Problem 1: Newton Raphson

Problem Statement

a

Using Newton-Raphson, solve the following system of equations in $\mathbb{R}^2$:

$$f_1(x) \coloneqq 4x_2^2 + 4x_2 + 52x_1 - 19 = 0$$

$$f_2(x) \coloneqq 169x_1^2 + 3x_2^2 + 111x_1 - 10x_2 - 10 = 0$$

starting from $x^0 = (5, 0)$.

Report x^1 and x^2. Terminate your iteration when the 2-norm of f is less than 10^{-10}. Report if it does not converge in 100 iterations. If it converges, report x^* and the iteration count.

Plot the error magnitudes $\|x^k - x^*\|$ as a function of k on a semilog plot.

b

Repeat (a), but with forward-difference approximation, using $h = 0.5$.

b

Repeat (a), but with center-difference approximation, using $h = 0.5$.

d

Repeat (a), but using Jacobian surrogates defined by Broyden's method. Clearly state how you start the sequence of Jacobian surrogates and how you choose x^1.

e

Vary the starting point appropriately and discuss qualitatively how the iterative process in parts a-d behave. If you have to choose between methods b-d, which one would you choose? Will this change if evaluation of f is much more computationally expensive?

Solution

a-e

See code at end of document for implementation, or my repo here:

https://github.com/eric-silk/ECE530

Values requested (x^1, x^2, iteration count) are reported in the subplot titles. Specific to part (e), in general it appears that the symbolic and center difference approximation methods tend to behave *very* similarly, and typically converge in far fewer iterations than the forward-difference or Broyden's method; however, these methods also appear to fail in cases where the forward-difference and Broyden's method do not. This may simply be an artifact of the relatively low limit on iterations, but it is worth noting.

From this, for an arbitrary function, I'd probably pick Broyden's method (or some other quasi-Newton method), especially as the computational cost of each function evaluation increases. It requires only one evaluation of f per iterate (which can quickly become the most significant cost), and empirically demonstrates good convergence properties in this limited set of experiments.

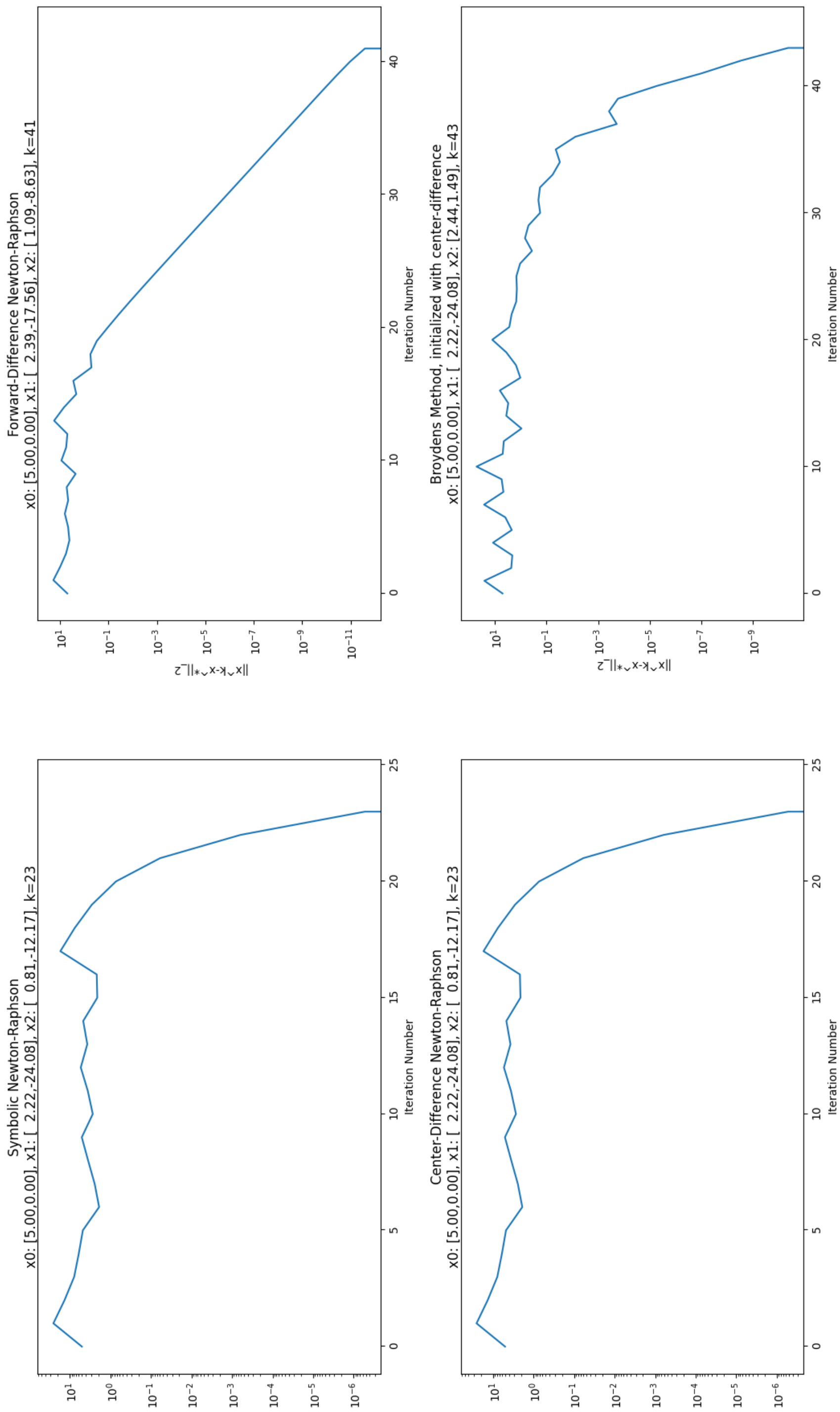

Figure 1: The four methods, started from $(5, 0)$

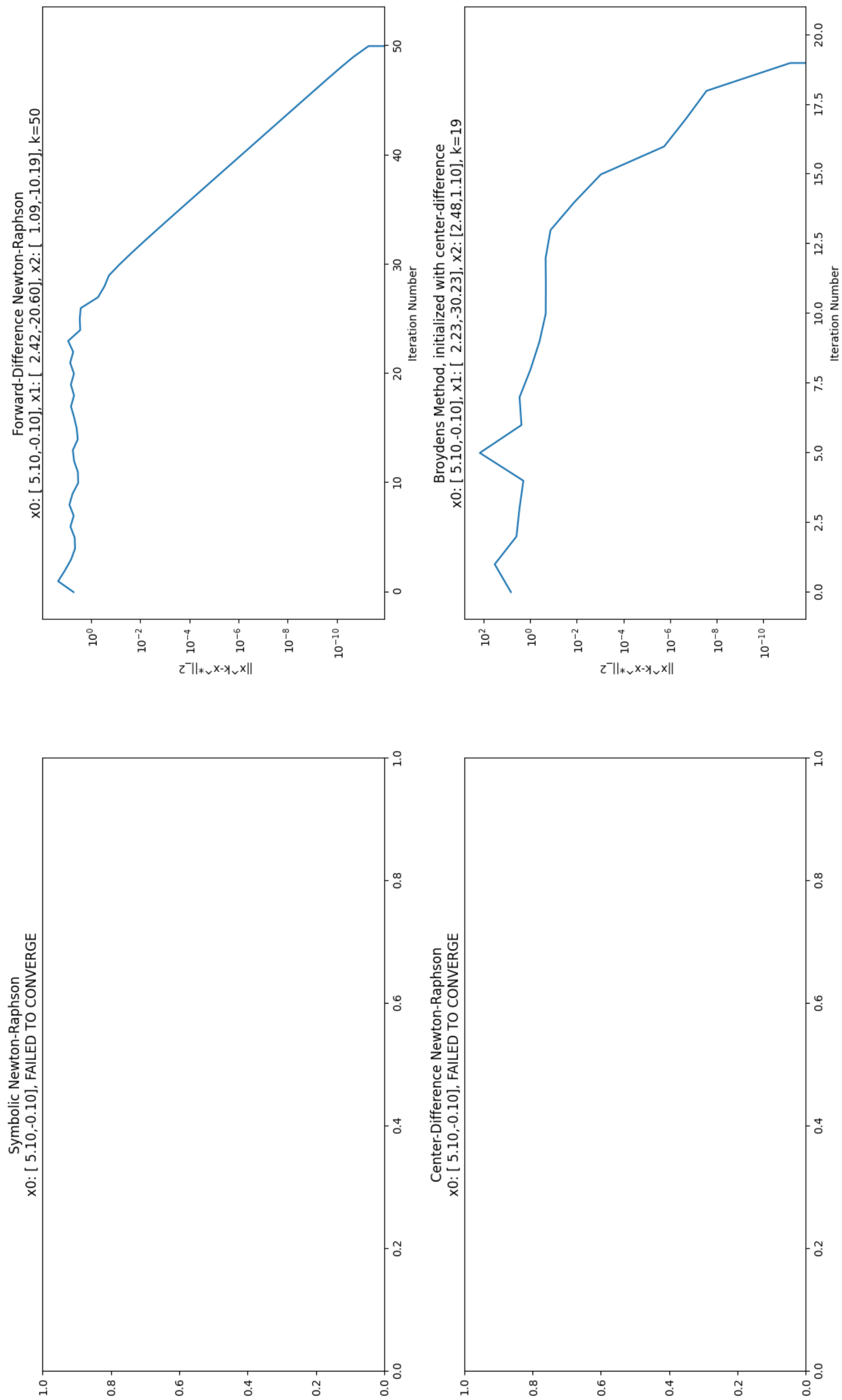

Figure 2: The four methods, started from $(5.1, -0.1)$

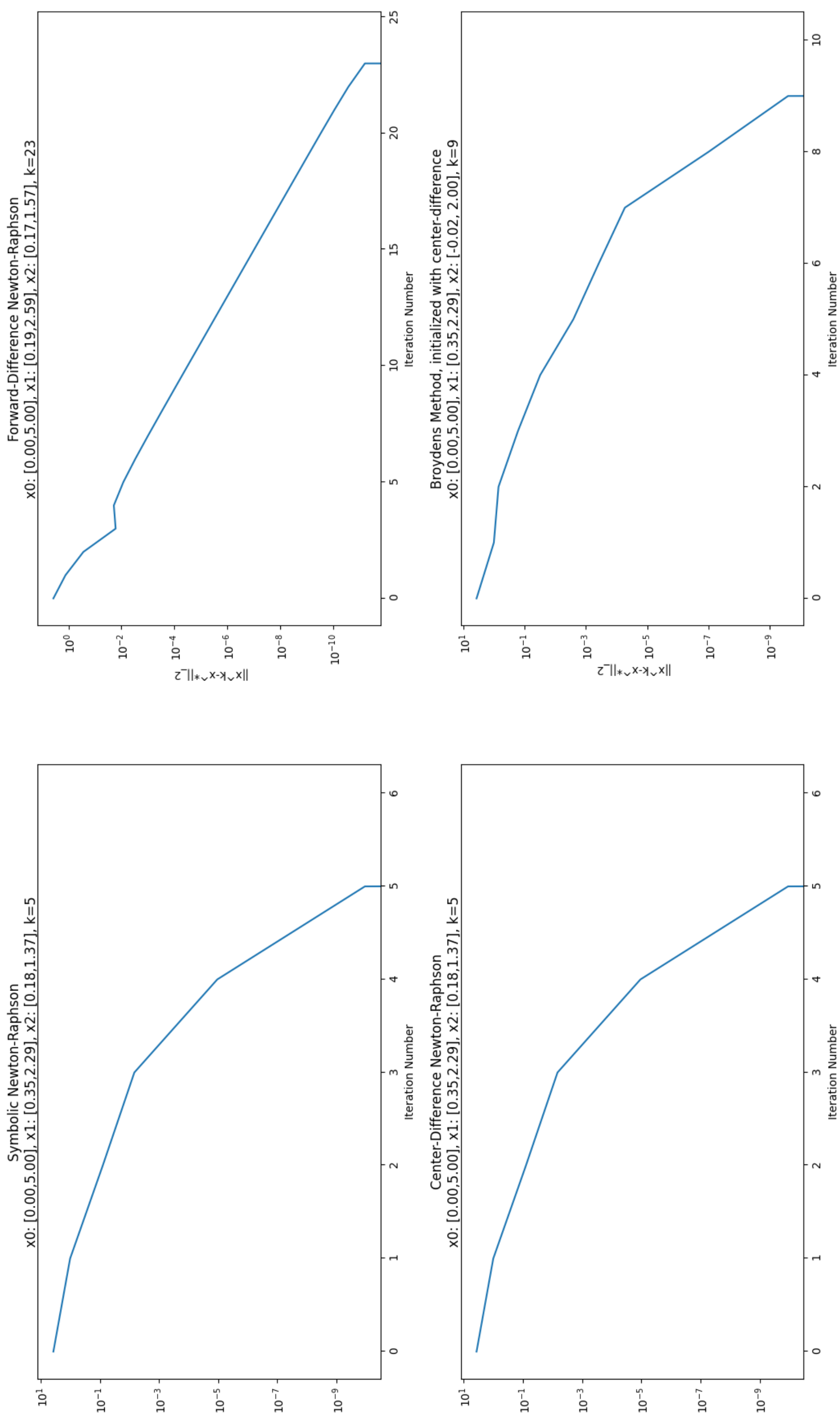

Figure 3: The four methods, started from $(0, 5)$

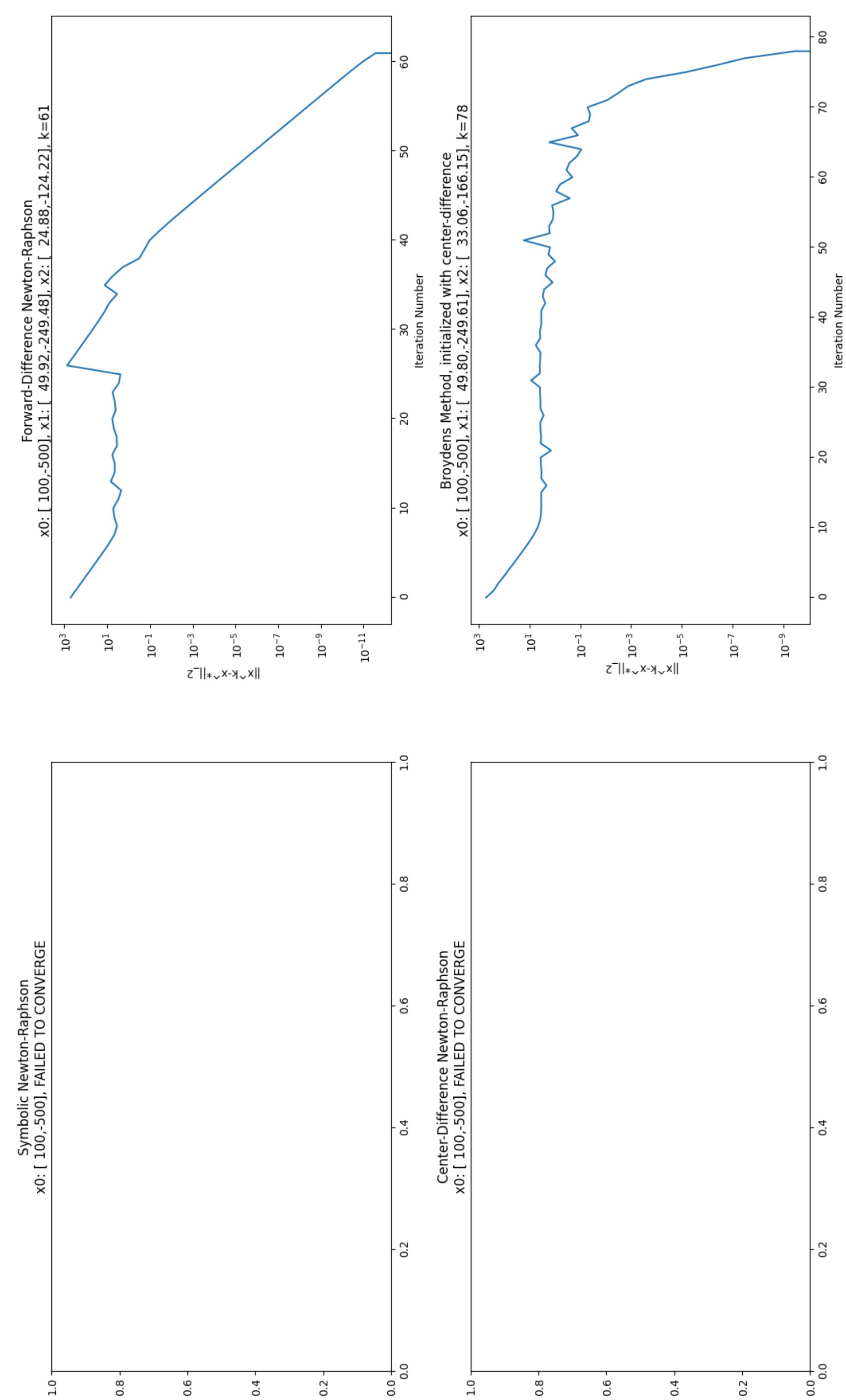

Figure 4: The four methods, started from $(100, -500)$

Problem 2: Understanding Numerical Differentiation

Problem Statement

We have empirically seen in class that the approximation quality of center difference approximation is far superior to that of forward and backward difference ones, when performing numerical differentiation. In this problem, we gain analytical insight into this behavior.

Suppose $f : \mathbb{R} \to \mathbb{R}$ is at least C^3 over $[a, b]$. The following Taylor's series expansion of f might be useful:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(\zeta)$$

$\forall x, x+h \in (a, b)$ where ζ lies on the line segment joining x and $x+h$.

a

Show that

$$\left|\frac{f(x+h)-f(x)}{h} - f'(x)\right| \leq M|h| \forall x \in (a,b), M > 0$$

b

Show that

$$\left|\frac{f(x+h)-f(x-h)}{2h} - f'(x)\right| \leq N|h|^2 \forall x \in (a,b), N > 0$$

c

Qualitatively discuss what the results in (a) and (b) mean.

Solution

a

Re-arrange the provided Taylor series expansion and take the absolute value of both sides:

$$\left|\frac{f(x+h)-f(x)}{h} - f'(x)\right| = \left|h(\frac{1}{2}f''(x) + \frac{1}{6}hf'''(\zeta))\right| = |h|\left|\frac{1}{2}f''(x) + \frac{1}{6}hf'''(\zeta)\right|$$

Now we just need to bound the rightmost absolute value. From the class notes ("Solving nonlinear equations"), note that because ζ lies on the line segment connecting a and b, $\zeta \in [a, b]$. Because the function is C^3 over $[a, b]$, there exists a scalar M_1 and M_2 that bounds f'' and f''' (repsectively) from above. So:

$$\left|\frac{1}{2}f''(x) + \frac{1}{6}hf'''(\zeta)\right| \leq \left|\frac{1}{2}M_1 + \frac{1}{6}hM_2\right|$$

Finally, because h is some fixed value, the whole quantity is bounded, and we can say:

$$\left|\frac{1}{2}M_1 + \frac{1}{6}hM_2\right| \leq M$$

$$\therefore \left|\frac{f(x+h)-f(x)}{h} - f'(x)\right| \leq M|h|$$

$\square$

b

We can modify the expansion to consider a negative h, or equivalently the subtraction of a positive h:

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(\zeta)$$

Then, subtract the two series to find:

$$\begin{aligned} f(x+h) - f(x-h) &= f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(\zeta) - f(x) + hf'(x) - \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(\zeta) \\ &= 2hf'(x) + \frac{2h^3}{6}f'''(\zeta) \end{aligned}$$

Rearranging:

$$\frac{f(x+h) - f(x-h)}{2h} - f'(x) = \frac{h^2}{6}f'''(\zeta)$$

$$\left|\frac{f(x+h) - f(x-h)}{2h} - f'(x)\right| = \left|\frac{h^2}{6}f'''(\zeta)\right| = \left|h^2\right| \left|\frac{1}{6}f'''(\zeta)\right|$$

By the same logic as in (2.a), we know that $f'''(\zeta)$ is upper bounded, so we can say:

$$\frac{1}{6}f'''(\zeta) \leq N$$

Thus:

$$\left|\frac{f(x+h) - f(x-h)}{2h} - f'(x)\right| \leq \left|h^2\right| N$$

□

c

As the proofs show, there's a cancellation effect that occurs for some of the more significant error terms, resulting in the far better approximation (at least for "small" h). There's an "averaging" effect that reduces the error.

Expanding on the numerical example provided in class, it's worth looking at it as a form of a statistical filter[1]. Some "signal" (the error of the approximation) is oversampled at a $2x$ rate and then is passed through an averaging filter. Assume that:

1. The samples are uncorrelated
2. The distribution of each sample is *normal*

which seems justified based on the numerical results presented in class. Then, letting X_i denote the distribution of the samples (where $i = 1, 2$), and $\overline{X}$ denote the average (i.e. the output of the filter), we can say:

$$\text{Var}(\overline{X}) = \text{Var}\left(\frac{1}{N}\sum_{i=1}^{N} X_i\right) = \text{Var}\left(\frac{1}{N}\sum_{i=1}^{N} X_i\right) = \frac{1}{N^2}\text{Var}\left(\sum_{i=1}^{N} X_i\right) = \frac{1}{N^2}\sum_{i=1}^{N}\text{Var}\left(X_i\right)$$

If the variances are the same ($X_1 = X_2 = X$), then we get:

$$\text{Var}(\overline{X}) = \frac{1}{2}\text{Var}(X)$$

Now, this doesn't seem to imply quite aggressive reduction in variance as demonstrated by the numerical example given in class, but it's at least a way to start thinking about how this works.

[1]Note: I'm not exactly a wizard at stats, take this with a grain of salt...

Code Listing

```python
import time

import numpy as np
import matplotlib.pyplot as plt

h = 0.5


def f(x: np.ndarray) -> np.ndarray:
    x1, x2 = x[0], x[1]
    fx1 = 4 * x2**2 + 4 * x2 + 52 * x1 - 19
    fx2 = 169 * x1**2 + 3 * x2**2 + 111 * x1 - 10 * x2 - 10

    return np.array([fx1, fx2])


def Jf(x: np.ndarray) -> np.ndarray:
    x1, x2 = x[0], x[1]
    ret = np.array([[52.0, 8.0 * x2 + 4.0], [338.0 * x1 + 111.0, 6.0 * x2 - 10.0]])

    return ret


def symbolic_newton(x):
    return x - (np.linalg.pinv(Jf(x)) @ f(x))


def forward_difference_newton(x):
    J = np.zeros((2, 2))
    J[:, 0] = (f(x + np.array([h, 0])) - f(x)) / h
    J[:, 1] = (f(x + np.array([0, h])) - f(x)) / h

    return x - (np.linalg.pinv(J) @ f(x))


def center_difference_newton(x):
    J = np.zeros((2, 2))
    J[:, 0] = (f(x + np.array([h, 0])) - f(x - np.array([h, 0]))) / (2 * h)
    J[:, 1] = (f(x + np.array([0, h])) - f(x - np.array([0, h]))) / (2 * h)

    return x - (np.linalg.pinv(J) @ f(x))


class BroydensMethod:
    def __init__(self, J0):
        self.J = J0

    def __call__(self, x):
        fx = f(x)
        dx = -np.linalg.solve(self.J, fx)
        x_i = x + dx
        dfx = f(x_i) - fx
        self.J += np.outer((dfx - (self.J @ dx)), dx.T) / (
            np.linalg.norm(dx, ord=2) ** 2
        )

        return x_i


def solve_and_plot(
    desc: str,
    iterator,
    x0: np.ndarray,
    fig: plt.Figure,
    ax: plt.Axes,
    epsilon: float = 1e-10,
    iter: int = 100,
) -> None:
    _ = fig
    x = x0
    xs = []
    xs.append(x0)
    t0 = time.time()
    tf = np.inf
    converged = False
    k = 0
    for i in range(iter):
        try:
            x = iterator(x)
            eps = np.linalg.norm(f(x), ord=2)
            xs.append(x)
            if eps < epsilon:
                tf = time.time() - t0
                print(f"Converged at iteration {i+1}")
                converged = True
                k = i
                break
        except Exception as e:
```

```python
            print(f"Exception occurred at iterate {i}")
            k = -1
            raise e
    else:
        print(f"Failed to converge in {iter} iterations!")

    xs = np.array(xs)
    errors = xs - xs[-1]
    errors = np.array([np.linalg.norm(i, ord=2) for i in errors])
    convert = lambda x: np.array2string(
        x, precision=2, separator=",", floatmode="fixed"
    )
    if converged:
        ax.semilogy(errors)
        ax.set_xlabel("Iteration Number")
        ax.set_ylabel("||x^k-x^*||_2")
        ax.set_title(
            f"{desc}\nx0: {convert(x0)}, x1: {convert(xs[1])}, x2: {convert(xs[2])}, k={k}",
            pad=-20,
        )
    else:
        ax.set_title(f"{desc}\nx0: {convert(x0)}, FAILED TO CONVERGE")


if __name__ == "__main__":
    x0s = [
        np.array([5.0, 0.0]),
        np.array([5.1, -0.1]),
        np.array([0.0, 5.0]),
        np.array([100, -500]),
    ]

    for x0 in x0s:
        fig, axes = plt.subplots(2, 2)
        fig.tight_layout()
        solve_and_plot("Symbolic Newton-Raphson", symbolic_newton, x0, fig, axes[0, 0])

        solve_and_plot(
            "Forward-Difference Newton-Raphson",
            forward_difference_newton,
            x0,
            fig,
            axes[0, 1],
        )

        solve_and_plot(
            "Center-Difference Newton-Raphson",
            center_difference_newton,
            x0,
            fig,
            axes[1, 0],
        )

        J0 = np.zeros((2, 2))
        J0[:, 0] = (f(x0 + np.array([h, 0])) - f(x0 - np.array([h, 0]))) / (2 * h)
        J0[:, 1] = (f(x0 + np.array([0, h])) - f(x0 - np.array([0, h]))) / (2 * h)
        print(f"Broyden's Method J0:\n{J0}")
        solve_and_plot(
            "Broydens Method, initialized with center-difference",
            BroydensMethod(J0),
            x0,
            fig,
            axes[1, 1],
        )

    plt.show()
```