ISSN: 0369-8963

Exploring Fixed Point Theorem in Numerical Analysis: A Comprehensive Python Implementation and Visualization

Poonam Maurya,Research Scholar, Bharti Vishwavidyalaya, Durg, (C.G.)
Rita Shukla, Associate professor, Bharti Vishwavidyalaya, Durg, (C.G.)
Manish Kumar Singh, Assistant professor, Bharti Vishwavidyalaya, Durg, (C.G.)

Bharti vishwavidyalaya, Balod Road, Chandkhuri, Durg,
Chhattisgarh, India
Pin: 491001

Abstract:
The use of fixed point theorems in numerical analysis is examined in this work, with an emphasis on how they might be applied to solve optimization and equation problems. Fixed point theorems are essential in many branches of mathematics and engineering because they state that there are points that do not change under specific functions. The paper offers a thorough Python implementation of important fixed point algorithms, emphasizing their useful applications in iterative techniques for solving nonlinear equations. These algorithms include the Banach and Brouwer fixed point theorems. Furthermore, visualizations are created to show how these algorithms perform in terms of convergence and efficacy in various contexts. The goal of the implementation is to give researchers and students in numerical analysis and associated fields an approachable way to comprehend fixed point theorems, as well as theoretical insights and useful tools. The constraints and potential difficulties of applying fixed point approaches in situations involving complicated problem solving are also covered in the study.
Keywords- Fixed Point Theorem, Numerical Analysis, Python Implementation, Iterative Methods, Convergence Visualization

1. Introduction
In the field of mathematics known as "fixed point theory," the existence, uniqueness, and characteristics of solutions to equations with the form $f(x) = x$—where f is a specified function—are examined. This seemingly easy equation has important ramifications and wide-ranging applications in a variety of domains, including pure mathematics and the solution of practical issues in computer science, physics, economics, and other areas.
Economics: The presence and stability of equilibria in game theory and economic models are examined using fixed point theory.
Physics: It is useful for understanding phase transitions and the behaviour of dynamical systems.
Computer science: It's used to create algorithms that solve optimization and equation issues.
Engineering: Control system design and structural stability analysis both make use of fixed point theory.
In short, A point x such that $f(x) = x$ is a fixed point of a function f(x).
Because fixed point stands for equilibrium conditions, stability, and answers to a variety of issues, it is significant. The instruments and procedures needed to thoroughly examine and comprehend the presence, characteristics, and actions of these unique points are provided by fixed point theory. The contraction mapping principle, which asserts that if f is a contraction mapping, then it has a unique fixed point, is the cornerstone of fixed point theory. A function that gradually shortens the distances between locations is known as a contraction mapping.

 https://doi.org/10.5281/zenodo.15614033

The well-known Banach fixed point theorem asserts that there is a single fixed point for any contraction mapping on a whole metric space.

Numerous additional significant theorems and applications exist in the large and dynamic area of fixed point theory [1-9]. A strong technique that may be used to a wide range of issues is fixed point theory. This discipline is expanding quickly and has seen a lot of interesting new discoveries.

There are three main branches of fixed point theory discussed in this study-

A. Banach Fixed-Point Theorem
B. Brouwer's Fixed Point Theorem
C. Newton's Iterative Method
D. Practical implementation: Solving a System of nonlinear equations.

The three main branches of fixed point theory are followed in the organization of this work. The existence of unique fixed points through Banach fixed point theorem is covered in Section 2. The third section examines Brouwer's Fixed Point Theorem. The primary findings pertaining to Newton's iterative method are discussed in Section 4. In section 5, we will solve a system of nonlinear equations using Newton's method demonstrating practical implementation of fixed point theorem. We alsoanalyse the implementation of these theorems using a python environment to simulate their working for better interpretability in each section.

2. Banach Fixed-Point Theorem

A major area in fixed point theory is the existence and uniqueness of fixed points. Finding the circumstances under which a particular function has fixed points and verifying the uniqueness of these points are the main goals of this field of study. Fixed points are parts of the domain of a function that don't change while the function is used. Understanding stability, equilibrium, and transformation behaviour in a variety of mathematical situations depends on this issue.

The Banach fixed-point theorem, often referred to as the contraction mapping principle, is a fundamental statement on the existence and uniqueness of fixed points. This theorem ensures that for a contraction mapping on a whole metric space, there is a fixed point and that it is unique. A contraction mapping is a function that guarantees convergence to a single equilibrium point by shortening the distance between locations. This result gives a strong instrument for proving the existence and uniqueness of fixed points in a variety of circumstances, and it forms the basis for other theorems.

The Banach Fixed-Point Theorem, a foundational result in the theory of contraction mappings, highlights the significance of this principle. This theorem, which bears the name of the Polish mathematician Stefan Banach [1], provides the foundation for comprehending the existence and uniqueness of fixed points for particular kinds of mappings.

The theorem basically gives the circumstances under which a function f on a whole metric space X has a single fixed point. The main idea is a contraction mapping, in which the distance between points is decreased following the transformation by the function *f*.

2.1 Implementation

Implementing the Banach Fixed-Point Theorem in Python involves defining a contraction mapping function and iteratively applying this function to an initial guess until convergence to the fixed point is achieved. A contraction mapping guarantees convergence to a single fixed point by ensuring that the distance between subsequent iterations shrinks. To put this approach into practice, a loop must be built up that runs until the fixed point is achieved, or until the difference between iterations is less than a predetermined tolerance threshold.

 https://doi.org/10.5281/zenodo.15614033

ISSN: 0369-8963

To demonstrate the implementation of the Banach Fixed-Point Theorem in Python with a specific function and plot the results, let's consider a simple contraction mapping:

$$f(x) = \cos(x)$$

Algorithm: Banach fixed point theorem

Input-

I. *f: Contraction mapping function*
II. *X_0: Initial guess*
III. *Tol: Tolerance for convergence*
IV. *Max_Iter: maximum Iterations*

Initialization-

V. *Set $X=X_0$*
VI. *Create empty history*

Iterative procedure

VII. *For i=1 to Max_Iter*

. *Calculate $X_{new}=f(x)$*
. *Append X_{new} to history*
. *If* $|X_{new}-X|<1$: update $X=X_{new}$

The result obtained by implementing this in python3.8 is shown in Fig.1. It shows both the convergence of the Banach Fixed point theorem and the convergence rate for the cosine function. It is clear that with an initial guess of value 1, it takes around 40 iterations to converge to the function's fixed point which is 0.739. According to the Fixed point theorem, cos(x)=x i.e. at x=0.739, cos(0.739)=0.739 (Note that x is in radians). Hence the solution is correct.

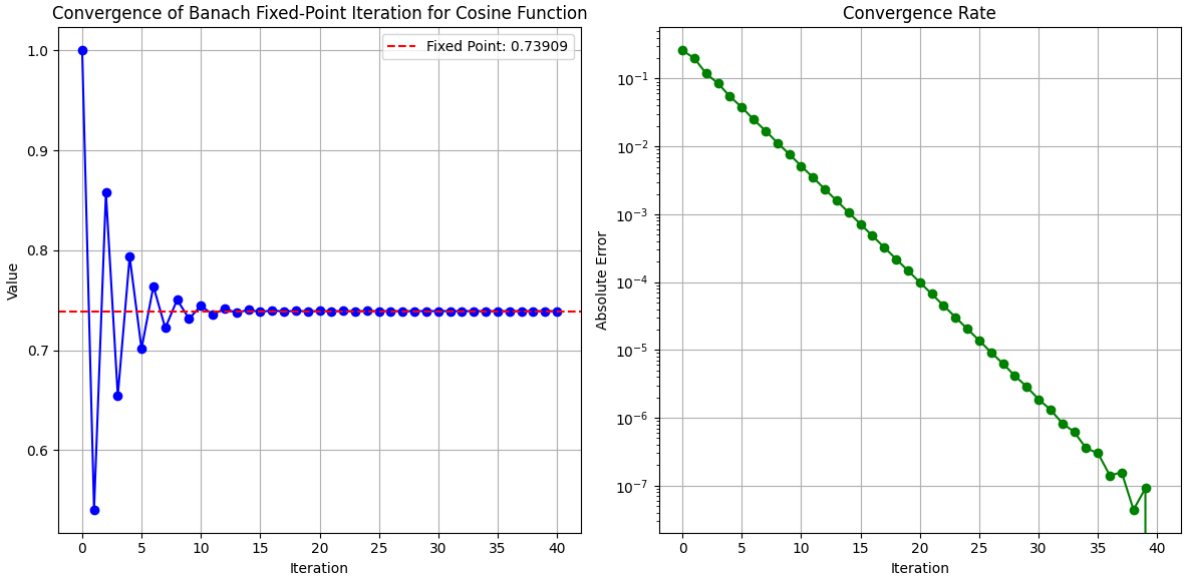

Fig.1 Convergence of the Banach Fixed point theorem for cosine function as well as the convergence rate under the maximum iterations.

3. Brouwer's Fixed Point Theorem

A pillar of fixed point theory and topology, Brouwer's theorem illustrates the larger context of fixed point existence. The theorem, first proved by Luitzen E.J. Brouwer in 1911 [3], states that there is always at least one fixed point for every continuous function *f* mapping a nonempty, compact, and convex subset D of Euclidean space R^n to itself. In other words, there is a point *x* within D such that $f(x) = x$.

https://doi.org/10.5281/zenodo.15614033

ISSN: 0369-8963

3.1 Implementation

To demonstrate the implementation of the Brouwer's Fixed-Point Theorem in Python with a specific function and plot the results, let's consider a simple contraction mapping:

$$f(x) = \sin(x)$$

Algorithm: Brouwer's Fixed Point Theorem

Input-

I. *f: Input function*
II. *X_0: Initial guess*
III. *Tol: Tolerance for convergence*
IV. *Max_Iter: maximum Iterations*

Initialization-

V. *Set $X=X_0$*
VI. *Create empty history*

Iterative procedure

VII. *For i=1 to Max_Iter*

. *Calculate $X_{new}=f(x)$*
. *Append X_{new} to history*
. *If $|X_{new}-X|<1$: update $X=X_{new}$*
. *Plot the function f(x) and successive iterations.*

The algorithm implementation is somewhat similar but there is a fundamental difference between both theorems. The Banach Fixed Point Theorem offers better guarantees for contraction mappings on complete metric spaces, whereas Brouwer's Fixed Point Theorem is more generic and covers a wider class of functions and domains and does not provide an exact way of finding the fixed point. The result is shown in Fig. 2, which demonstrates the convergence of function *f(x)* along the fixed point 0.02. According to the fixed point theorem, *sin(x)=x* i.e. *sin(0.02) =0.02*. Hence the implementation is correct. Note that *x* is in radians.

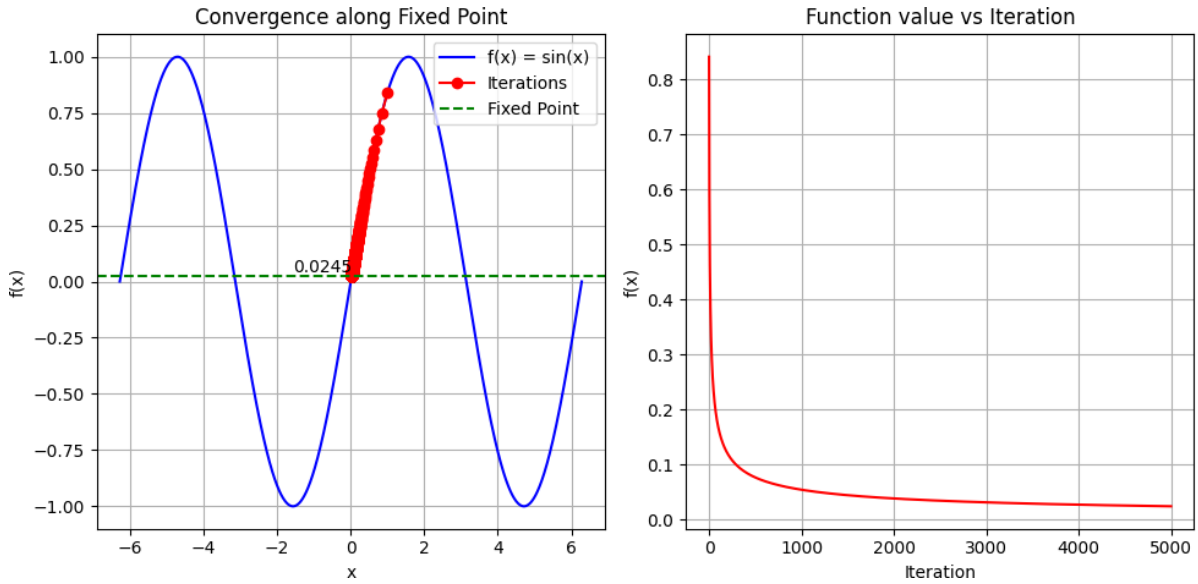

Fig.2 Brouwer's Fixed Point theorem implementation results in python showing the function plot along with its value on successive iterations.

4. Newton's Iterative method

Newton's iterative method or Newton-Raphson method is a special case of fixed point theorem. It is an iterative process of finding the fixed point of a function-

https://doi.org/10.5281/zenodo.15614033

ISSN: 0369-8963

$$g(x) = x - \frac{f(x)}{f'(x)}$$

Where *f(x)* is the function whose root we have to find and *f'(x)* is the derivative of the function.

Newton's approach iteratively converges to the roots of a function by utilizing the ideas and characteristics offered by the fixed-point theorem. The theoretical basis for Newton's method's efficacy in locating function roots is the existence and convergence of the fixed point of *g(x)*. In Newton's method, a fixed point corresponds to the root of *f(x)* such that if x is fixed point of *g(x)*then-

$$x = x - \frac{f(x)}{f'(x)}$$

$$f(x) = 0$$

Newton's method relies on convergence of a fixed point of *g(x)* in the neighbourhood of the root starting from the initial guess. The rate of convergence of Newton's method is quadratic and this converges much quickly.

4.1 Implementation

To implement Newton's method in python, let us lake a function f(x)-

$$f(x) = x^3-3x^2+2$$

Algorithm: Newton's Iterative Method

Input-

I. f(x): input function
II. f'(x): derivative of input function
III. x_0: Initial guess
IV. Tol: Tolerance for convergence
V. Max_Iter: maximum Iterations

Initialization-

VI. Set $x=x_0$
VII. Create empty history

Iterative procedure

VIII. For |f(x)|>*Tol and iterations<Max_Iter*

. *Calculate x_{new}*= x-f(x)/f'(x)
. *Update $x = x_{new}$*
. *Increment iteration*

Fig. 3 shows the convergence of Newton's method to find the root of function *f(x)* as well as the plot of f(x) vs iterations. The root of the function was found out to be (-0.84,0) in 11 iterations with an initial guess of (1.5,0.88).

https://doi.org/10.5281/zenodo.15614033

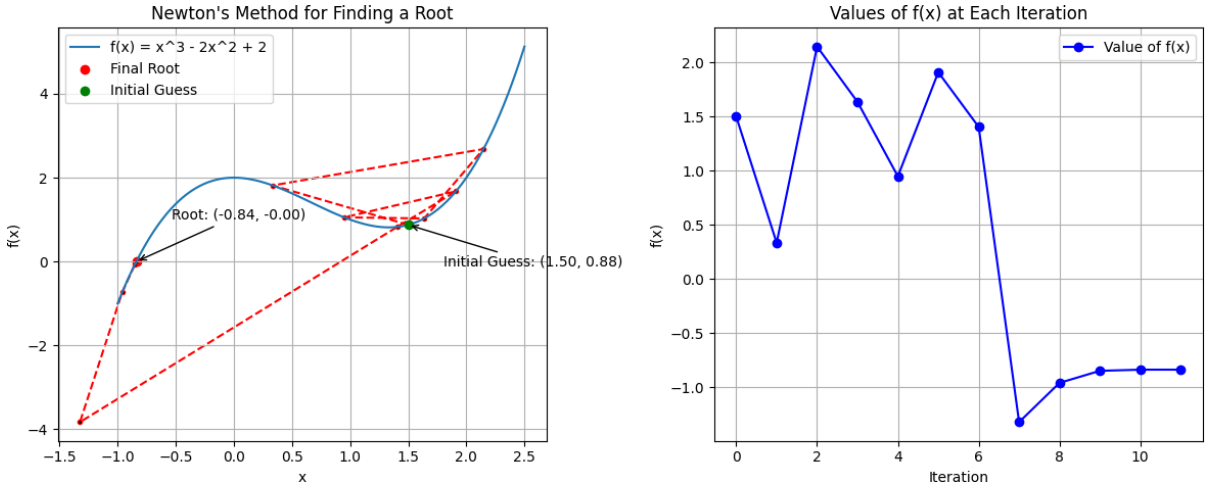

Fig. 3 Finding the root of the equation using Newton's method considering root as the fixed point of function *f(x)*and also showing the convergence with successive iterations.

5. Solving a system of nonlinear equations: A practical implementation

Let us consider two functions $f_1(x, y)$ and $f_2(x,y)$ defined in a system as follows:

$$f_1(x, y) = sin(x) + x^2 - y^2 - 1$$

$$f_2(x, y) = e^x + y^3 - 2$$

Algorithm- Implemented in Python
Input-
I. *f(x)€ $f_1(x, y)$ and $f_2(x, y)$: input functions in a system*
II. *J(x): Jacobian Matrix*
III. *x_0: Initial guess*
IV. *Tol: Tolerance for convergence*
V. *Max_Iter: maximum Iterations*
Initialization-
VI. *Set $x=x_0$*
VII. *Create empty history*
Iterative procedure
VIII. *For*|f(x)|>*Tol and iterations<Max_Iter*
- *A.* *Calculate $delX=-J(x)^{-1}f(x)$*
- *B.* *Calculate $x_{new}= x+delX$*
- *C.* *Update $x = x_{new}$*
- *D.* *Increment iteration*

Fig. 4 shows the surface plot for system of nonlinear functions. Through successive iterations, we can see in the Fig. 5 that the solution to the system of equations is approximately [0.67951804 0.30027148] found in 11 iterations with initial guess [0,0.1].

https://doi.org/10.5281/zenodo.15614033

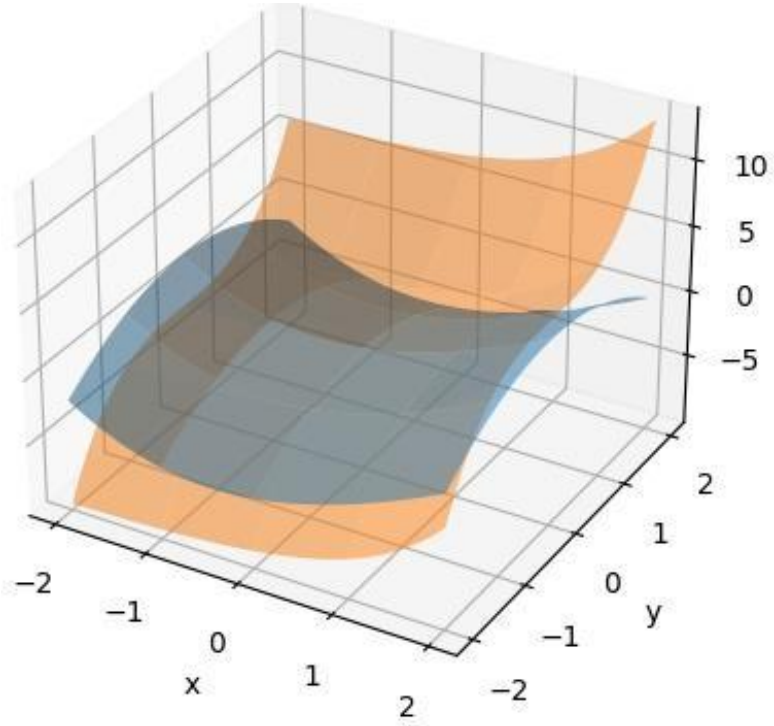

Fig. 4 Surface plot of defined system of nonlinear functions.

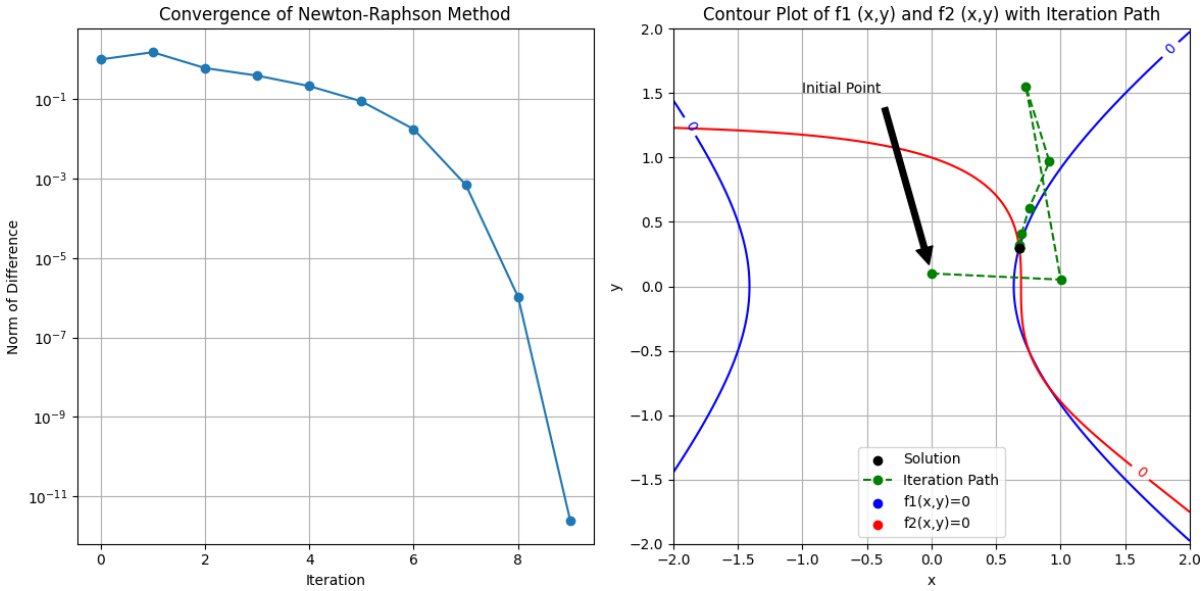

Fig. 5 Solution to the system of nonlinear functions using Newton's method showing convergence with respect to iterations and contour plot showing value of x after every iteration till the final value.

6. Conclusion

In this study, we have effectively used Python 3.8 to implement the fixed-point theorem and Newton's method being a special case of fixed point theorem. We were able to solve nonlinear equations efficiently by utilizing the fixed-point iteration technique. The graphic analysis shed light on each method's convergence tendency by showing how quickly it approaches the fixed point when given a credible starting guess. The following points are the key takeaways of this study-

https://doi.org/10.5281/zenodo.15614033

A. The existence and uniqueness of fixed points for contraction mappings on complete metric spaces are ensured by Banach's fixed-point theorem, sometimes referred to as the contraction mapping theorem.
B. Any continuous function that maps a compact convex set to itself has at least one fixed point, according to Brouwer's fixed-point theorem. This theorem is fundamental to several disciplines, such as game theory and economics.
C. Newton's method relies on the iterative formula $g(x) = x - \frac{f(x)}{f'(x)}$which is derived from the fixed point theorem.
D. We plotted the function, the root, and the iterative process using matplotlib in the python environment. The plots offered a good visual representation of the convergence behaviour of the approach.
E. Finally, a practical implementation of fixed point theorem for finding out the solution to the system of nonlinear equations is given. Through visualization, the convergence after each iteration is clear and the solution can also be seen on the plot where both functions intersect.

7. References

Banach, S. (1922). Sur les operations dans les ensembles abstraits et leur application aux equations integrales. FundamentaMathematicae, 3, 133–181.
Border, K. C. (1985). Fixed Point Theorems with Applications to Economics and Game Theory. Cambridge University Press.
Brouwer, L. E. J. (1912). Uber Abbildung von Mannigfaltigkeiten. Math. Ann., 71, 97–115.
Farmakis, I., & Moskowitz, M. (2013). Fixed Point Theorems and Their Applications. World Scientific Publishing Company.
Goebel, K., & Kirk, W. A. (1990). Topics in metric fixed point theory (No. 28). Cambridge University Press.
Granas, A., &Dugundji, J. (2003). Fixed point theory (Vol. 14, pp. 15-16). New York: Springer.
Pata, V. (2019). Fixed point theorems and applications (Vol. 116). Cham: Springer. 92.
Pathak, H.K. (2018). An Introduction to Nonlinear Analysis and Fixed Point Theory. Springer.
Subrahmanyam, P. V. (2018). Elementary Fixed Point Theorems (Forum for Interdisciplinary Mathematics). Singapore: Springer.
Xie, L., Li, J., & Wen, C. F. (2013). Applications of fixed point theory to extended Nash equilibriums of nonmonetized noncooperative games on posets. Fixed Point Theory and Applications, 2013, 1-13.

APPENDIX

A. Python code for Banach Fixed Point Theorem

```
####start#########
import numpy as np
import matplotlib.pyplot as plt


def banach_fixed_point(f, x0, tol=1e-7, max_iter=1000):
    """
    Implements the Banach Fixed-Point Theorem to find the fixed point of a
contraction mapping.
```

 https://doi.org/10.5281/zenodo.15614033

ISSN: 0369-8963

```
    Parameters:
    f (function): The contraction mapping function.
    x0 (float): Initial guess.
tol (float): Tolerance for convergence.
max_iter (int): Maximum number of iterations.

    Returns:
    float: The fixed point.
    list: History of iterates.
    """
    x = x0
    history = [x0]
    for i in range(max_iter):
x_new = f(x)
history.append(x_new)
        if abs(x_new - x) <tol:
            return x_new, history
        x = x_new
    raise Exception("The method did not converge")

# Example contraction mapping function
def contraction_mapping(x):
    return np.cos(x)

# Initial guess
x0 = 1.0

# Finding the fixed point
fixed_point, history = banach_fixed_point(contraction_mapping, x0)

# Display the result
print(f"The fixed point is: {fixed_point}")

# Plotting the results
iterations = np.arange(len(history))
plt.figure(figsize=(12, 6))

# Plot 1: Convergence of iterates
plt.subplot(1, 2, 1)
plt.plot(iterations, history, marker='o', linestyle='-', color='blue')
plt.axhline(y=fixed_point, color='r', linestyle='--', label=f'Fixed Point:
{fixed_point:.5f}')
```


https://doi.org/10.5281/zenodo.15614033

```
plt.xlabel('Iteration')
plt.ylabel('Value')
plt.title('Convergence of Banach Fixed-Point Iteration for Cosine Function')
plt.legend()
plt.grid(True)

# Plot 2: Convergence Rate
plt.subplot(1, 2, 2)
errors = np.abs(np.array(history) - fixed_point)
plt.semilogy(iterations, errors, marker='o', linestyle='-', color='green')
plt.xlabel('Iteration')
plt.ylabel('Absolute Error')
plt.title('Convergence Rate')
plt.grid(True)
plt.tight_layout()
plt.show()
####end######
```

B. Python code for Brouwer's Fixed point theorem

```
########start########
import numpy as np
import matplotlib.pyplot as plt

# Define the function
def f(x):
    return np.sin(x)

# Implementing the fixed point iteration method
def fixed_point_iteration(x0, tol=1e-6, max_iter=5000):
x_iterations = [x0]
y_iterations = [f(x0)]

    # Iterate until convergence or maximum iterations reached
    for i in range(1, max_iter):
        x1 = f(x_iterations[-1])
x_iterations.append(x1)
y_iterations.append(f(x1))
        if abs(x1 - x_iterations[-2]) <tol:
            break

    return x_iterations, y_iterations, x1

# Plot the function, its iterations, and the fixed point
```

 https://doi.org/10.5281/zenodo.15614033

ISSN: 0369-8963

```
def plot_function_and_iterations(x_iterations, y_iterations, fixed_point):
x_values = np.linspace(-2*np.pi, 2*np.pi, 1000)
y_values = f(x_values)
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(x_values, y_values, label='f(x) = sin(x)', color='blue')
plt.plot(x_iterations, y_iterations, color='red', linestyle='-', marker='o',
label='Iterations')
plt.axhline(y=f(fixed_point), color='green', linestyle='--', label='Fixed Point')
plt.text(fixed_point, f(fixed_point), f'{fixed_point:.4f}', ha='right', va='bottom')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Convergence along Fixed Point')
plt.grid(True)
plt.legend()

   # Plot value vs iteration
plt.subplot(1, 2, 2)
plt.plot(range(len(y_iterations)), y_iterations, color='red', linestyle='-')
plt.xlabel('Iteration')
plt.ylabel('f(x)')
plt.title('Function value vs Iteration')
plt.grid(True)

plt.tight_layout()
plt.show()

# Example usage
if __name___== "__main__":
   x0 =1# Initial guess
x_iterations, y_iterations, fixed_point = fixed_point_iteration(x0)
plot_function_and_iterations(x_iterations, y_iterations, fixed_point)
##########end#################
```

C. Python code for Newton-Raphson Method

```
##################start#################
import numpy as np
import matplotlib.pyplot as plt

# Define the function and its derivative
def f(x):
   return x**3 - 2*x**2 + 2
```


https://doi.org/10.5281/zenodo.15614033

ISSN: 0369-8963

```
def df(x):
    return 3*x**2 - 4*x

# Implement Newton's method
def newtons_method(f, df, x0, tol=1e-6, max_iter=100):
    x = x0
x_values = [x0]  # List to store values of x at each iteration
f_values = [f(x0)]  # List to store values of f(x) at each iteration
    iterations = 0
    while abs(f(x)) >tol and iterations <max_iter:
        x = x - f(x) / df(x)
x_values.append(x)
f_values.append(f(x))
        iterations += 1
    if iterations == max_iter:
print("Maximum iterations reached. Convergence not achieved.")
    else:
print(f"Root found: {x} after {iterations} iterations.")
    return x, x_values, f_values

# Define the range of x values for plotting
x_values = np.linspace(-1, 2.5, 100)
y_values = f(x_values)

# Find the root using Newton's method
initial_guess = 1.5
root, x_iter, f_iter = newtons_method(f, df, initial_guess)

# Plot the function and values vs iteration side by side
fig, axs = plt.subplots(1, 2, figsize=(12, 5))

# Plot the function
axs[0].plot(x_values, y_values, label='f(x) = x^3 - 2x^2 + 2')
axs[0].scatter(root, f(root), color='red', label='Final Root')

# Plot the values of f(x) after each iteration
axs[0].plot(x_iter, f_iter, linestyle='--', color='r',marker= '.',mfc='k',zorder=1)

# Annotate the root
axs[0].annotate(f'Root: ({root:.2f}, {f(root):.2f})', xy=(root, f(root)),
xytext=(root+0.3, f(root)+1),
arrowprops=dict(facecolor='black', arrowstyle='->'))
```


https://doi.org/10.5281/zenodo.15614033

ISSN: 0369-8963

```
# Annotate the initial guess point
axs[0].scatter(initial_guess, f(initial_guess), color='green', label='Initial
Guess',zorder=2)
axs[0].annotate(f'Initial Guess: ({initial_guess:.2f}, {f(initial_guess):.2f})',
xy=(initial_guess, f(initial_guess)),
xytext=(initial_guess+0.3, f(initial_guess)-1), arrowprops=dict(facecolor='black',
arrowstyle='->'))

axs[0].set_xlabel('x')
axs[0].set_ylabel('f(x)')
axs[0].set_title('Newton\'s Method for Finding a Root')
axs[0].legend()
axs[0].grid(True)

# Plot values vs iteration
iterations = range(len(x_iter))
axs[1].plot(iterations, x_iter, marker='o', label='Value of f(x)',color='b')
axs[1].set_xlabel('Iteration')
axs[1].set_ylabel('f(x)')
axs[1].set_title('Values of f(x) at Each Iteration')
axs[1].grid(True)
axs[1].legend()

plt.tight_layout()
plt.show()
############end###################
```

D. Python code for solving system of nonlinear equations

```
#############start################
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Define the system of nonlinear equations
def F(X):
    x, y = X
    return np.array([np.sin(x) + x**2 - y**2 - 1, np.exp(x) + y**3 - 2])

# Define the Jacobian matrix of the system
def J(X):
    x, y = X
    return np.array([[np.cos(x) + 2*x, -2*y],
                     [np.exp(x), 3*y**2]])
```

 https://doi.org/10.5281/zenodo.15614033

ISSN: 0369-8963

```
# Implement the fixed point iteration method (Newton-Raphson for systems)
def newton_raphson_system(F, J, initial_guess, tolerance=1e-6, max_iterations=100):
    X = np.array(initial_guess)
    history = [X]
    for iteration in range(max_iterations):
        FX = F(X)
        JX = J(X)
delta_X = np.linalg.solve(JX, -FX)
X_new = X + delta_X
history.append(X_new)
        if np.linalg.norm(X_new - X) < tolerance:
print(f'Converged to solution after {iteration} iterations.')
            return X_new, np.array(history)
        X = X_new
    raise ValueError('Did not converge to a solution within the maximum number of
iterations.')

# Set the initial guess
initial_guess = [0, 0.1]

# Find the solution
solution, history = newton_raphson_system(F, J, initial_guess)

print(f'The solution to the system is approximately {solution}')

# Convergence plot
norms = [np.linalg.norm(history[i] - history[i-1]) for i in range(1, len(history))]

# Surface and contour plots
x = np.linspace(-2, 2, 400)
y = np.linspace(-2, 2, 400)
X, Y = np.meshgrid(x, y)
Z1 = np.sin(X) + X**2 - Y**2 - 1
Z2 = np.exp(X) + Y**3 - 2

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

# Plot Convergence
ax1.plot(norms, marker='o')
ax1.set_yscale('log')
ax1.set_xlabel('Iteration')
ax1.set_ylabel('Norm of Difference')
```


https://doi.org/10.5281/zenodo.15614033

ISSN: 0369-8963

```
ax1.set_title('Convergence of Newton-Raphson Method')
ax1.grid(True)

# Plot Contour
CS1 = ax2.contour(X, Y, Z1, levels=[0], colors='b')
CS2 = ax2.contour(X, Y, Z2, levels=[0], colors='r')
ax2.clabel(CS1, inline=1, fontsize=10)
ax2.clabel(CS2, inline=1, fontsize=10)
ax2.scatter(solution[0], solution[1], color='k', zorder=5)
ax2.plot(*zip(*history), marker='o', color='g', linestyle='--', zorder=4)
ax2.scatter([], [], color='b', label='f1(x,y)=0')
ax2.scatter([], [], color='r', label='f2(x,y)=0')
ax2.annotate('Initial Point', xy=(initial_guess[0], initial_guess[1]), xytext=(-1, 1.5),
arrowprops=dict(facecolor='black', shrink=0.05), fontsize=10, color='black')
ax2.set_xlabel('x')
ax2.set_ylabel('y')
ax2.set_title('Contour Plot of f1 (x,y) and f2 (x,y) with Iteration Path')
ax2.legend(['Solution', 'Iteration Path', 'f1(x,y)=0', 'f2(x,y)=0'], loc='lower center')
ax2.grid(True)

plt.tight_layout()
plt.show()
# 3D Surface plots
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z1, alpha=0.5, rstride=100, cstride=100)
ax.plot_surface(X, Y, Z2, alpha=0.5, rstride=100, cstride=100)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('Function Value')
ax.set_title('Surface Plots of f1 (x,y) and f2 (x,y)')
plt.show()
##############end####################
```

 https://doi.org/10.5281/zenodo.15614033