cs-374-lab-4

August 28, 2024

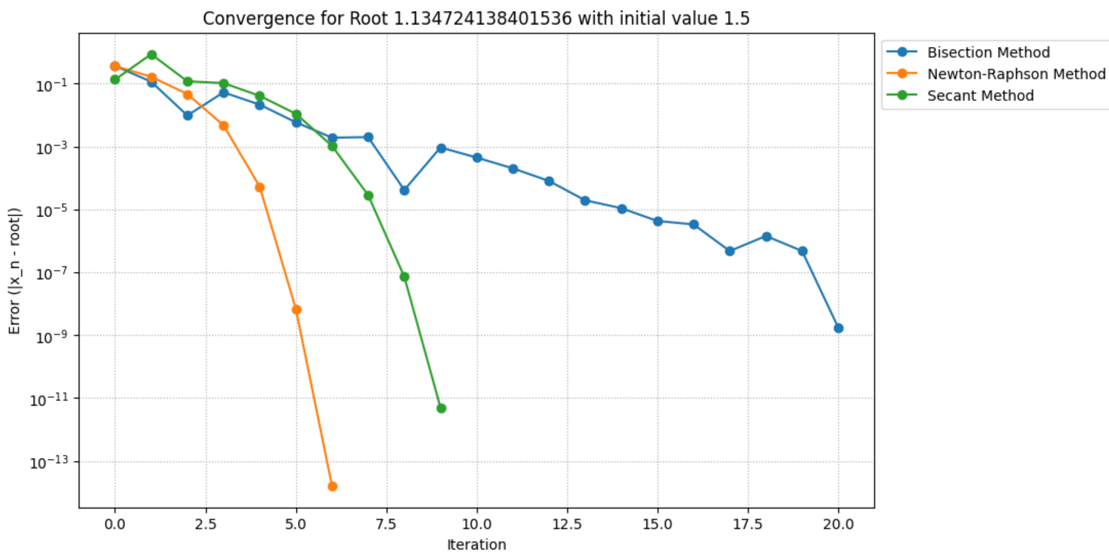
Lab 4

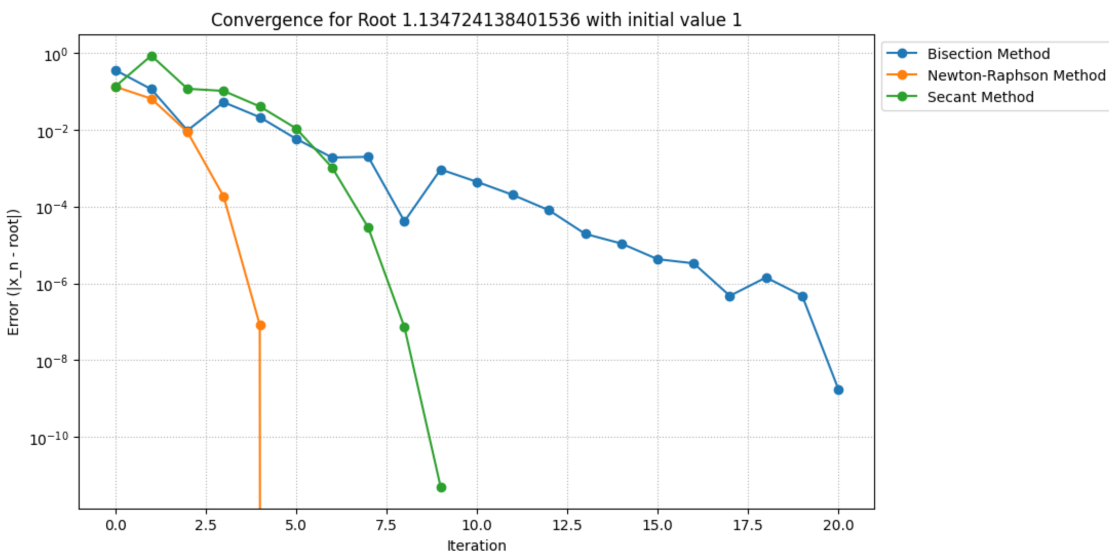
Question 1

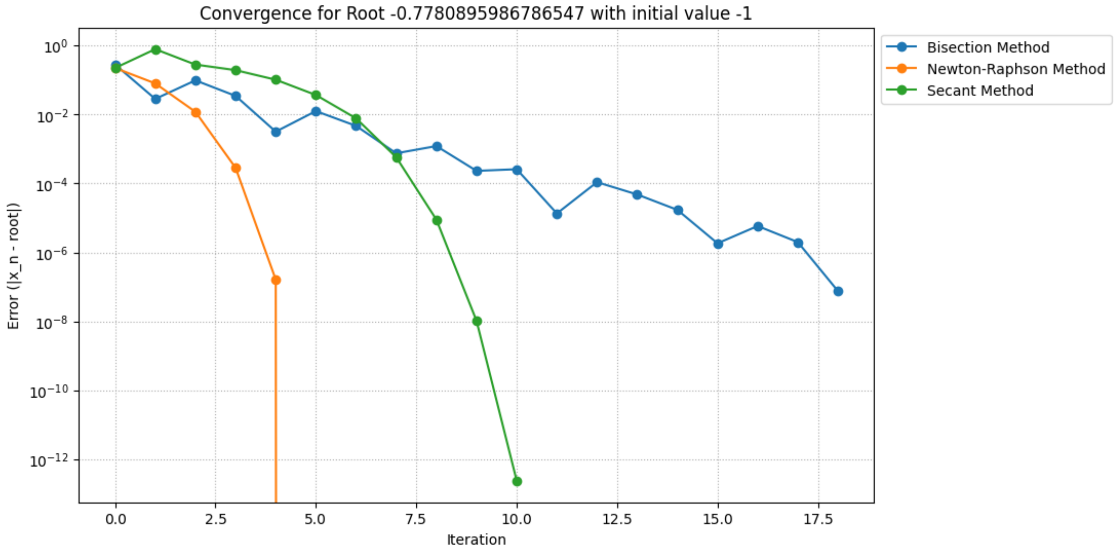
[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

# Function definition: f(x) = x^6 - x - 1 = 0
def f(x):
    return x**6 - x - 1

# Derivative of f(x) needed for Newton-Raphson method
def df(x):
    return 6*x**5 - 1

# Bisection method implementation
def bisection_method(f, a, b, tolerance=1e-6, max_iterations=100):
    iterations = []
    for i in range(max_iterations):
        c = (a + b) / 2
        iterations.append(c)
        if abs(f(c)) < tolerance:
            break
        elif f(a) * f(c) < 0:
            b = c
        else:
            a = c
    return iterations

# Newton-Raphson method implementation
def newton_raphson_method(f, df, x0, tolerance=1e-6, max_iterations=100):
    iterations = [x0]
    for i in range(max_iterations):
        x1 = x0 - f(x0) / df(x0)
        iterations.append(x1)
        if abs(x1 - x0) < tolerance:
            break
        x0 = x1
```

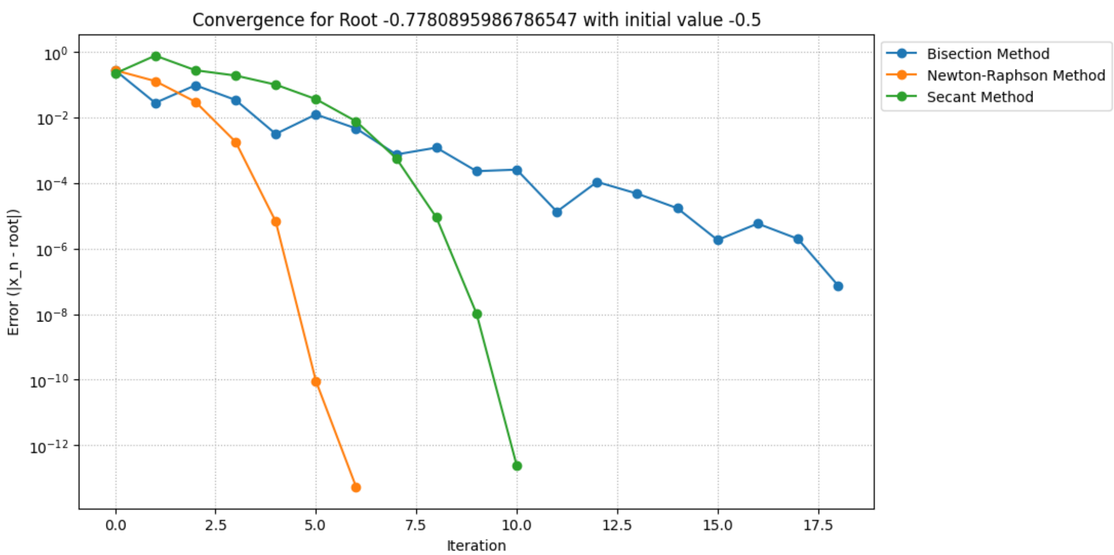
```python
    return iterations

def secant_method(f, x0, x1, tolerance=1e-6, max_iterations=100):
    iterations = [x0, x1]
    for i in range(max_iterations):
        if abs(f(x1) - f(x0)) < 1e-12:
            break  # Avoid division by a very small number
        x2 = x1 - f(x1) * (x1 - x0) / (f(x1) - f(x0))
        iterations.append(x2)
        if abs(x2 - x1) < tolerance:
            break
        x0 = x1
        x1 = x2
    return iterations

# Plotting the convergence of the methods
def plot_convergence(iterations, method_name, root):
    errors = [abs(x - root) for x in iterations]
    plt.plot(range(len(errors)), errors, marker='o', label=method_name)

# Define the root values
root_1 = 1.134724138401536
root_2 = -0.7780895986786547

# Plot for Root 1 with initial value 1.5
bisection_iters_1 = bisection_method(f, 1, 2)
newton_iters_1 = newton_raphson_method(f, df, 1.5)
secant_iters_1 = secant_method(f, 1, 2)

plt.figure(figsize=(10, 6))
plot_convergence(bisection_iters_1, "Bisection Method", root_1)
plot_convergence(newton_iters_1, "Newton-Raphson Method", root_1)
plot_convergence(secant_iters_1, "Secant Method", root_1)
plt.yscale('log')
plt.xlabel("Iteration")
plt.ylabel("Error (|x_n - root|)")
plt.title(f"Convergence for Root {root_1} with initial value 1.5")
plt.legend(loc ='upper left',bbox_to_anchor = (1,1))
plt.grid(True,linestyle=':')
plt.show()

# Plot for Root 1 with initial value 1
bisection_iters_2 = bisection_method(f, 1, 2)
newton_iters_2 = newton_raphson_method(f, df, 1)
secant_iters_2 = secant_method(f, 1, 2)

plt.figure(figsize=(10, 6))
```

```python
plot_convergence(bisection_iters_2, "Bisection Method", root_1)
plot_convergence(newton_iters_2, "Newton-Raphson Method", root_1)
plot_convergence(secant_iters_2, "Secant Method", root_1)
plt.yscale('log')
plt.xlabel("Iteration")
plt.ylabel("Error (|x_n - root|)")
plt.title(f"Convergence for Root {root_1} with initial value 1")
plt.legend(loc ='upper left',bbox_to_anchor = (1,1))
plt.grid(True,linestyle=':')
plt.show()

# Plot for Root 2 with initial value -1
bisection_iters_3 = bisection_method(f, -1, 0)
newton_iters_3 = newton_raphson_method(f, df, -1)
secant_iters_3 = secant_method(f, -1, 0)

plt.figure(figsize=(10, 6))
plot_convergence(bisection_iters_3, "Bisection Method", root_2)
plot_convergence(newton_iters_3, "Newton-Raphson Method", root_2)
plot_convergence(secant_iters_3, "Secant Method", root_2)
plt.yscale('log')
plt.xlabel("Iteration")
plt.ylabel("Error (|x_n - root|)")
plt.title(f"Convergence for Root {root_2} with initial value -1")
plt.legend(loc ='upper left',bbox_to_anchor = (1,1))
plt.grid(True,linestyle=':')
plt.show()

# Plot for Root 2 with initial value -0.5
bisection_iters_4 = bisection_method(f, -1, 0)
newton_iters_4 = newton_raphson_method(f, df, -0.5)
secant_iters_4 = secant_method(f, -1, 0)

plt.figure(figsize=(10, 6))
plot_convergence(bisection_iters_4, "Bisection Method", root_2)
plot_convergence(newton_iters_4, "Newton-Raphson Method", root_2)
plot_convergence(secant_iters_4, "Secant Method", root_2)
plt.yscale('log')
plt.xlabel("Iteration")
plt.ylabel("Error (|x_n - root|)")
plt.title(f"Convergence for Root {root_2} with initial value -0.5")
plt.legend(loc ='upper left',bbox_to_anchor = (1,1))
plt.grid(True,linestyle=':')
plt.show()
```


Convergence for Root 1.134724138401536 with initial value 1.5
Bisection Method
Newton-Raphson Method
Secant Method
Error (|x_n - root|)
Iteration

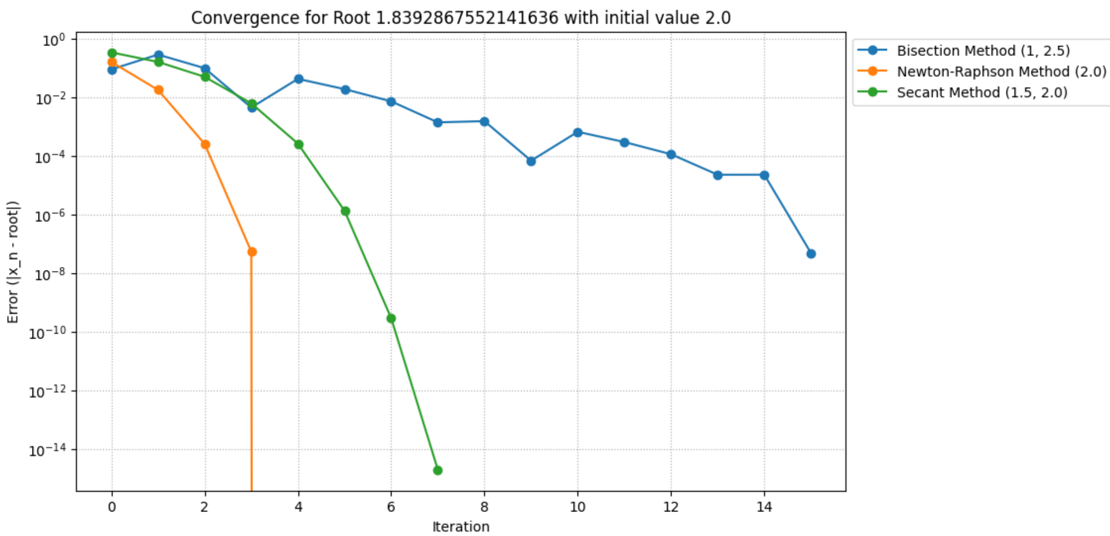

Convergence for Root 1.134724138401536 with initial value 1
Bisection Method
Newton-Raphson Method
Secant Method
Error (|x_n - root|)
Iteration

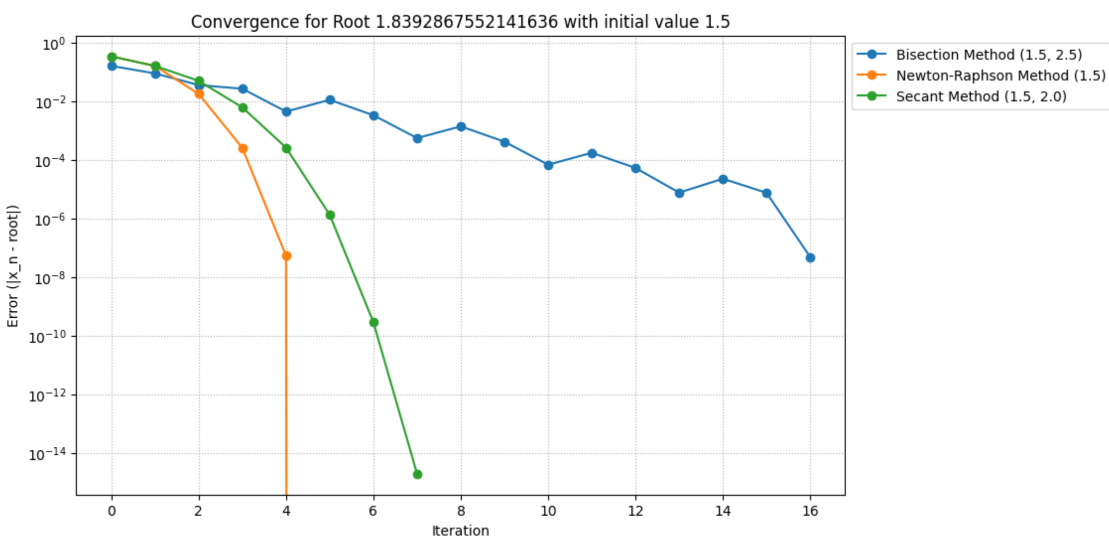




[ ]:
```
def f(x):
    return x**3 - x**2 - x - 1

# Derivative of f(x) needed for Newton-Raphson method
def df(x):
    return 3*x**2 - 2*x - 1

# Define the root value
root = 1.8392867552141636
```

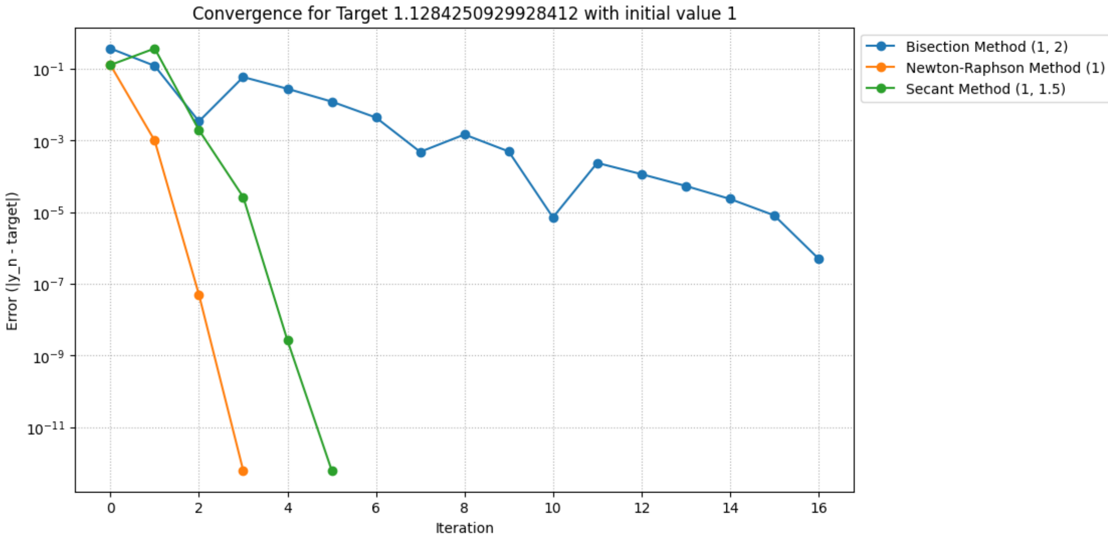
```python
# Initial values for plotting
initial_values = [
    (2.0, 1.5),  # (x0, x1) for Secant Method
]

# Plot for initial value 2.0
plt.figure(figsize=(10, 6))
# Bisection method
bisection_iters_1 = bisection_method(f, 1, 2.5)
plot_convergence(bisection_iters_1, "Bisection Method (1, 2.5)", root)
# Newton-Raphson method
newton_iters_1 = newton_raphson_method(f, df, 2.0)
plot_convergence(newton_iters_1, "Newton-Raphson Method (2.0)", root)
# Secant method
secant_iters_1 = secant_method(f, 1.5, 2.0)
plot_convergence(secant_iters_1, "Secant Method (1.5, 2.0)", root)
plt.yscale('log')
plt.xlabel("Iteration")
plt.ylabel("Error (|x_n - root|)")
plt.title("Convergence for Root 1.8392867552141636 with initial value 2.0")
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.grid(True, linestyle=':')
plt.show()

# Plot for initial value 1.5
plt.figure(figsize=(10, 6))
# Bisection method
bisection_iters_2 = bisection_method(f, 1.5, 2.5)
plot_convergence(bisection_iters_2, "Bisection Method (1.5, 2.5)", root)
# Newton-Raphson method
newton_iters_2 = newton_raphson_method(f, df, 1.5)
plot_convergence(newton_iters_2, "Newton-Raphson Method (1.5)", root)
# Secant method
secant_iters_2 = secant_method(f, 1.5, 2.0)
plot_convergence(secant_iters_2, "Secant Method (1.5, 2.0)", root)
plt.yscale('log')
plt.xlabel("Iteration")
plt.ylabel("Error (|x_n - root|)")
plt.title("Convergence for Root 1.8392867552141636 with initial value 1.5")
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.grid(True, linestyle=':')
plt.show()
```

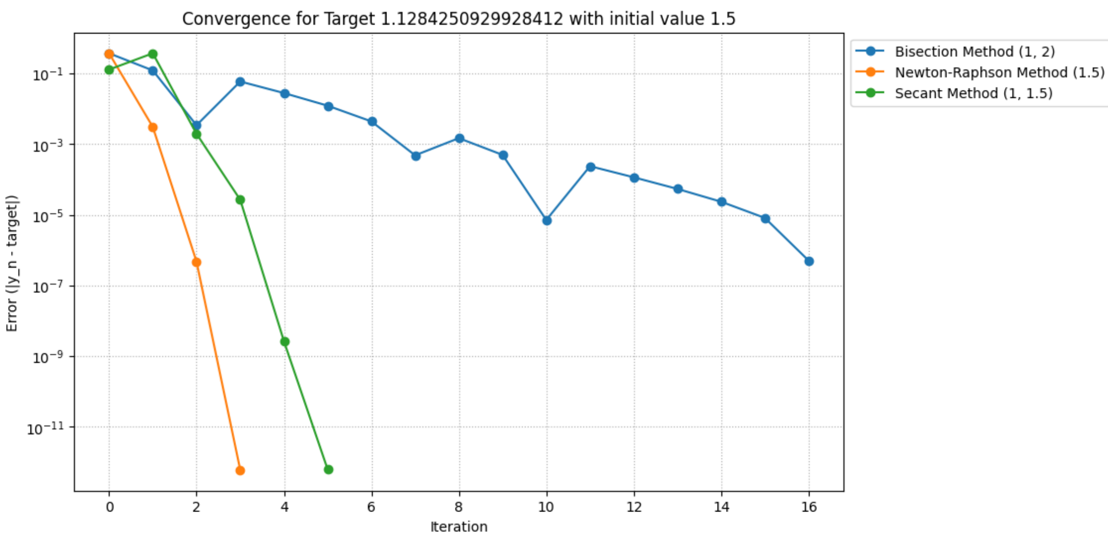




[ ]:
```python
def g(y):
    return 1 + 0.3 * np.cos(y) - y

# Derivative of g(y) needed for Newton-Raphson method
def dg(y):
    return -0.3 * np.sin(y) - 1

# Define the target value
target = 1.1284250929928412

# Plot for initial value 1
```

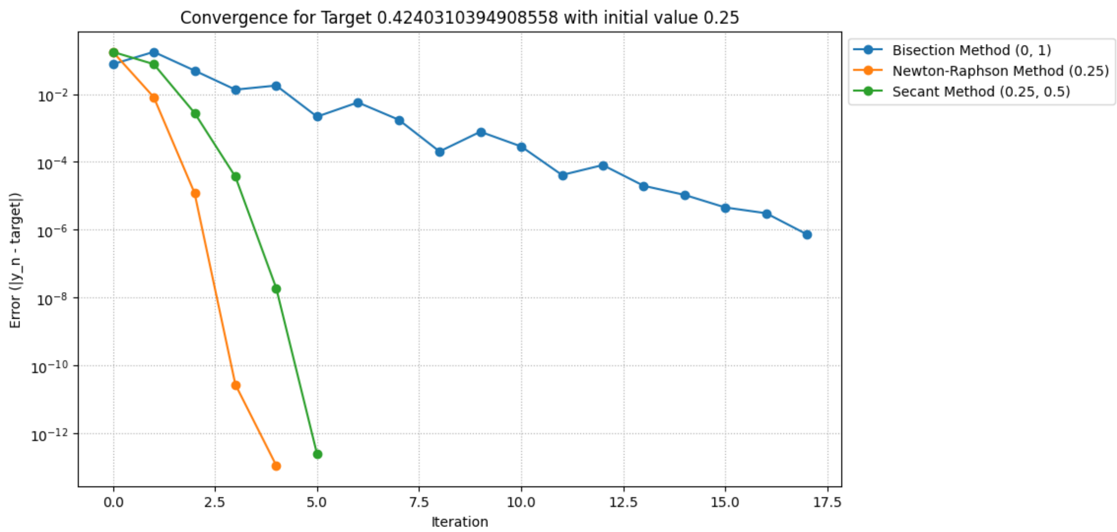
```python
plt.figure(figsize=(10, 6))
# Bisection method
bisection_results_1 = bisection_method(g, 1, 2)
plot_convergence(bisection_results_1, "Bisection Method (1, 2)", target)
# Newton-Raphson method
newton_results_1 = newton_raphson_method(g, dg, 1)
plot_convergence(newton_results_1, "Newton-Raphson Method (1)", target)
# Secant method
secant_results_1 = secant_method(g, 1, 1.5)
plot_convergence(secant_results_1, "Secant Method (1, 1.5)", target)
plt.yscale('log')
plt.xlabel("Iteration")
plt.ylabel("Error (|y_n - target|)")
plt.title("Convergence for Target 1.1284250929928412 with initial value 1")
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.grid(True, linestyle=':')
plt.show()

# Plot for initial value 1.5
plt.figure(figsize=(10, 6))
# Bisection method
bisection_results_2 = bisection_method(g, 1, 2)
plot_convergence(bisection_results_2, "Bisection Method (1, 2)", target)
# Newton-Raphson method
newton_results_2 = newton_raphson_method(g, dg, 1.5)
plot_convergence(newton_results_2, "Newton-Raphson Method (1.5)", target)
# Secant method
secant_results_2 = secant_method(g, 1, 1.5)
plot_convergence(secant_results_2, "Secant Method (1, 1.5)", target)
plt.yscale('log')
plt.xlabel("Iteration")
plt.ylabel("Error (|y_n - target|)")
plt.title("Convergence for Target 1.1284250929928412 with initial value 1.5")
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.grid(True, linestyle=':')
plt.show()
```

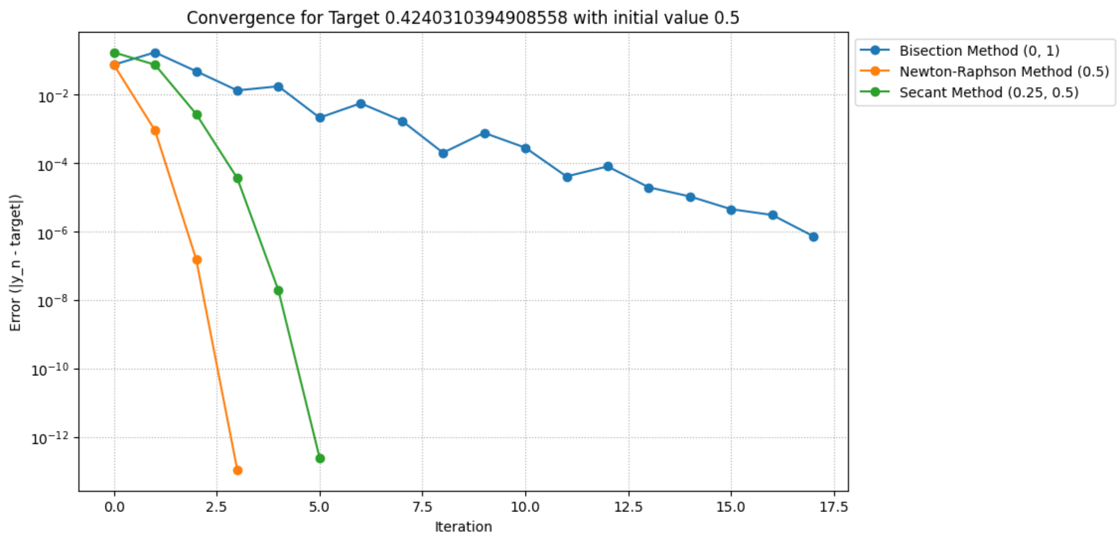




[ ]:
```
def g(y):
    return np.cos(y) - np.sin(y) - 0.5

# Derivative of g(y) needed for Newton-Raphson method
def dg(y):
    return -(np.sin(y) + np.cos(y))

# Define the target value
target = 0.4240310394908558

# Plot for initial value 0.25
```

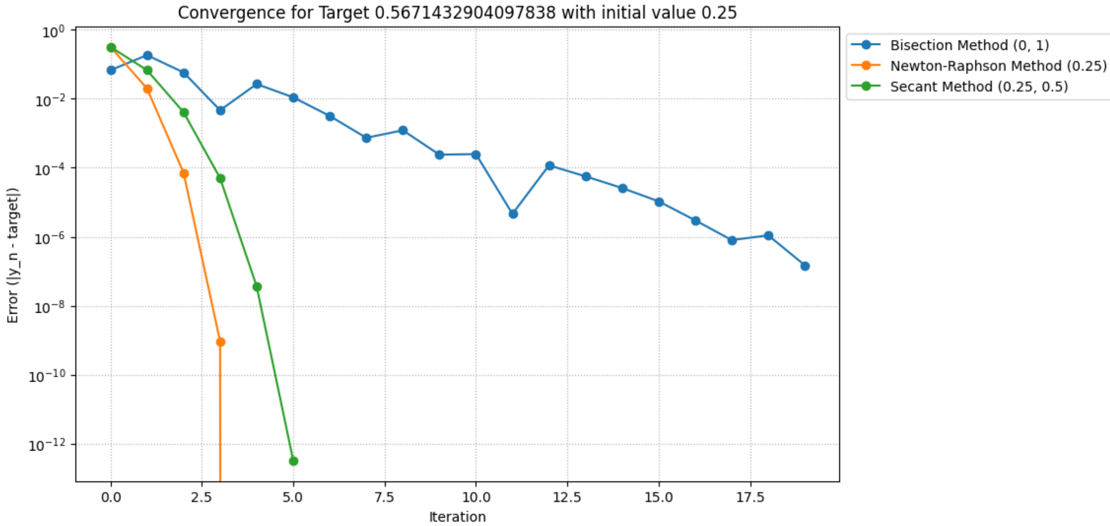
```
plt.figure(figsize=(10, 6))
# Bisection method
bisection_results_1 = bisection_method(g, 0, 1)
plot_convergence(bisection_results_1, "Bisection Method (0, 1)", target)
# Newton-Raphson method
newton_results_1 = newton_raphson_method(g, dg, 0.25)
plot_convergence(newton_results_1, "Newton-Raphson Method (0.25)", target)
# Secant method
secant_results_1 = secant_method(g, 0.25, 0.5)
plot_convergence(secant_results_1, "Secant Method (0.25, 0.5)", target)
plt.yscale('log')
plt.xlabel("Iteration")
plt.ylabel("Error (|y_n - target|)")
plt.title("Convergence for Target 0.4240310394908558 with initial value 0.25")
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.grid(True, linestyle=':')
plt.show()

# Plot for initial value 0.5
plt.figure(figsize=(10, 6))
# Bisection method
bisection_results_2 = bisection_method(g, 0, 1)
plot_convergence(bisection_results_2, "Bisection Method (0, 1)", target)
# Newton-Raphson method
newton_results_2 = newton_raphson_method(g, dg, 0.5)
plot_convergence(newton_results_2, "Newton-Raphson Method (0.5)", target)
# Secant method
secant_results_2 = secant_method(g, 0.25, 0.5)
plot_convergence(secant_results_2, "Secant Method (0.25, 0.5)", target)
plt.yscale('log')
plt.xlabel("Iteration")
plt.ylabel("Error (|y_n - target|)")
plt.title("Convergence for Target 0.4240310394908558 with initial value 0.5")
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.grid(True, linestyle=':')
plt.show()
```

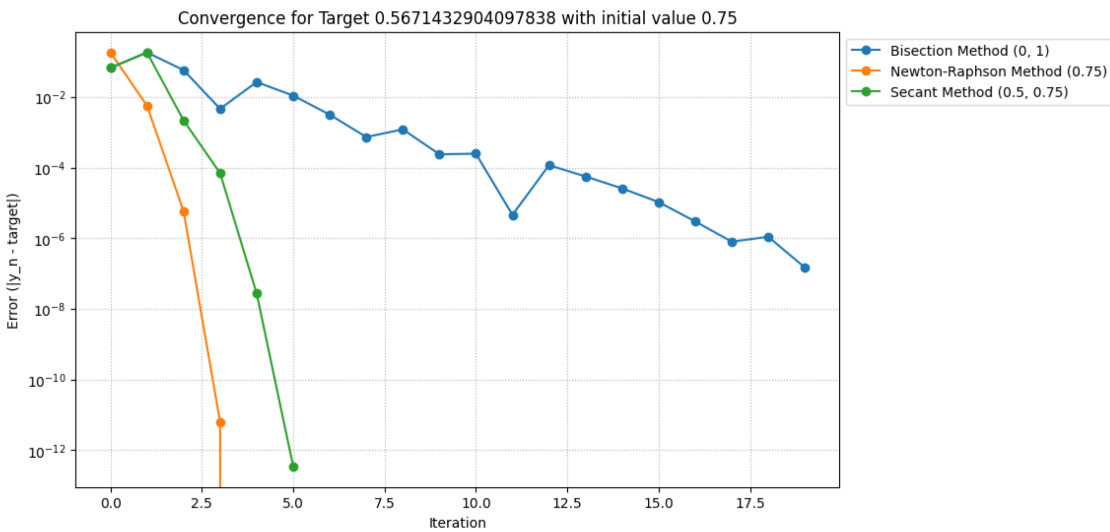




[ ]:
```
def g(y):
    return y - np.exp(-y)

# Derivative of g(y) needed for Newton-Raphson method
def dg(y):
    return 1 + np.exp(-y)

# Define the target value
target = 0.5671432904097838

# Plot for initial value 0.25
```

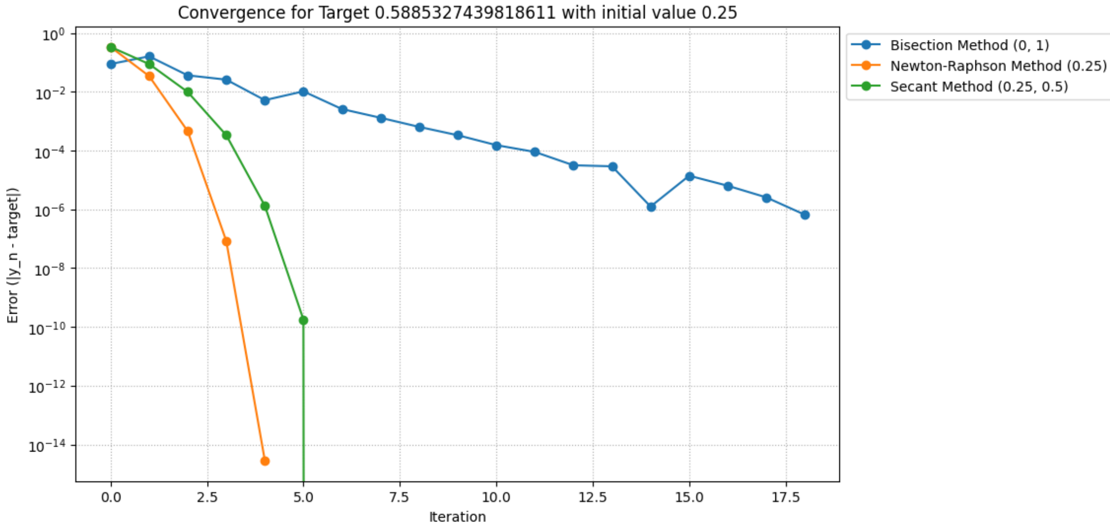
```python
plt.figure(figsize=(10, 6))
# Bisection method
bisection_results_1 = bisection_method(g, 0, 1)
plot_convergence(bisection_results_1, "Bisection Method (0, 1)", target)
# Newton-Raphson method
newton_results_1 = newton_raphson_method(g, dg, 0.25)
plot_convergence(newton_results_1, "Newton-Raphson Method (0.25)", target)
# Secant method
secant_results_1 = secant_method(g, 0.25, 0.5)
plot_convergence(secant_results_1, "Secant Method (0.25, 0.5)", target)
plt.yscale('log')
plt.xlabel("Iteration")
plt.ylabel("Error (|y_n - target|)")
plt.title("Convergence for Target 0.5671432904097838 with initial value 0.25")
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.grid(True, linestyle=':')
plt.show()

# Plot for initial value 0.75
plt.figure(figsize=(10, 6))
# Bisection method
bisection_results_2 = bisection_method(g, 0, 1)
plot_convergence(bisection_results_2, "Bisection Method (0, 1)", target)
# Newton-Raphson method
newton_results_2 = newton_raphson_method(g, dg, 0.75)
plot_convergence(newton_results_2, "Newton-Raphson Method (0.75)", target)
# Secant method
secant_results_2 = secant_method(g, 0.5, 0.75)
plot_convergence(secant_results_2, "Secant Method (0.5, 0.75)", target)
plt.yscale('log')
plt.xlabel("Iteration")
plt.ylabel("Error (|y_n - target|)")
plt.title("Convergence for Target 0.5671432904097838 with initial value 0.75")
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.grid(True, linestyle=':')
plt.show()
```

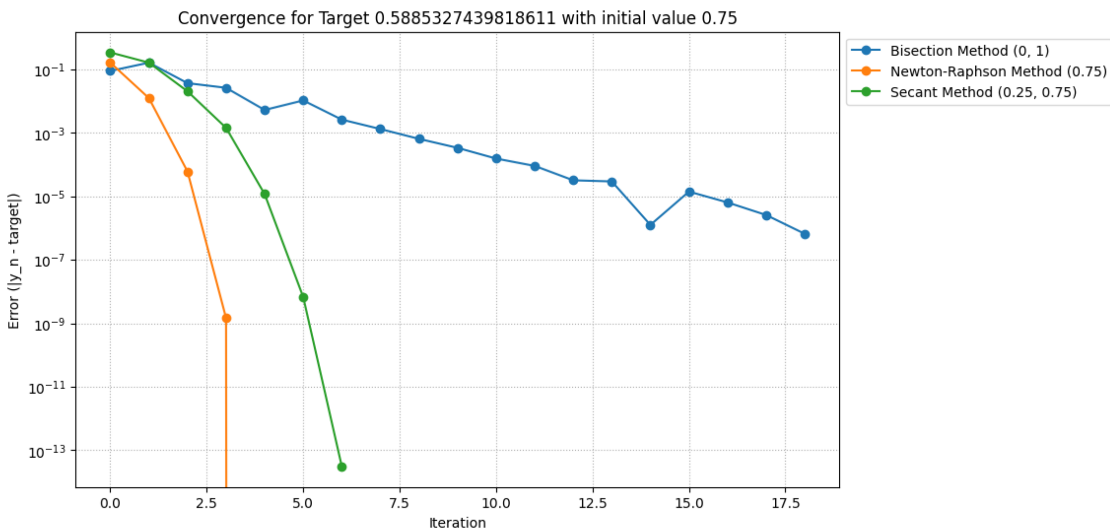




[ ]:
```
def g(y):
    return np.exp(-y) - np.sin(y)

# Derivative of g(y) needed for Newton-Raphson method
def dg(y):
    return -np.exp(-y) - np.cos(y)

# Define the target value
target = 0.5885327439818611

# Plot for initial value 0.25
```

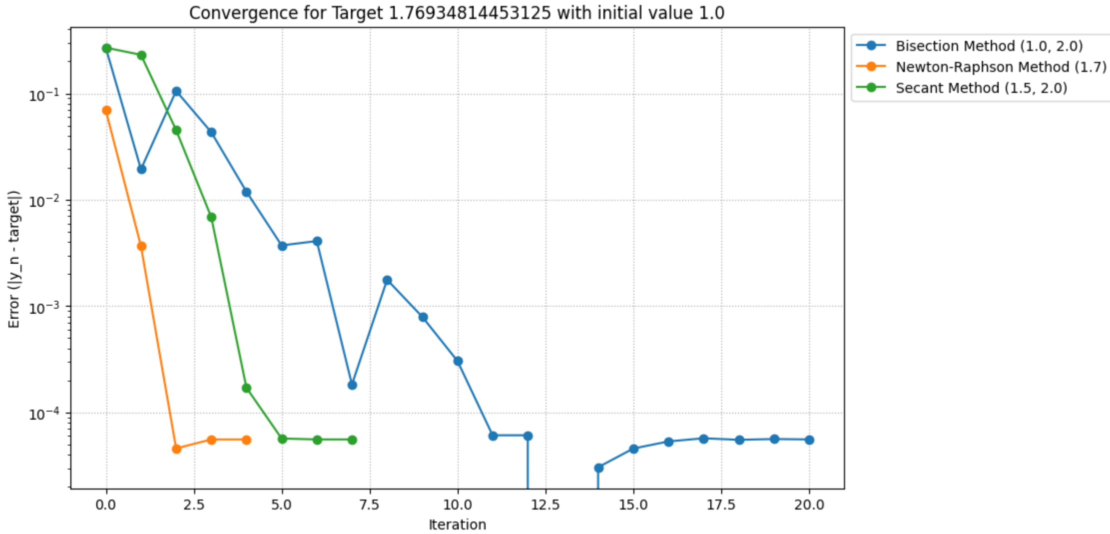
```python
plt.figure(figsize=(10, 6))
# Bisection method
bisection_results_1 = bisection_method(g, 0, 1)
plot_convergence(bisection_results_1, "Bisection Method (0, 1)", target)
# Newton-Raphson method
newton_results_1 = newton_raphson_method(g, dg, 0.25)
plot_convergence(newton_results_1, "Newton-Raphson Method (0.25)", target)
# Secant method
secant_results_1 = secant_method(g, 0.25, 0.5)
plot_convergence(secant_results_1, "Secant Method (0.25, 0.5)", target)
plt.yscale('log')
plt.xlabel("Iteration")
plt.ylabel("Error (|y_n - target|)")
plt.title("Convergence for Target 0.5885327439818611 with initial value 0.25")
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.grid(True, linestyle=':')
plt.show()

# Plot for initial value 0.75
plt.figure(figsize=(10, 6))
# Bisection method
bisection_results_2 = bisection_method(g, 0, 1)
plot_convergence(bisection_results_2, "Bisection Method (0, 1)", target)
# Newton-Raphson method
newton_results_2 = newton_raphson_method(g, dg, 0.75)
plot_convergence(newton_results_2, "Newton-Raphson Method (0.75)", target)
# Secant method
secant_results_2 = secant_method(g, 0.25, 0.75)
plot_convergence(secant_results_2, "Secant Method (0.25, 0.75)", target)
plt.yscale('log')
plt.xlabel("Iteration")
plt.ylabel("Error (|y_n - target|)")
plt.title("Convergence for Target 0.5885327439818611 with initial value 0.75")
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.grid(True, linestyle=':')
plt.show()
```

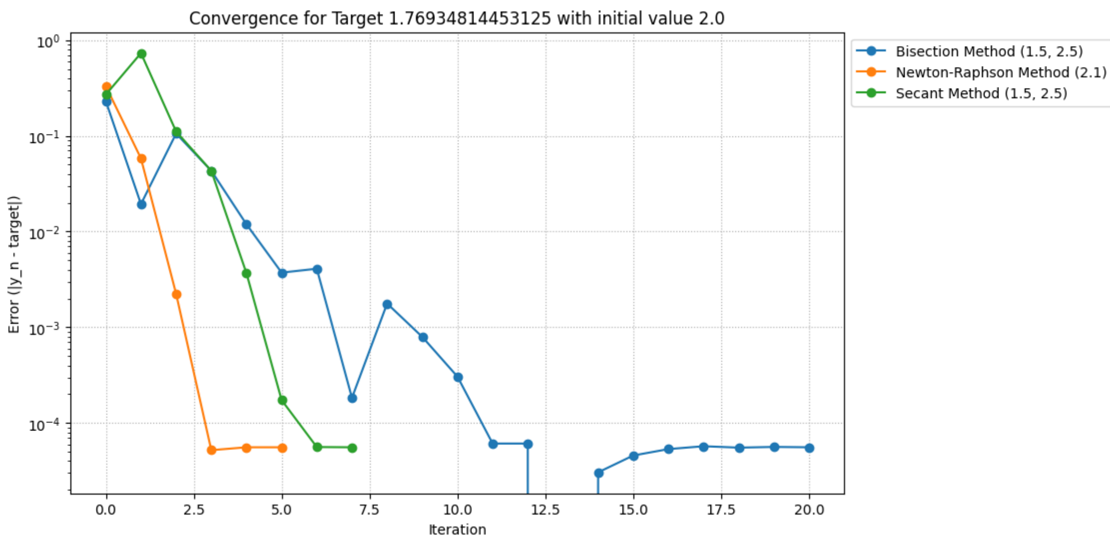




[ ]:
```
def g(y):
    return y**3 - 2*y - 2

# Derivative of g(y) needed for Newton-Raphson method
def dg(y):
    return 3*y**2 - 2

# Define the target value
target = 1.76934814453125

# Plot for initial value 1.0 (lesser than target)
```

```python
plt.figure(figsize=(10, 6))
# Bisection method
bisection_results_1 = bisection_method(g, 1.0, 2.0)
plot_convergence(bisection_results_1, "Bisection Method (1.0, 2.0)", target)
# Newton-Raphson method
newton_results_1 = newton_raphson_method(g, dg, 1.7)
plot_convergence(newton_results_1, "Newton-Raphson Method (1.7)", target)
# Secant method
secant_results_1 = secant_method(g, 1.5, 2.0)
plot_convergence(secant_results_1, "Secant Method (1.5, 2.0)", target)
plt.yscale('log')
plt.xlabel("Iteration")
plt.ylabel("Error (|y_n - target|)")
plt.title("Convergence for Target 1.76934814453125 with initial value 1.0")
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.grid(True, linestyle=':')
plt.show()

# Plot for initial value 2.0 (greater than target)
plt.figure(figsize=(10, 6))
# Bisection method
bisection_results_2 = bisection_method(g, 1.5, 2.5)
plot_convergence(bisection_results_2, "Bisection Method (1.5, 2.5)", target)
# Newton-Raphson method
newton_results_2 = newton_raphson_method(g, dg, 2.1)
plot_convergence(newton_results_2, "Newton-Raphson Method (2.1)", target)
# Secant method
secant_results_2 = secant_method(g, 1.5, 2.5)
plot_convergence(secant_results_2, "Secant Method (1.5, 2.5)", target)
plt.yscale('log')
plt.xlabel("Iteration")
plt.ylabel("Error (|y_n - target|)")
plt.title("Convergence for Target 1.76934814453125 with initial value 2.0")
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.grid(True, linestyle=':')
plt.show()
```

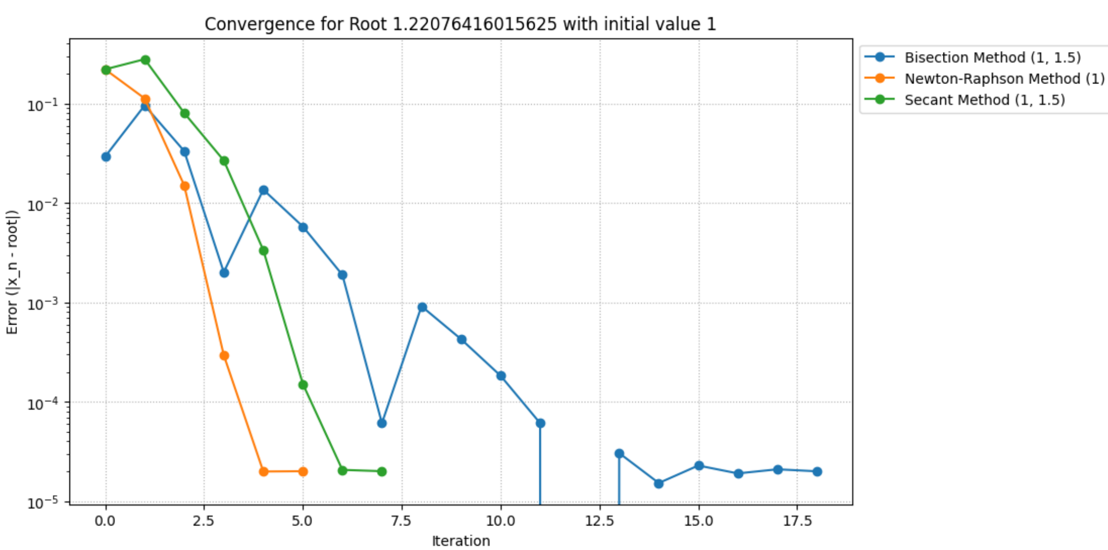


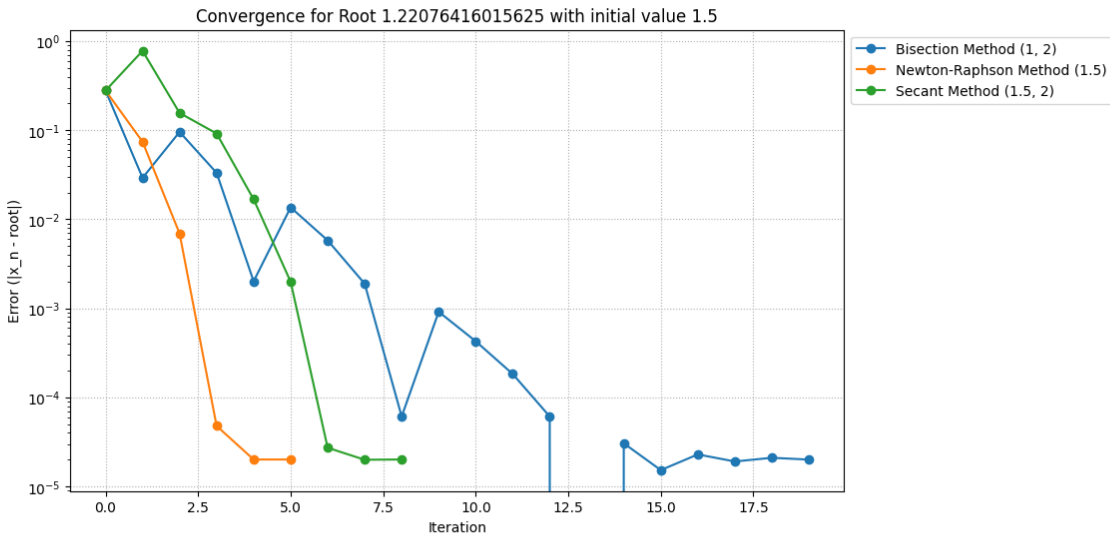


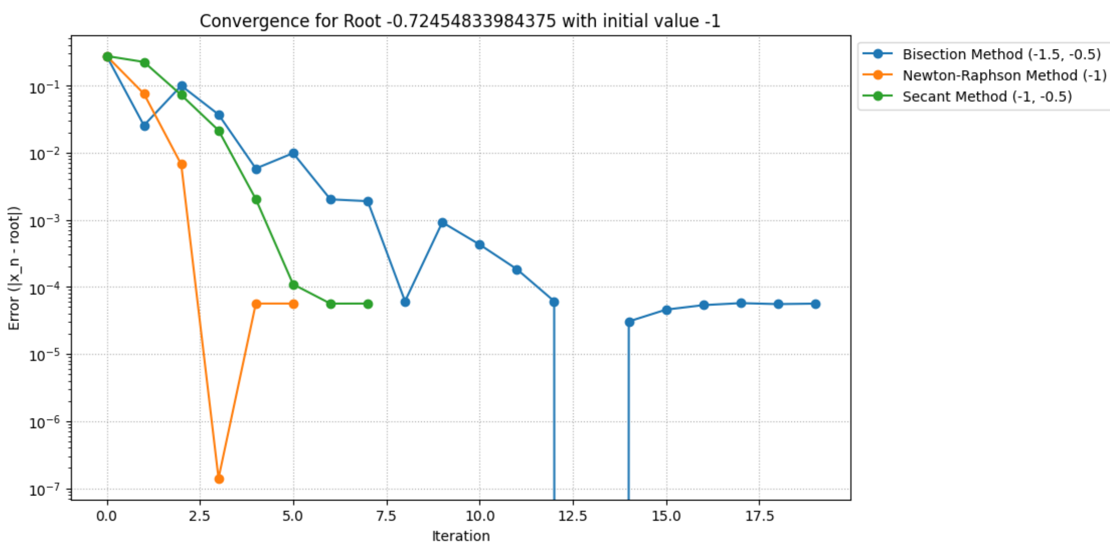
[ ]:
```
def f(x):
    return x**4 - x - 1

# Derivative of f(x) needed for Newton-Raphson method
def df(x):
    return 4*x**3 - 1

# Define the first root value
root1 = 1.22076416015625

# Plot for initial value 1 (less than root1)
```

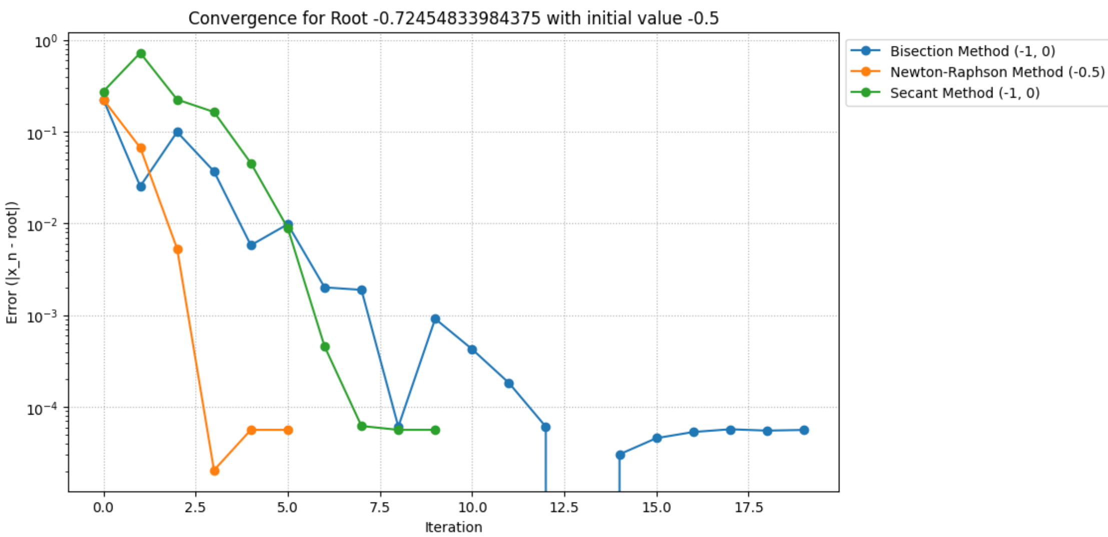
```
plt.figure(figsize=(10, 6))
# Bisection method
bisection_iters_1 = bisection_method(f, 1, 1.5)
plot_convergence(bisection_iters_1, "Bisection Method (1, 1.5)", root1)
# Newton-Raphson method
newton_iters_1 = newton_raphson_method(f, df, 1)
plot_convergence(newton_iters_1, "Newton-Raphson Method (1)", root1)
# Secant method
secant_iters_1 = secant_method(f, 1, 1.5)
plot_convergence(secant_iters_1, "Secant Method (1, 1.5)", root1)
plt.yscale('log')
plt.xlabel("Iteration")
plt.ylabel("Error (|x_n - root|)")
plt.title("Convergence for Root 1.22076416015625 with initial value 1")
plt.legend()
plt.legend(loc='upper left', bbox_to_anchor=(1,1))
plt.grid(True, linestyle=':')
plt.show()

# Plot for initial value 1.5 (greater than root1)
plt.figure(figsize=(10, 6))
# Bisection method
bisection_iters_2 = bisection_method(f, 1, 2)
plot_convergence(bisection_iters_2, "Bisection Method (1, 2)", root1)
# Newton-Raphson method
newton_iters_2 = newton_raphson_method(f, df, 1.5)
plot_convergence(newton_iters_2, "Newton-Raphson Method (1.5)", root1)
# Secant method
secant_iters_2 = secant_method(f, 1.5, 2)
plot_convergence(secant_iters_2, "Secant Method (1.5, 2)", root1)
plt.yscale('log')
plt.xlabel("Iteration")
plt.ylabel("Error (|x_n - root|)")
plt.title("Convergence for Root 1.22076416015625 with initial value 1.5")
plt.legend(loc='upper left', bbox_to_anchor=(1,1))
plt.grid(True, linestyle=':')
plt.show()

root2 = -0.72454833984375

# Plot for initial value -1 (less than root2)
plt.figure(figsize=(10, 6))
# Bisection method
bisection_iters_3 = bisection_method(f, -1.5, -0.5)
plot_convergence(bisection_iters_3, "Bisection Method (-1.5, -0.5)", root2)
# Newton-Raphson method
newton_iters_3 = newton_raphson_method(f, df, -1)
```

```python
plot_convergence(newton_iters_3, "Newton-Raphson Method (-1)", root2)
# Secant method
secant_iters_3 = secant_method(f, -1, -0.5)
plot_convergence(secant_iters_3, "Secant Method (-1, -0.5)", root2)
plt.yscale('log')
plt.xlabel("Iteration")
plt.ylabel("Error (|x_n - root|)")
plt.title("Convergence for Root -0.72454833984375 with initial value -1")
plt.legend(loc='upper left', bbox_to_anchor=(1,1))
plt.grid(True, linestyle=':')
plt.show()

# Plot for initial value -0.5 (greater than root2)
plt.figure(figsize=(10, 6))
# Bisection method
bisection_iters_4 = bisection_method(f, -1, 0)
plot_convergence(bisection_iters_4, "Bisection Method (-1, 0)", root2)
# Newton-Raphson method
newton_iters_4 = newton_raphson_method(f, df, -0.5)
plot_convergence(newton_iters_4, "Newton-Raphson Method (-0.5)", root2)
# Secant method
secant_iters_4 = secant_method(f, -1, 0)
plot_convergence(secant_iters_4, "Secant Method (-1, 0)", root2)
plt.yscale('log')
plt.xlabel("Iteration")
plt.ylabel("Error (|x_n - root|)")
plt.title("Convergence for Root -0.72454833984375 with initial value -0.5")
plt.legend(loc ='upper left',bbox_to_anchor = (1,1))
plt.grid(True,linestyle=':')
plt.show()
```




Convergence for Root 1.22076416015625 with initial value 1.5
Bisection Method (1, 2)
Newton-Raphson Method (1.5)
Secant Method (1.5, 2)
Error (|x_n - root|)
Iteration


Convergence for Root -0.72454833984375 with initial value -1
Bisection Method (-1.5, -0.5)
Newton-Raphson Method (-1)
Secant Method (-1, -0.5)
Error (|x_n - root|)
Iteration



Matrix Decomposition

[6]:
```python
import copy

def myUL(matrix):
    n = len(matrix)
    lowmat = copy.deepcopy(matrix)
    permute = np.eye(n)
    identity = np.eye(n)
    operation = identity
    for i in range(n-1, -1, -1):
        if lowmat[i][i] == 0:
            print('enter')
            j = i - 1
            while j > -1 and lowmat[j][i] == 0:
                j -= 1
            if j == -1:
                continue
            else:
                permute[[i, j]] = permute[[j, i]]
                lowmat[[i,j]] = lowmat[[j,i]]
        op = copy.deepcopy(identity)
        for j in range(i - 1, -1, -1):
            m = lowmat[j][i] / lowmat[i][i]
            op[j][i] = -m
            lowmat[j][i] = 0
            for k in range(i - 1, -1, -1):
                x1 = m * lowmat[i][k]
                x2 = lowmat[j][k]
```

```python
                lowmat[j][k] = x2 - x1
        print(i,op)
        operation = np.dot(op,operation)
    upper = get_inv(operation)
    return upper,lowmat,permute
def get_inv(matrix):
    matinv = np.linalg.inv(matrix)
    return matinv

def printList(lst):
    for a in lst:
        print(a)
```

[7]:

```python
mat = np.array([[0,1,2],[3,4,5],[6,7,9]],dtype = float)
upper,lower,permute = myUL(mat)
print('Upper Triangular :')
printList(upper)
print('Lower Triangular :')
printList(lower)
print('Permutation Matrix :')
printList(permute)
print('PA = UL')
printList((np.dot(upper,lower)))
```

```
2 [[ 1.          0.         -0.22222222]
 [ 0.          1.         -0.55555556]
 [ 0.          0.          1.        ]]
1 [[1. 5. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
0 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
Upper Triangular :
[ 1.          -5.          0.22222222]
[0.          1.          0.55555556]
[0. 0. 1.]
Lower Triangular :
[-3.  0.  0.]
[-0.33333333  0.11111111  0.        ]
[6. 7. 9.]
Permutation Matrix :
[1. 0. 0.]
[0. 1. 0.]
[0. 0. 1.]
PA = UL
[-2.22044605e-15  1.00000000e+00  2.00000000e+00]
```

```
[3. 4. 5.]
[6. 7. 9.]
```

Gaussian Elimination

[1]:
```python
import numpy as np

def Gaussian_Elimination(A, B):
    n = len(A)
    for i in range(n):
        for j in range(i+1, n):
            factor = A[j][i]/A[i][i]
            for k in range(i, n):
                A[j][k] -= factor*A[i][k]
            B[j] -= factor*B[i]
    M = np.zeros(n)
    for i in range(n-1, -1, -1):
        M[i] = B[i]
        for j in range(i+1, n):
            M[i] -= A[i][j]*M[j]
        M[i] /= A[i][i]
        if abs(M[i]) < 1e-10:
            M[i] = 0.0
    return M
```

[2]:
```python
A = [[1,2,1],[2,2,3],[-1,-3,0]]
B = [0,3,2]
M = Gaussian_Elimination(A,B)
print("\n Solution of the System of Equations: ")
print(M)
```

```
 Solution of the System of Equations:
[ 1. -1.  1.]
```

[3]:
```python
A = [[4,3,2,1],[3,4,3,2],[2,3,4,3],[1,2,3,4]]
B = [1,1,-1,-1]
M = Gaussian_Elimination(A,B)
print("\n Solution of the System of Equations: ")
print(M)
```

```
 Solution of the System of Equations:
[ 0.  1. -1.  0.]
```