CGD Convergence Evaluation

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

We will use a simple standard quardatic function to start with.

Let $A \in \mathbb{R}^{n \times n}$ be a diagonal matrix with diagonal entries

$$A_{ii} = i, \quad \text{i.e., the entries run from } 1 \text{ to } n,$$

and let $b \in \mathbb{R}^n$ be a vector with all entries equal to 1. Define the function

$$f(x) = \frac{1}{2} x^T A x - b^T x.$$

We want to compare the convergence behavior of the conjugate gradient (version 0 or 1) and gradient descent methods. Perform the following tasks for $n = 20$ and $n = 100$ with initialization $x^{(0)} = 0$.

`Find the optimal result first for comparison that we are gonna do`

In [2]:
```
def f(x, A, b):
    return 0.5 * x.T @ A @ x - b.T @ x

def grad_f(x, A, b):
    return A @ x - b

n_values = [2, 100]
results = {}
```

Gradient Descent

1. **Initialization:**
   Set $x^{(0)} = 0$ (or any initial guess), choose a step size $\alpha > 0$, and let $t = 0$.
2. **Repeat for $t = 0, 1, \ldots, T - 1$:**
   - Compute the gradient:
   $$\nabla f(x^{(t)}) = Ax^{(t)} - b.$$
   - Update the solution:
   $$x^{(t+1)} = x^{(t)} - \alpha \nabla f(x^{(t)}).$$
3. **Stop after T iterations.**
   - Both methods can stop after $T = n$ iterations or based on convergence criteria such as:
   $$f(x^{(t)}) - f(x^*) \leq \epsilon,$$
   where $\epsilon > 0$ is a small tolerance.

In [3]:
```
def gradient_descent(A, b, x0, stepsize, iterations):
    x = x0.copy()
    f_values = []
    x_values = []
    for _ in range(iterations):
        f_values.append(f(x, A, b))
        x_values.append(x.copy())
        x -= stepsize * grad_f(x, A, b)
    return np.array(f_values), np.array(x_values)
```

Conjugate Gradient

1. **Initialization:**
   Set $x^{(0)} = 0$, $r^{(0)} = b - Ax^{(0)}$, $p^{(0)} = r^{(0)}$, and let $t = 0$.

2. **Repeat for $t = 0, 1, \ldots, T-1$:**

   - Compute $Ap^{(t)}$:
   $$Ap^{(t)}.$$
   - Compute the step size $\alpha^{(t)}$:
   $$\alpha^{(t)} = \frac{(r^{(t)})^T r^{(t)}}{(p^{(t)})^T A p^{(t)}}.$$
   - Update the solution:
   $$x^{(t+1)} = x^{(t)} + \alpha^{(t)} p^{(t)}.$$
   - Update the residual:
   $$r^{(t+1)} = r^{(t)} - \alpha^{(t)} A p^{(t)}.$$
   - Compute the conjugate direction coefficient $\beta^{(t)}$:
   $$\beta^{(t)} = \frac{(r^{(t+1)})^T r^{(t+1)}}{(r^{(t)})^T r^{(t)}}.$$
   - Update the search direction:
   $$p^{(t+1)} = r^{(t+1)} + \beta^{(t)} p^{(t)}.$$

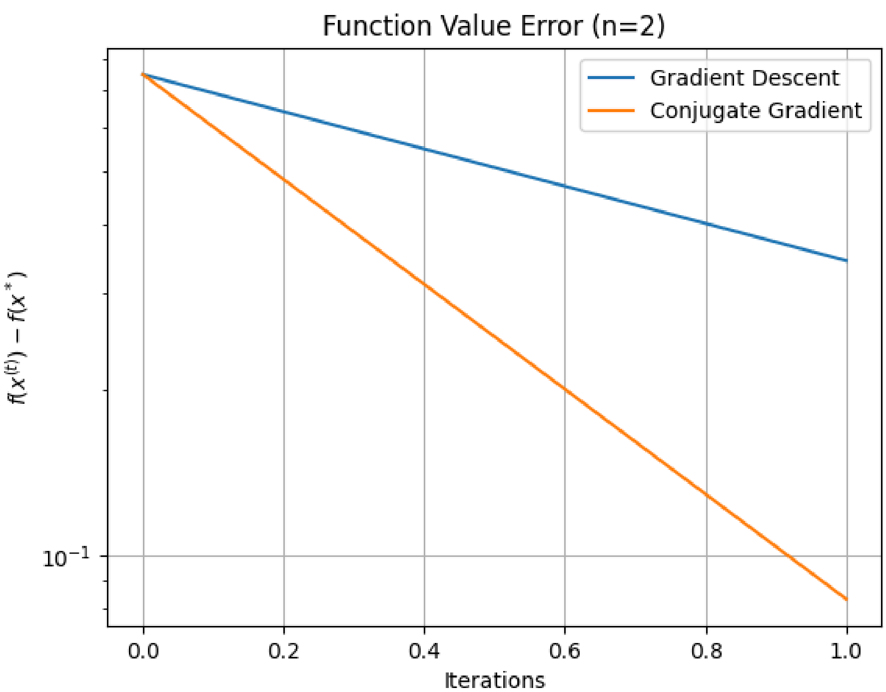
3. **Stop after T iterations.**

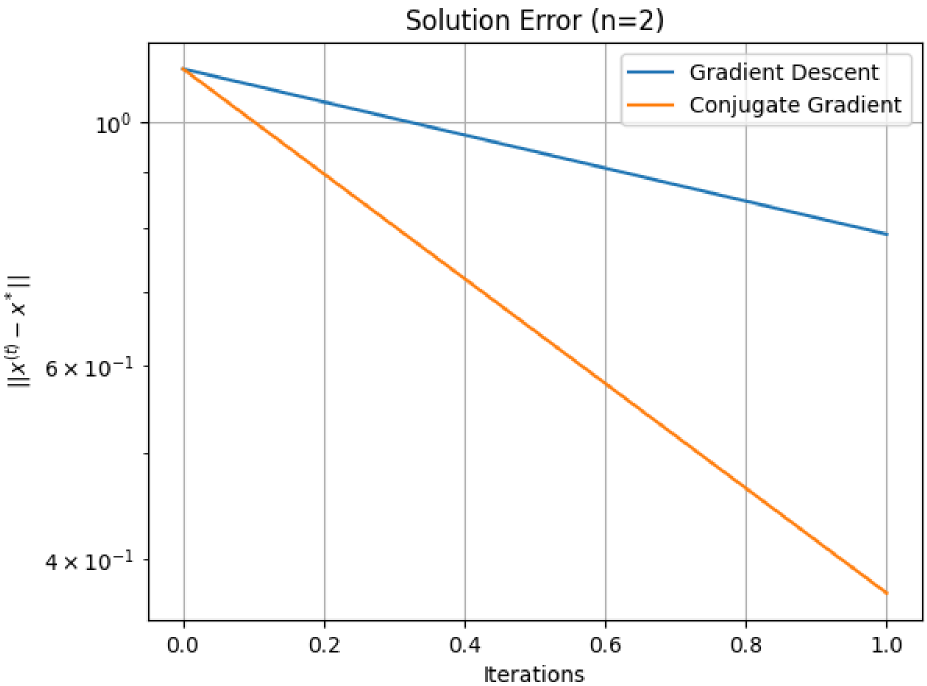
   - Both methods can stop after $T = n$ iterations or based on convergence criteria such as:
   $$\|r^{(t)}\| \leq \epsilon, \quad f(x^{(t)}) - f(x^*) \leq \epsilon,$$
   where $\epsilon > 0$ is a small tolerance.

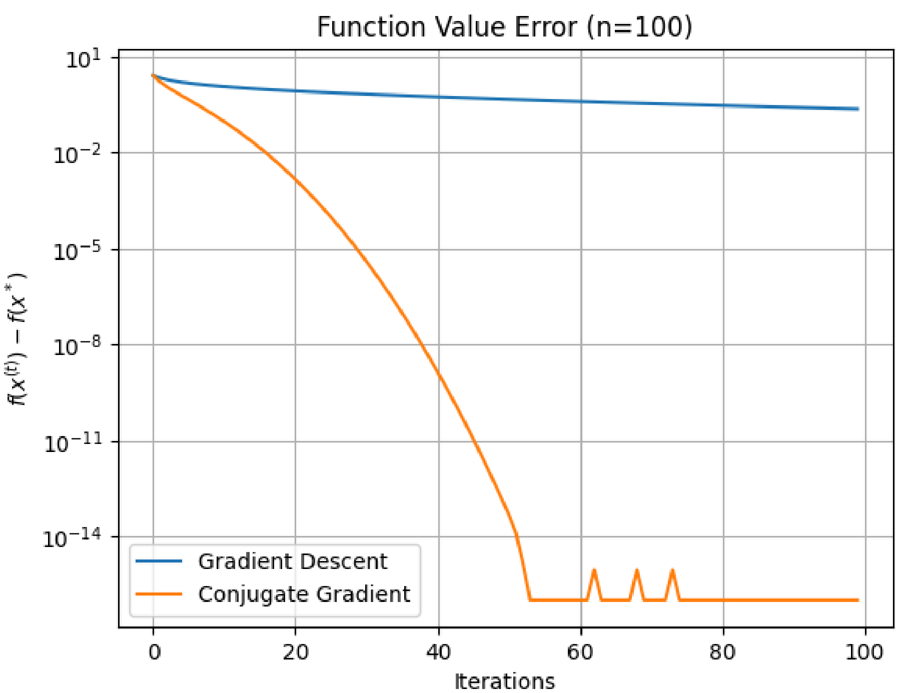
In [4]:
```
def conjugate_gradient(A, b, x0, iterations):
    x = x0.copy()
    r = b - A @ x
    p = r.copy()
    f_values = []
    x_values = []
    for _ in range(iterations):
        f_values.append(f(x, A, b))
        x_values.append(x.copy())
        Ap = A @ p
        alpha = r.T @ r / (p.T @ Ap)
        x += alpha * p
        r_new = r - alpha * Ap
        beta = (r_new.T @ r_new) / (r.T @ r)
        p = r_new + beta * p
        r = r_new
    return np.array(f_values), np.array(x_values)
```

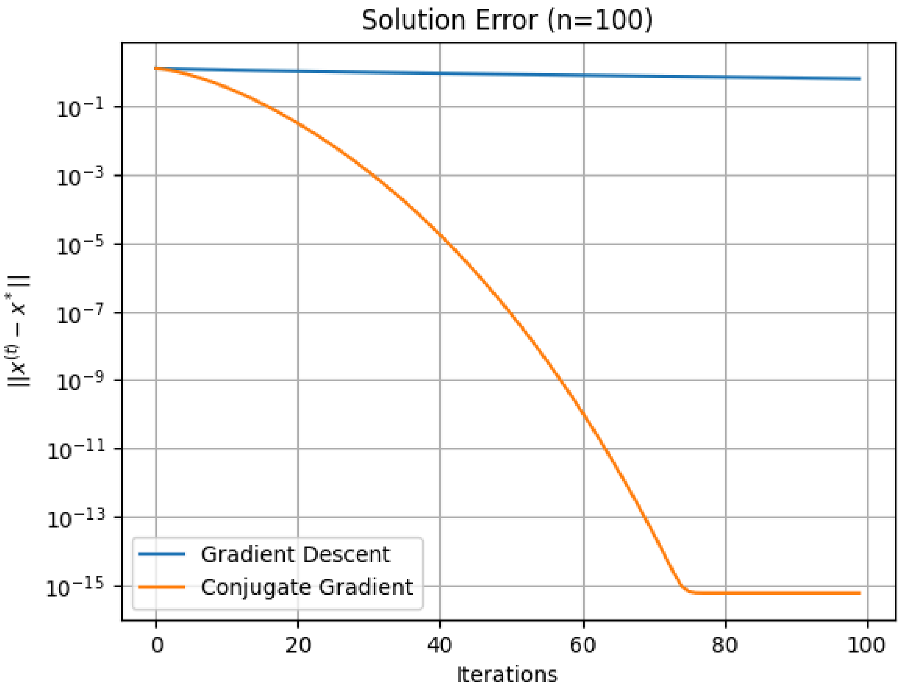
In [5]:
```
for n in n_values:
    A = np.diag(np.arange(1, n + 1))
    b = np.ones(n)

    # Optimal solution
```

```
    x_star = np.linalg.solve(A, b)

    x0 = np.zeros(n)
    iterations = n

    # Adjust the stepsize based on the largest eigenvalue
    stepsize = 1 / (2 * n)

    f_values_gd, x_values_gd = gradient_descent(A, b, x0, stepsize, iterations)
    f_values_cg, x_values_cg = conjugate_gradient(A, b, x0, iterations)

    f_star = f(x_star, A,  b)
    f_errors_gd = np.maximum(f_values_gd - f_star, 1e-16)
    f_errors_cg = np.maximum(f_values_cg - f_star, 1e-16)
    x_errors_gd = np.maximum(np.linalg.norm(x_values_gd - x_star, axis=1), 1e-16)
    x_errors_cg = np.maximum(np.linalg.norm(x_values_cg - x_star, axis=1), 1e-16)

    results[n] = {
        "f_errors_gd": f_errors_gd,
        "f_errors_cg": f_errors_cg,
        "x_errors_gd": x_errors_gd,
        "x_errors_cg": x_errors_cg,
    }

    plt.figure()
    plt.semilogy(f_errors_gd, label="Gradient Descent")
    plt.semilogy(f_errors_cg, label="Conjugate Gradient")
    plt.xlabel("Iterations")
    plt.ylabel("$f(x^{(t)}) - f(x^*)$")
    plt.title(f"Function Value Error (n={n})")
    plt.legend()
    plt.grid(True)
    plt.show()

    plt.figure()
    plt.semilogy(x_errors_gd, label="Gradient Descent")
    plt.semilogy(x_errors_cg, label="Conjugate Gradient")
    plt.xlabel("Iterations")
    plt.ylabel("$||x^{(t)} - x^*||$")
    plt.title(f"Solution Error (n={n})")
    plt.legend()
    plt.grid(True)
    plt.show()
```




Solution Error (n=2)
Gradient Descent
Conjugate Gradient
10^0
6×10^{-1}
4×10^{-1}
$||x^{(t)} - x^*||$
0.0
0.2
0.4
0.6
0.8
1.0
Iterations


Function Value Error (n=100)
10^1
10^{-2}
10^{-5}
10^{-8}
10^{-11}
10^{-14}
$f(x^{(t)}) - f(x^*)$
Gradient Descent
Conjugate Gradient
0
20
40
60
80
100
Iterations


Solution Error (n=100)
$||x^{(t)} - x^*||$
10^{-1}
10^{-3}
10^{-5}
10^{-7}
10^{-9}
10^{-11}
10^{-13}
10^{-15}
0
20
40
60
80
100
Iterations
Gradient Descent
Conjugate Gradient