    print("\nGradient descent: \n\nx: ",x, "\n\nf(x): ", function(x), "\n")
    return xlist, ylist

def conjugate_direction(optimal_solution) -> Tuple[List[np.ndarray], List[np.ndarray]]:
    x = np.array([0 for _ in range(dim)], dtype=float)
    n = x.shape[0]
    vectors = []
    xlist = []
    ylist = []
    for i in range(n):
        array = np.zeros(n)
        array[i] = 1
        vectors.append(array)

    for vector in vectors:
        f = function_of_a(x, vector)
        optimal_a = minimize_scalar(f).x
        x += optimal_a * vector
        xlist.append(max(np.linalg.norm(x - optimal_solution), 1e-16))
        ylist.append(max(function(x)-function(optimal_solution), 1e-16))
    print("\nConjugate descent: \n\nx: ",x, "\n\nf(x): ", function(x), "\n")
    return xlist,ylist

gd_iter = [i for i in range(iterations)]

for optimal_solution in optimal_solutions:
    cg_iter = [i for i in range(len(optimal_solution))]
    x_c , y_c = conjugate_direction(optimal_solution)
    x_gd , y_gd = gradient_descent(optimal_solution)

    plt.figure()
    plt.plot(cg_iter, y_c, label='Conjugate Direction-method')
    plt.plot(gd_iter, x_gd, label='Gradient Descent')
    plt.xlabel('Iterations')
    plt.ylabel('|| x_i - x_* ||')
    plt.yscale('log')
    plt.legend()
    plt.savefig(f'convergence_comparison_{optimal_solution[0]}.png')
    plt.show()


    plt.figure()
    plt.plot(cg_iter, y_c, label= 'Conjugate Direction-method')
    plt.plot(gd_iter, y_gd, label='Gradient Descent')
    plt.xlabel('Iterations')
    plt.ylabel('f(x) - f(x_*)')
    plt.yscale('log')
    plt.legend()
    plt.savefig(f'function_value_comparison_{optimal_solution[0]}.png')
    plt.show()
```


Conjugate Direction-method
Gradient Descent
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{-12}
|| x_i - x_* ||
0
2000
4000
6000
8000
10000
12000
14000
Iterations


Conjugate Gradient-method
Gradient Descent
10^{-1}
10^{-3}
10^{-5}
10^{-7}
10^{-9}
10^{-11}
10^{-13}
f(x) - f(x_*)
0
2000
4000
6000
8000
10000
12000
14000
Iterations