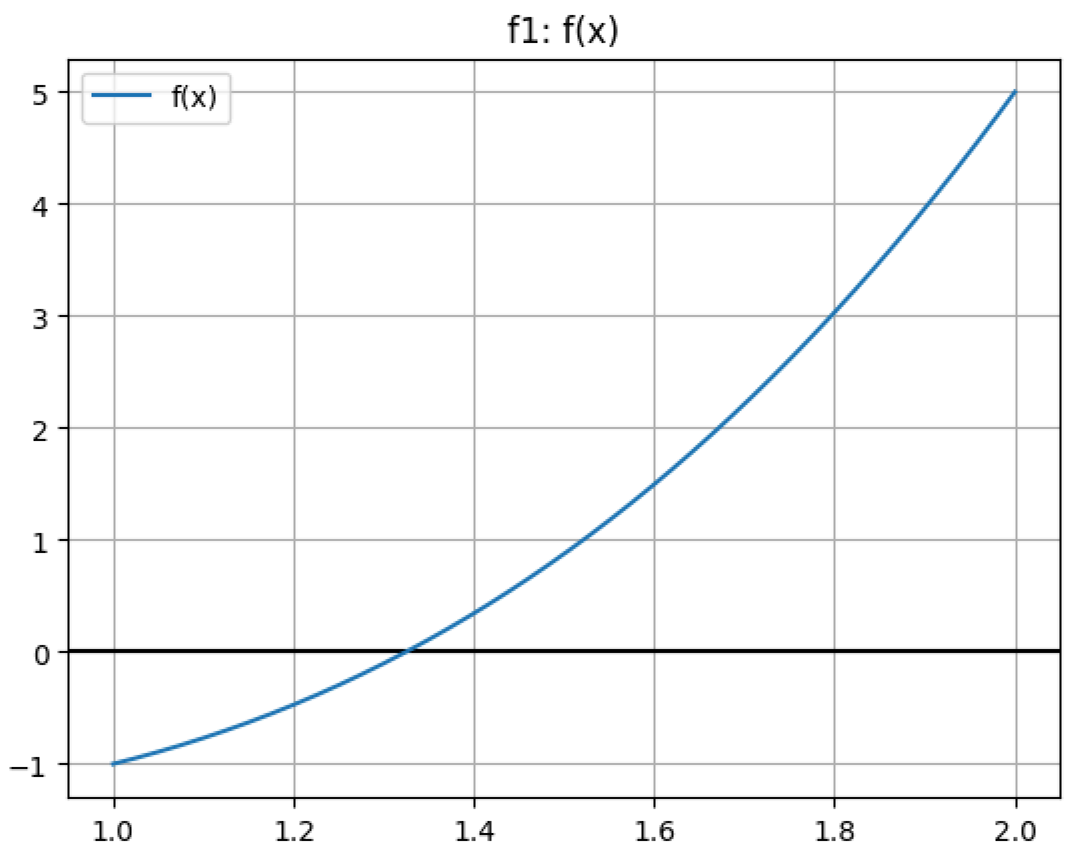
In [ ]:
```
pip install matplotlib
```

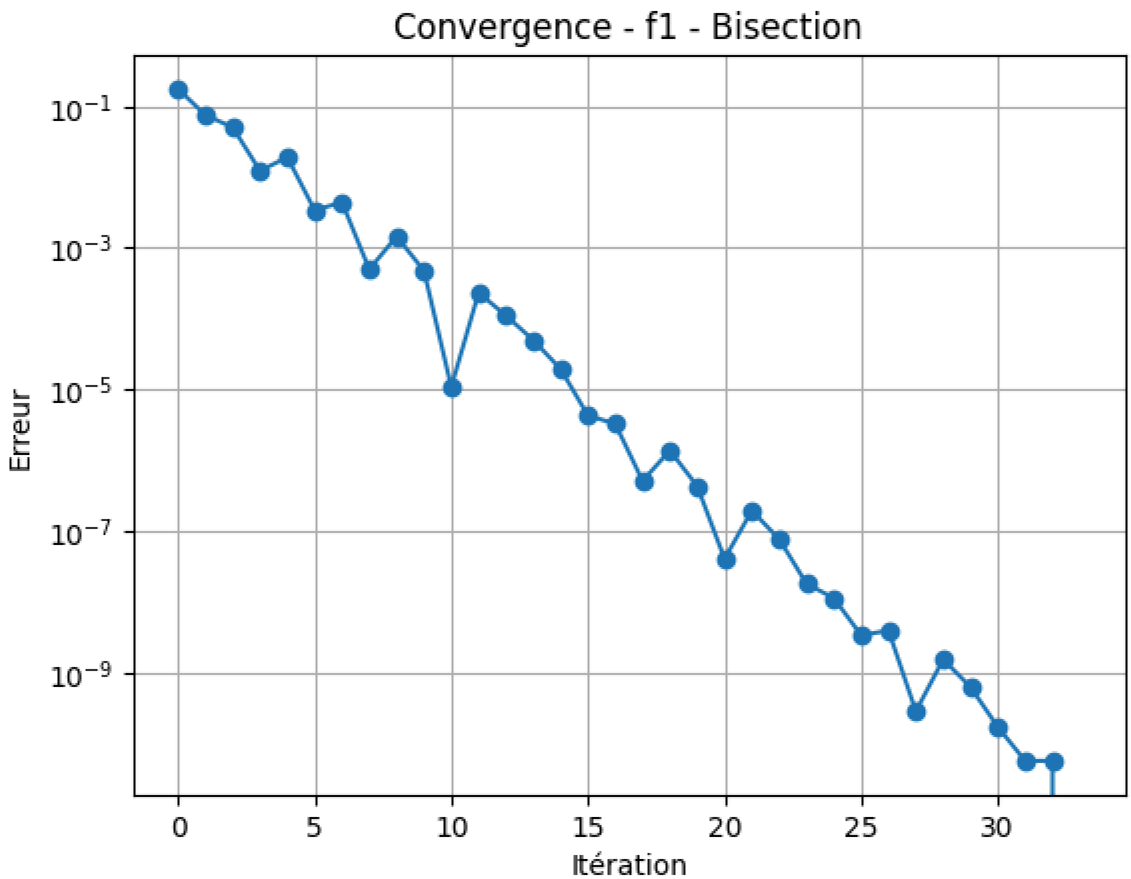
In [6]:
```
import numpy as np
import matplotlib.pyplot as plt

class EquationSolver:
    def __init__(self, f, df=None):
        self.f = f
        self.df = df
        self.history = []

    def bisection(self, a, b, tol=1e-10, max_iter=100):
        self.history = []
        fa, fb = self.f(a), self.f(b)
        if fa * fb > 0:
            raise ValueError("La fonction doit changer de signe sur [a,b].")
        for k in range(max_iter):
            m = (a + b) / 2
            fm = self.f(m)
            self.history.append(m)
            if abs(fm) < tol or (b - a) / 2 < tol:
                return m
            if fa * fm < 0:
                b, fb = m, fm
            else:
                a, fa = m, fm
        return (a + b) / 2

    def fixed_point(self, g, x0, tol=1e-10, max_iter=100):
        self.history = []
        x = x0
        for k in range(max_iter):
            self.history.append(x)
            x_new = g(x)
            if abs(x_new - x) < tol:
                return x_new
            x = x_new
        return x

    def newton_raphson(self, x0, tol=1e-10, max_iter=100):
        if self.df is None:
            raise ValueError("Il faut fournir la dérivée pour Newton-Raphson")
        self.history = []
        x = x0
        for k in range(max_iter):
            self.history.append(x)
            fx, dfx = self.f(x), self.df(x)
            if dfx == 0:
                raise ZeroDivisionError("Dérivée nulle, Newton échoue")
            x_new = x - fx / dfx
            if abs(x_new - x) < tol:
                return x_new
            x = x_new
        return x
```

```
    def plot_function(self, a, b, title=""):
        X = np.linspace(a, b, 400)
        Y = [self.f(x) for x in X]
        plt.axhline(0, color="black")
        plt.plot(X, Y, label="f(x)")
        plt.title(title)
        plt.grid()
        plt.legend()
        plt.show()

    def plot_convergence(self, true_root=None, method=""):
        if not self.history:
            return
        errors = [abs(x - true_root) for x in self.history] if true_root else None
        plt.plot(range(len(self.history)), errors if errors else self.history, mark
        plt.xlabel("Itération")
        plt.ylabel("Erreur" if true_root else "Approximation")
        plt.title(f"Convergence - {method}")
        plt.yscale("log") if true_root else None
        plt.grid()
        plt.show()


# ==============================
# Définition des fonctions
# ==============================

functions = {
    "f1": {
        "f": lambda x: x**3 - x - 1,
        "df": lambda x: 3*x**2 - 1,
        "interval": (1, 2),
        "g": lambda x: (x + 1)**(1/3)   # reformulation point fixe
    },
    "f2": {
        "f": lambda x: np.exp(x) - 2*x - 1,
        "df": lambda x: np.exp(x) - 2,
        "interval": (0, 2),
        "g": lambda x: (np.exp(x) - 1)/2
    },
    "f3": {
        "f": lambda x: np.cos(x) - x,
        "df": lambda x: -np.sin(x) - 1,
        "interval": (0, 1),
        "g": lambda x: np.cos(x)
    }
}

# ==============================
# Application
# ==============================

for name, data in functions.items():
    print(f"\n=== {name} ===")
    solver = EquationSolver(data["f"], data["df"])
```

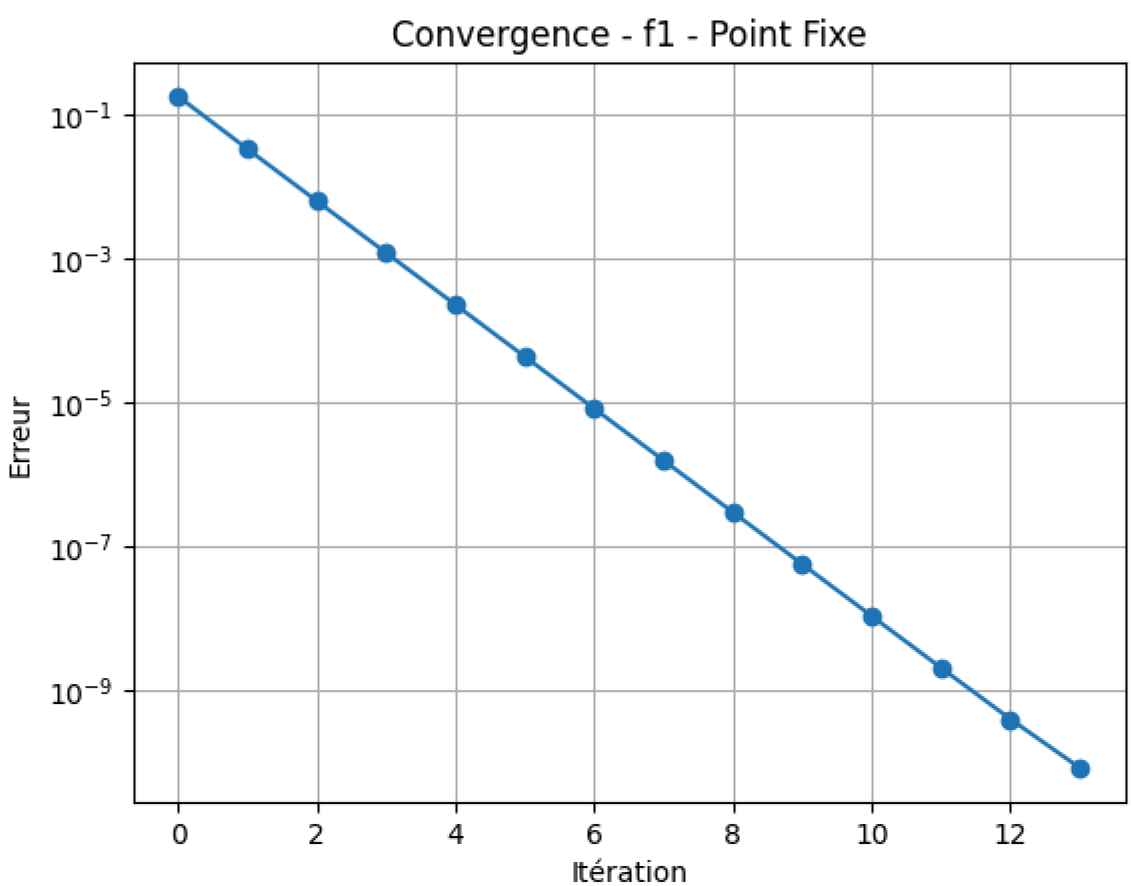
```
    a, b = data["interval"]

    # Tracer fonction
    solver.plot_function(a, b, title=f"{name}: f(x)")

    # Bisection
    root_bisec = solver.bisection(a, b)
    solver.plot_convergence(true_root=root_bisec, method=f"{name} - Bisection")
    print("Bisection root =", root_bisec)

    # Point fixe
    root_fixed = solver.fixed_point(data["g"], (a+b)/2)
    solver.plot_convergence(true_root=root_bisec, method=f"{name} - Point Fixe")
    print("Fixed point root =", root_fixed)

    # Newton-Raphson (différents points de départ)
    for x0 in [a, (a+b)/2, b]:
        solver = EquationSolver(data["f"], data["df"])
        try:
            root_newton = solver.newton_raphson(x0)
            solver.plot_convergence(true_root=root_bisec, method=f"{name} - Newton
            print(f"Newton-Raphson root (x0={x0}) =", root_newton)
        except Exception as e:
            print(f"Newton échoue pour x0={x0}: {e}")
```

```
=== f1 ===
```

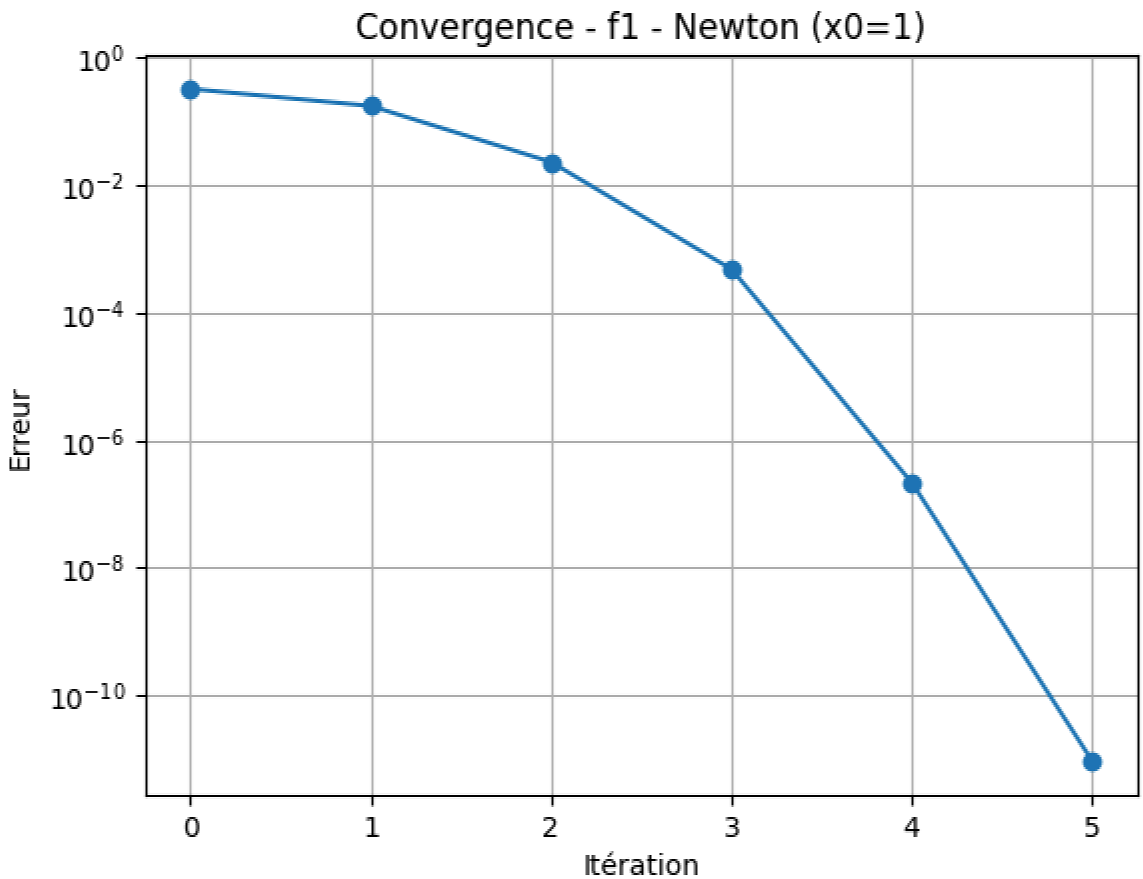




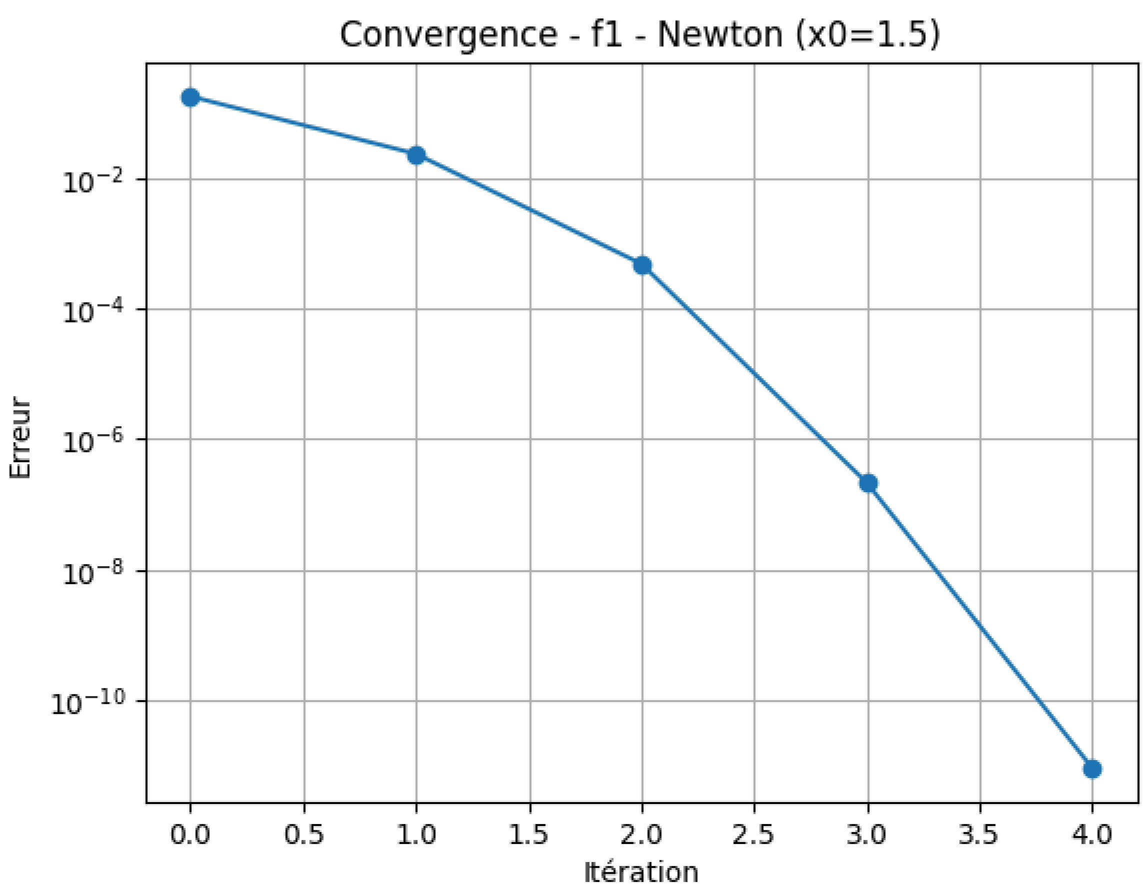

```
Bisection root = 1.324717957235407
```



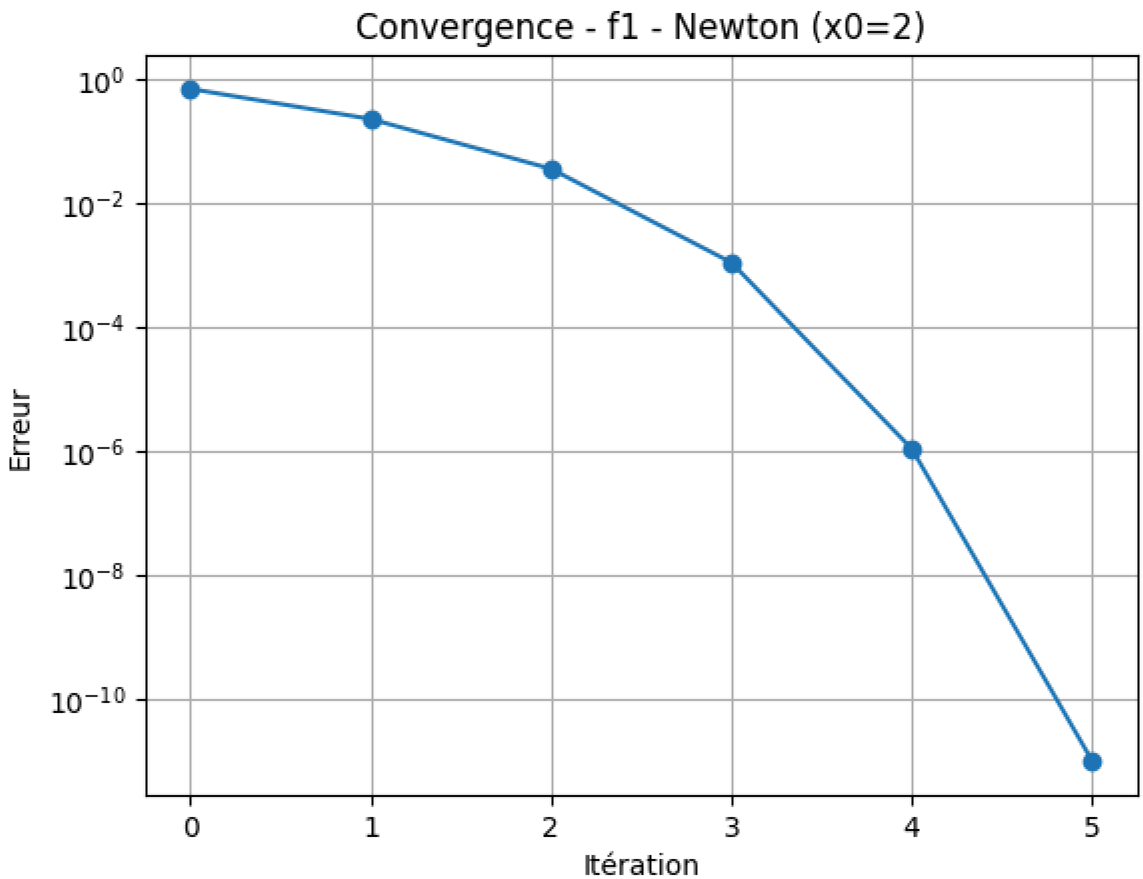

```
Fixed point root = 1.3247179572582821
```



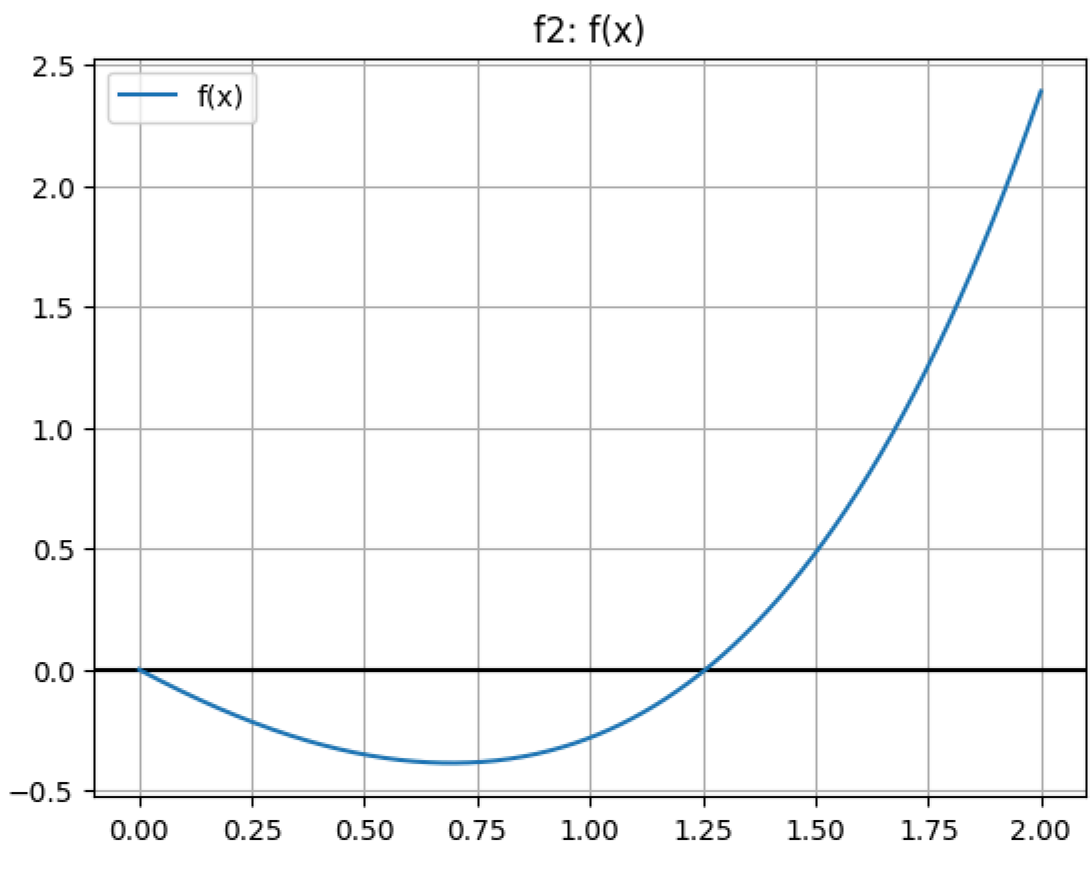

```
Newton-Raphson root (x0=1) = 1.324717957244746
```

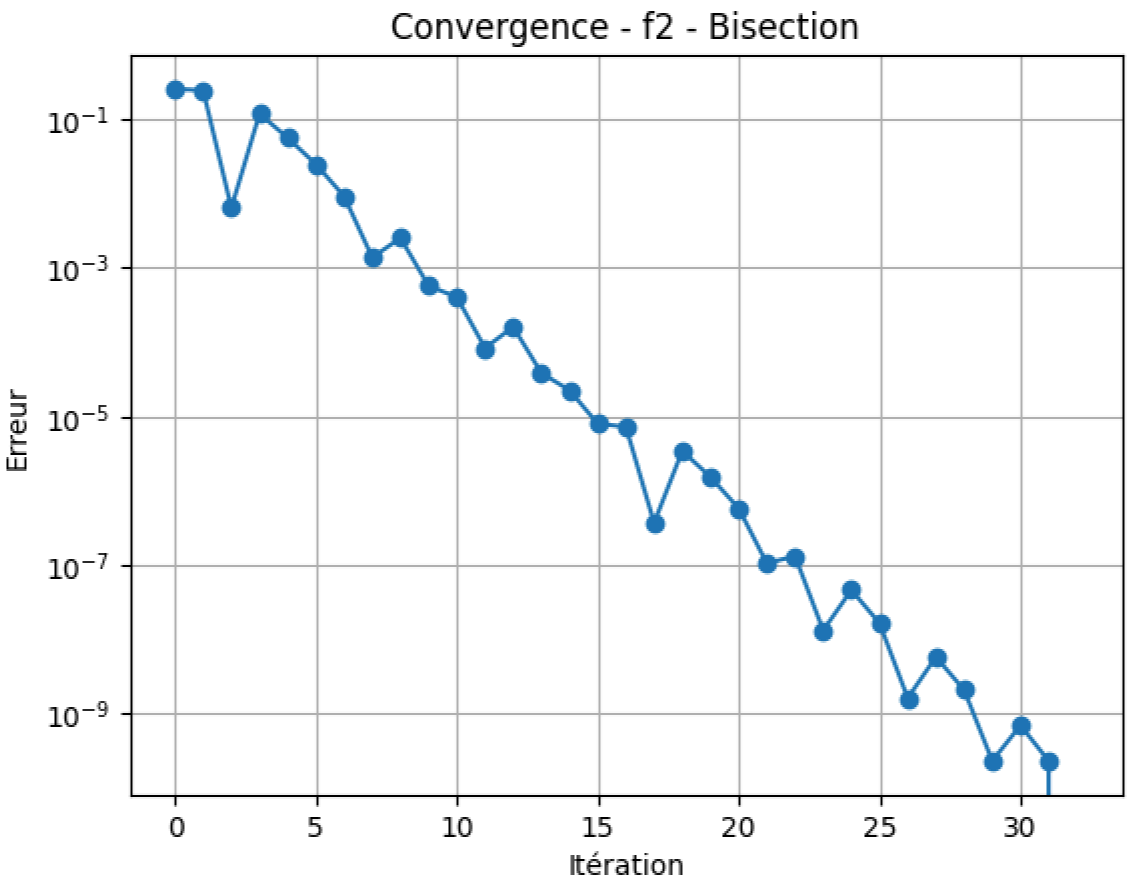



```
Newton-Raphson root (x0=1.5) = 1.324717957244746
```

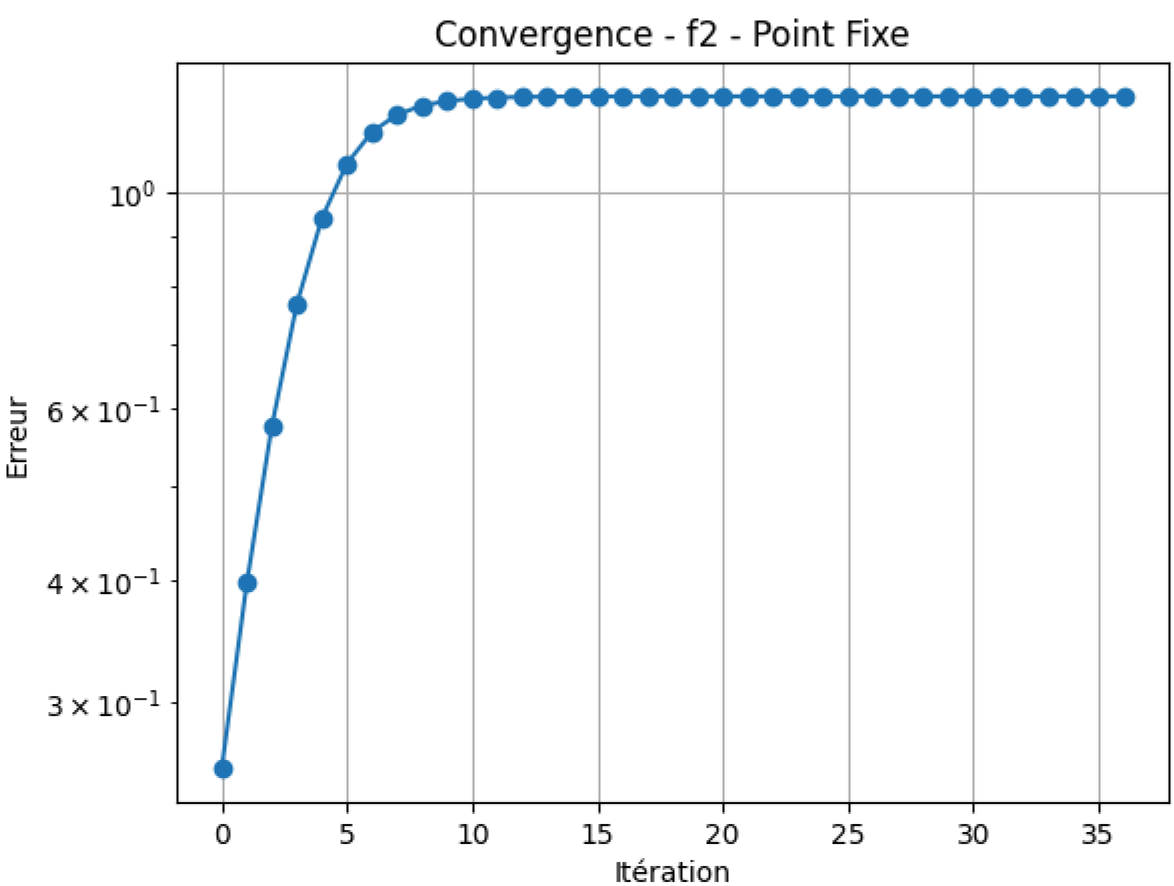



```
Newton-Raphson root (x0=2) = 1.324717957244746

=== f2 ===
```

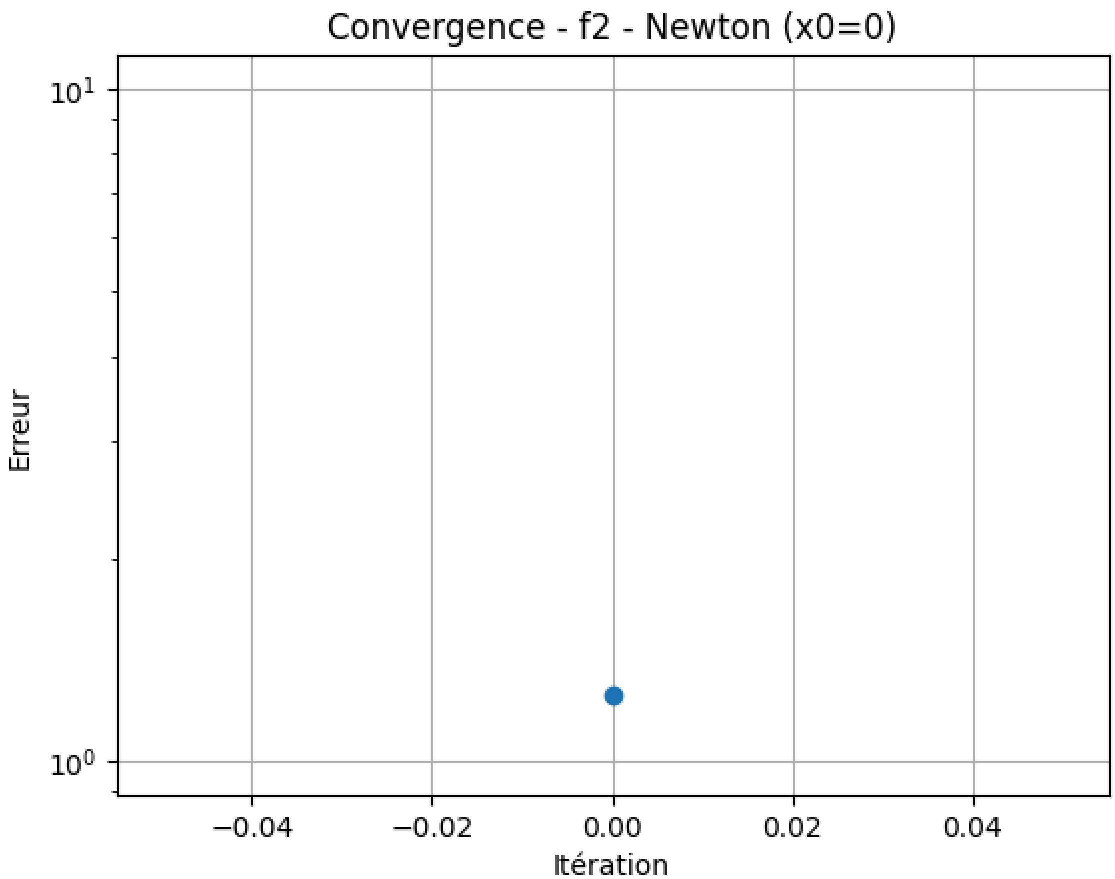




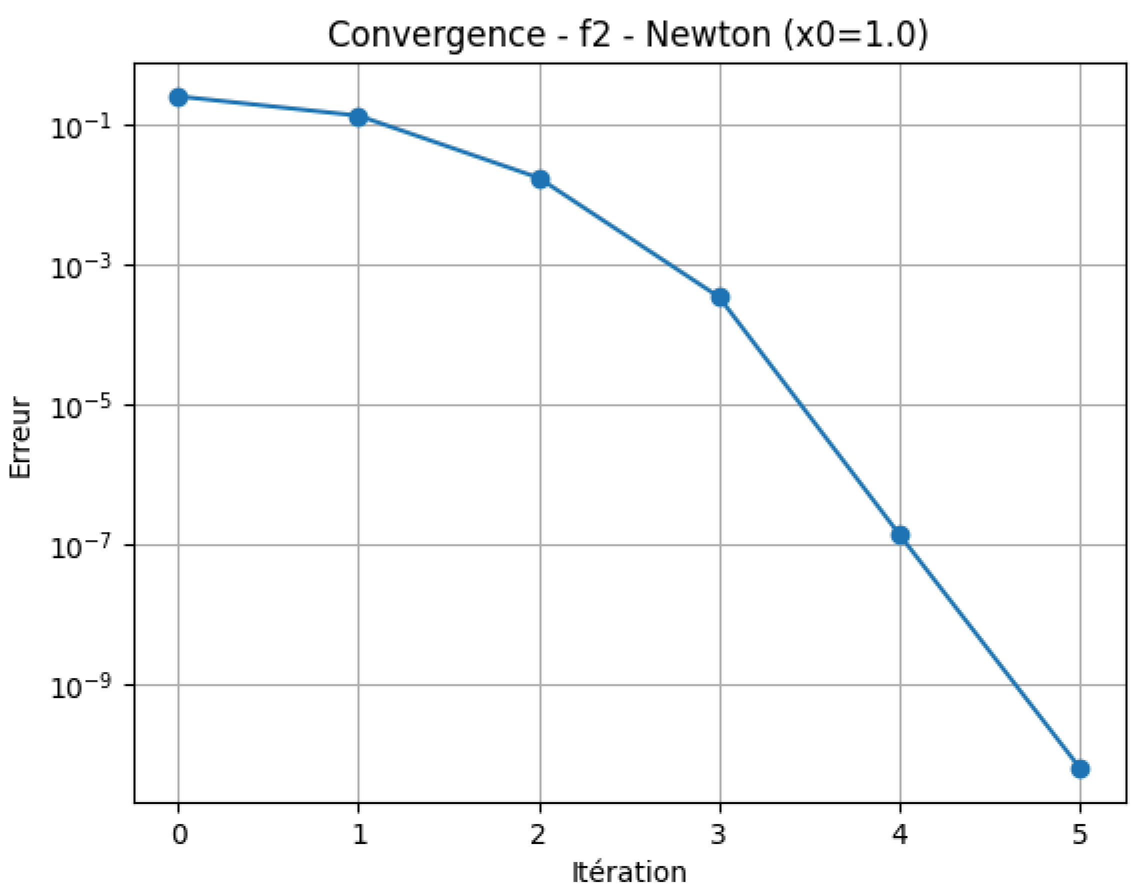

```
Bisection root = 1.2564312086906284
```



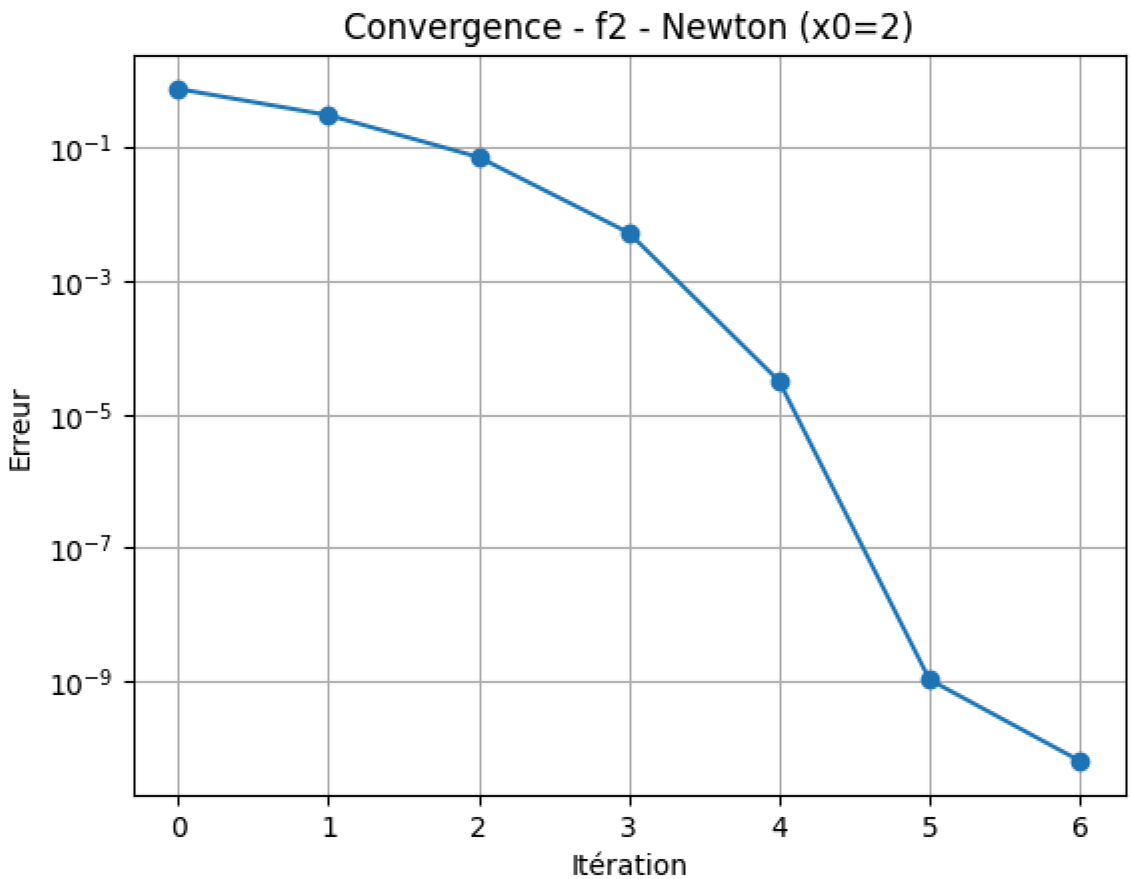

```
Fixed point root = 5.243483425232398e-11
```



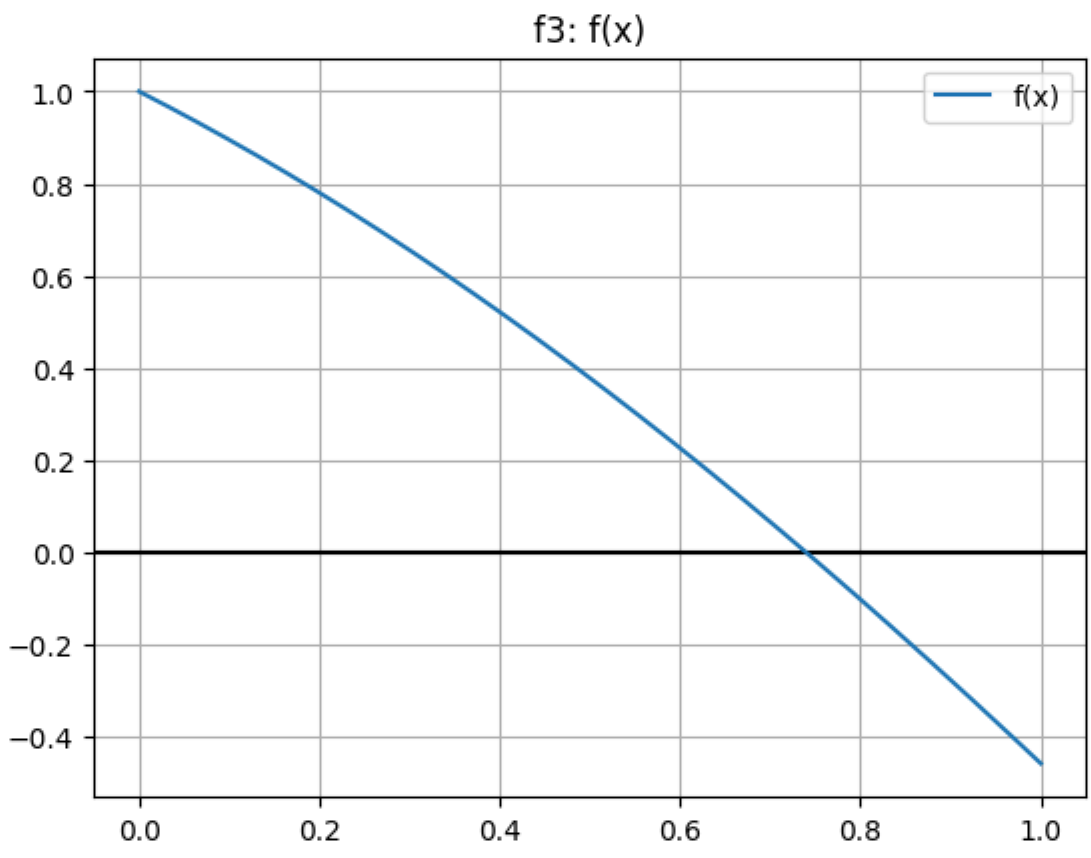

```
Newton-Raphson root (x0=0) = 0.0
```

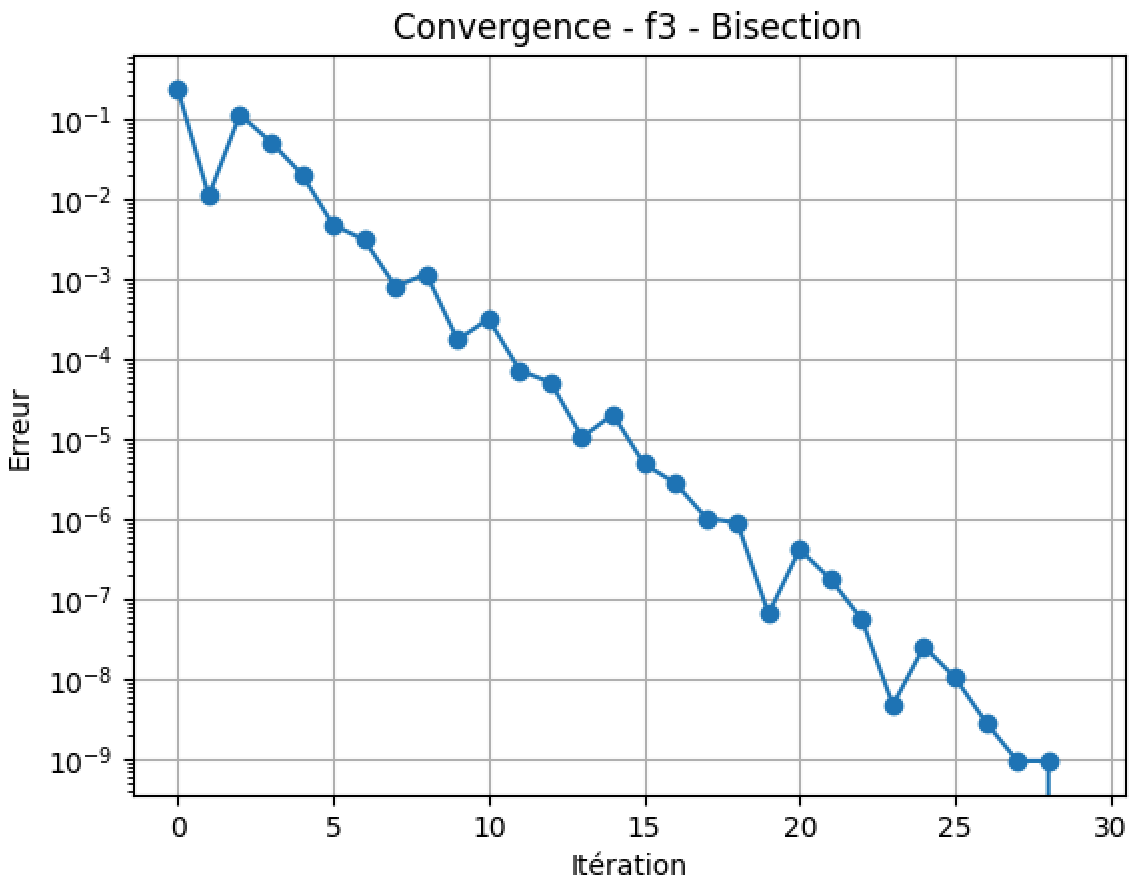



```
Newton-Raphson root (x0=1.0) = 1.2564312086261695
```

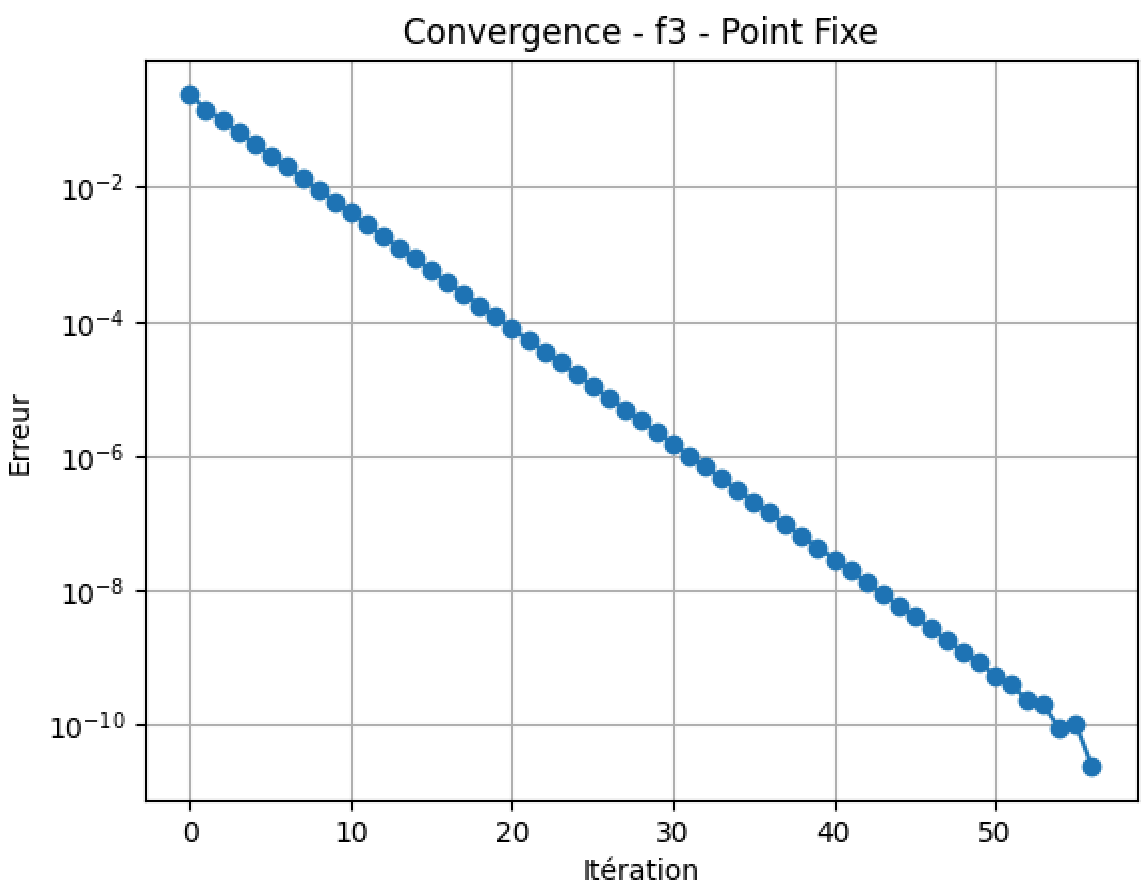



```
Newton-Raphson root (x0=2) = 1.2564312086261697

=== f3 ===
```

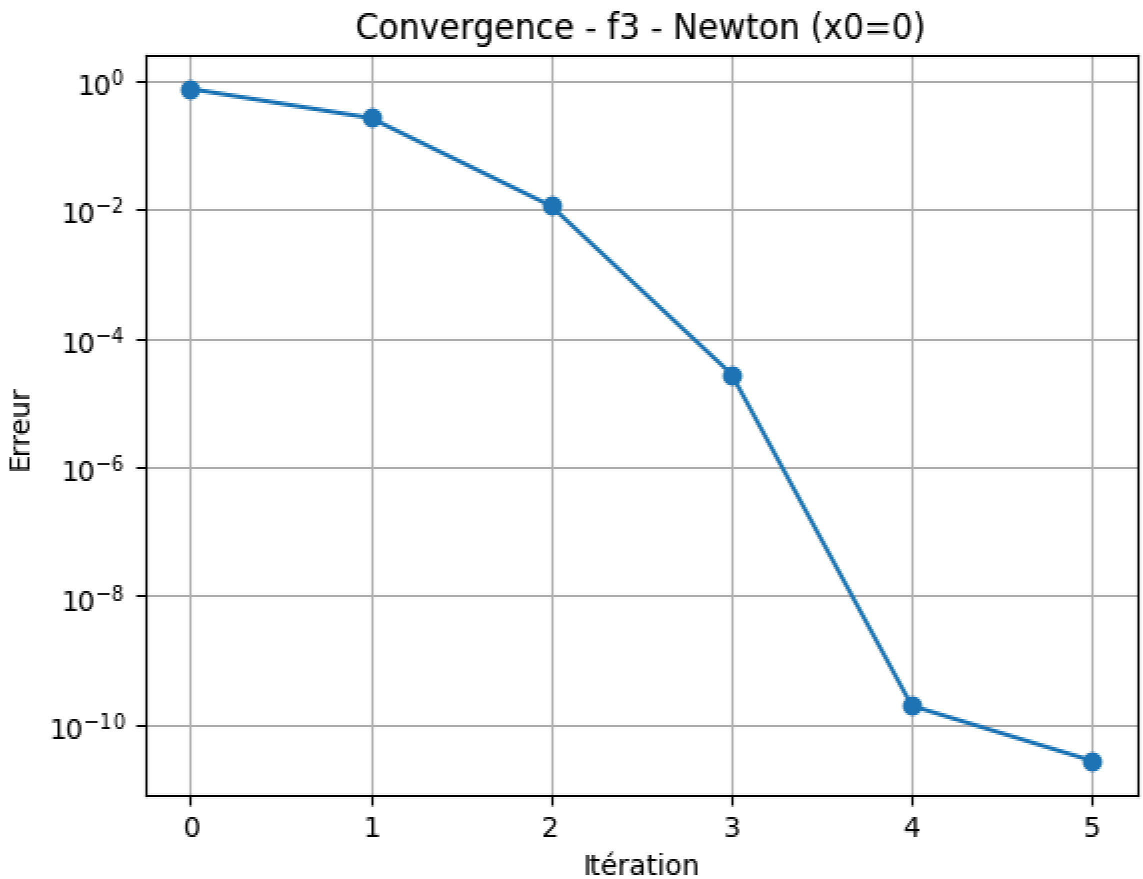




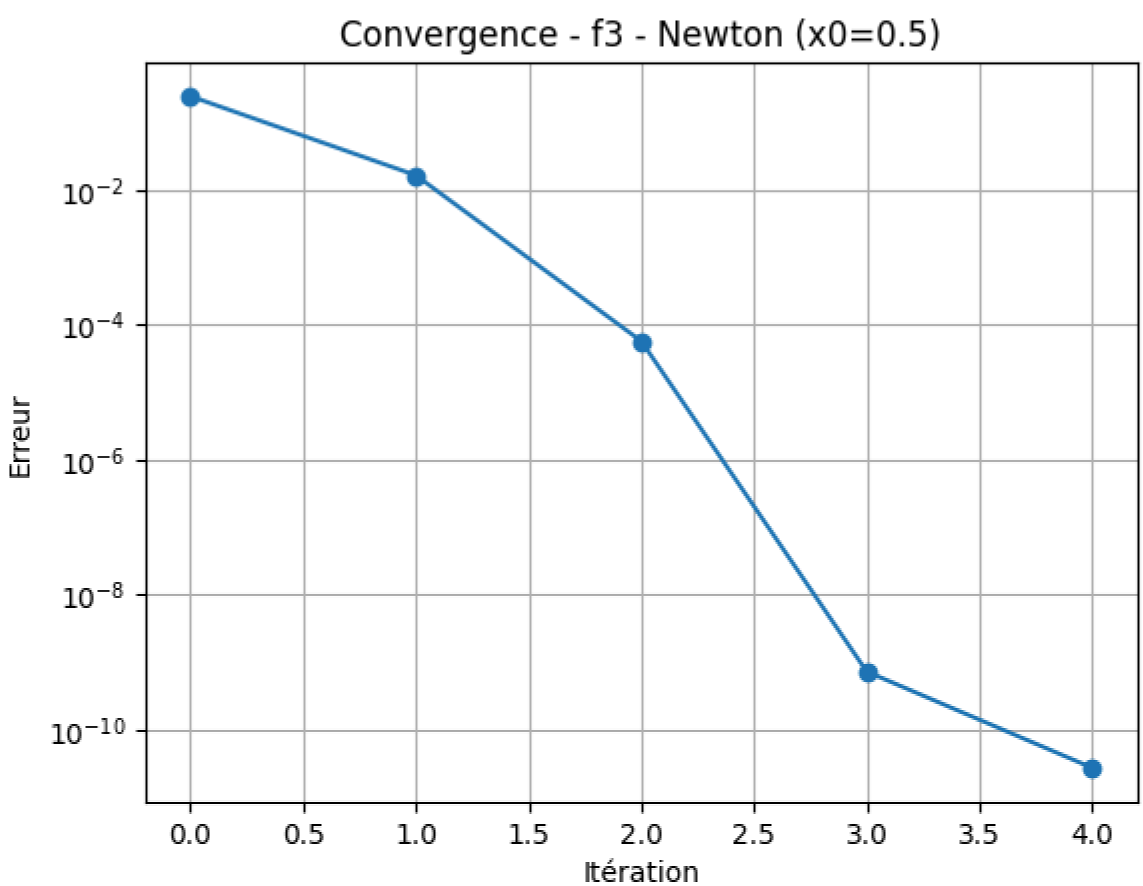

```
Bisection root = 0.7390851331874728
```



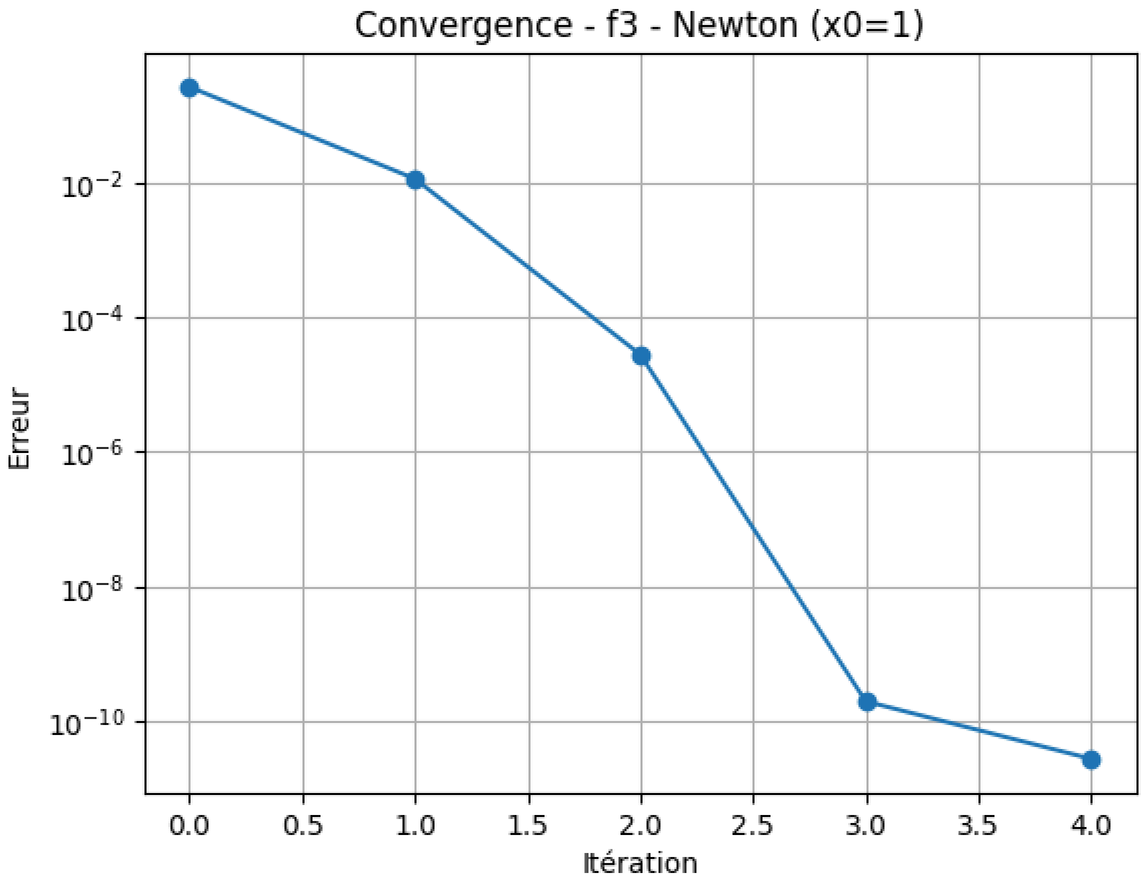

```
Fixed point root = 0.7390851332502528
```




```
Newton-Raphson root (x0=0) = 0.7390851332151607
```




```
Newton-Raphson root (x0=0.5) = 0.7390851332151607
```




```
Newton-Raphson root (x0=1) = 0.7390851332151607
```