Root-Finding Methods

Wanzhu Zheng

April 2023

1 Problem Description

This report demonstrates the different root-finding methods and how they are used to calculate roots to equations. Given two equations,

$$f_1(x) = f_1(x) = \frac{1}{2} sin((x-1)^3) \tag{1}$$

$$f_2(x) = 6^{-x} - 2 \tag{2}$$

approximate their roots to at least $1e^{-8}$ accuracy using Bisection method, Fixed Point Method, Newton's Method, and Secant Method.

The Bisection method is represented mathematically as:

$$x_{n+1}(x) = \frac{a_n + b_n}{2}) \tag{3}$$

where x_{n+1} is the midpoint and approximation of the root in the interval $[a_n, b_n]$. Then, $f(x_{n+1})$ is evaluated at every midpoint until the midpoints is less than our tolerance error.

The Fixed Point Method is represented mathematically as:

$$x_{n+1} = g(x_n) \tag{4}$$

where x_{n+1} is the approximation of the root, x_n is the previous approximation and $g(x_n)$ is the value of the function at each approximation. Sometimes, the Fixed Point Method does not converge for certain functions, so you need to manipulate g such that it converges.

The Newton-Raphson's Method is represented mathematically as:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{5}$$

where x_{n+1} is the approximation of the root, $x_n$is previous approximation, $f(x_n)$ is the values of the function at each approximation $f'(x_n)$ is the values of the derivative function at each approximation.

The Secant Method is represented mathematically as:

$$x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n - f(x_{n-1})} \tag{6}$$

where x_{n+1} is the approximation of the root, x_n and x_{n-1} are previous approximations, and $f(x_n)$ and $f(x_{n-1})$ are the values of the function at each approximation.

2 Results

Here we demonstrate the results of the four root-finding methods. First, we plot the two equations as seen in Figure 1. As expected, f_1) is sinusoidal and f_2 is a hyperbolic curve. Next, we show the convergence of each function by plotting absolute error against each iteration. Figure 2 shows the convergence for f_1 and the convergence of f_2. Since we want our tolerance error to be less than $1e^{-8}$, we see that all absolute errors tend to 0.0 on the graphs. This shows that f_1) and f_2) converge because it implies that the error between the real root and the approximated root is less than our tolerance error. Finally, we plot convergence of each function again, but for log-log plots. The log-log plot shows the logarithm of values of the variables. From these two plots (Figure 3), we observe the order of convergence for each function.The chosen initial points are listed below:
Bisection Method

- f_1: Choose $a = -0.3$, $b = 2$
- f_2: Choose $a = -2$, $b = 1$

Fixed Point Method:

- f_1: Choose $p_0 = 0$
- f_2: Choose $p_0 = -2$

Newton-Raphson's Method:

- f_1: Choose $p_0 = -0.3$
- f_2: Choose $p_0 = -5$

Secant Method:

- f_1: Choose $p_0 = -0.3$, $p_1 = 2$
- f_2: Choose $p_0 = -2$, $p_1 = 1$

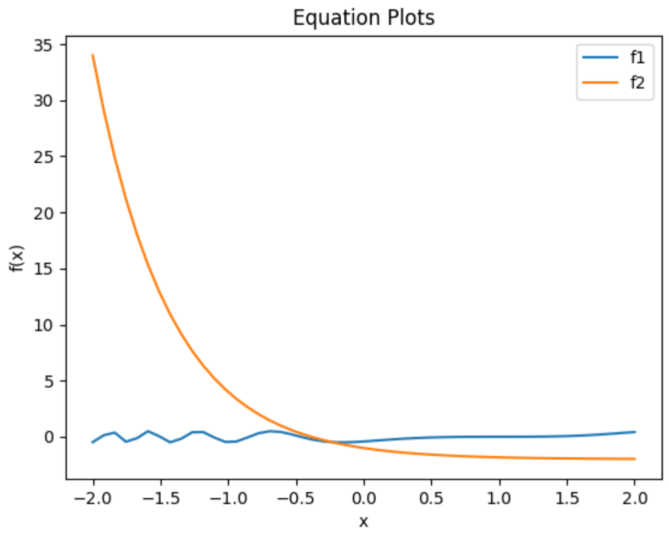
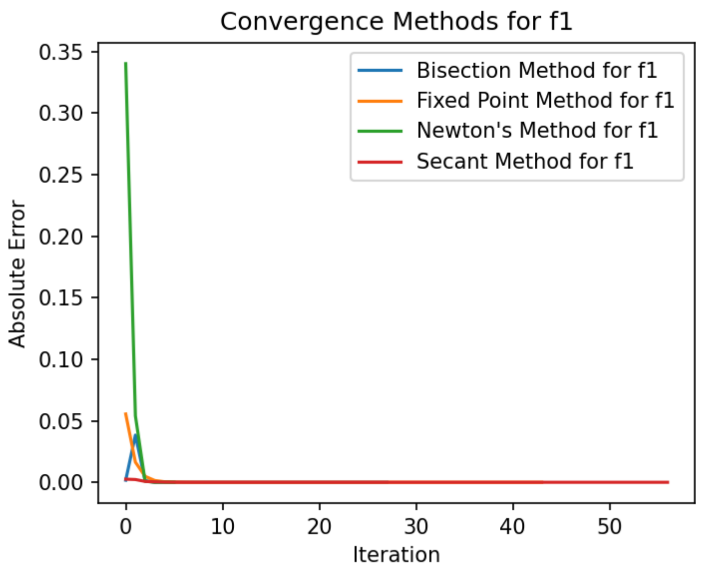


Figure 1. Plot of f_1 and f_2



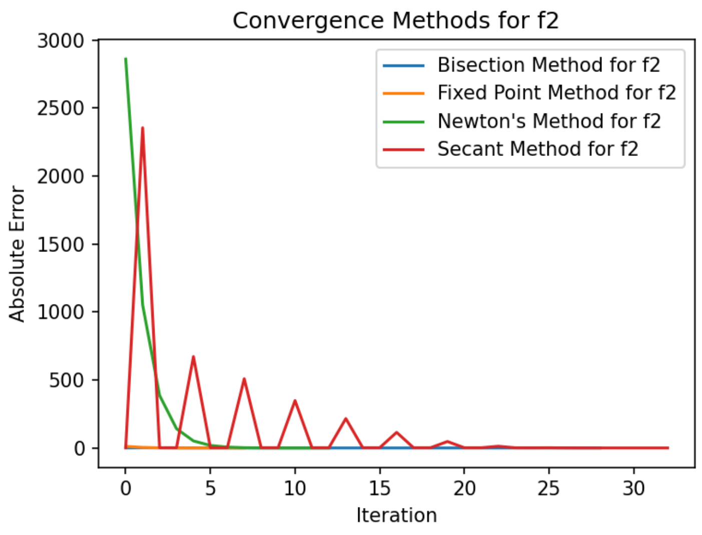
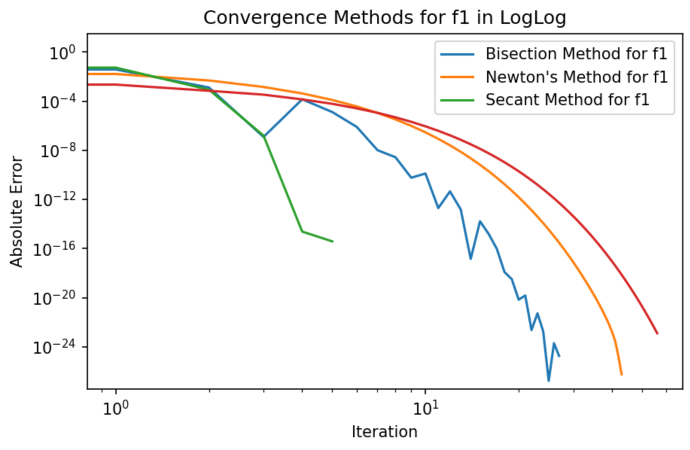


Figure 2. Top: Convergence plot of f_1; Bottom: Convergence plot of f_2



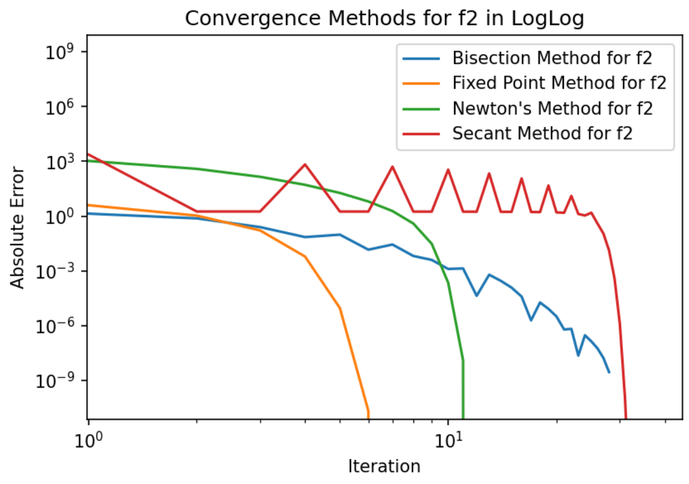


Figure 3. Top: Log-Log plot of f_1; Bottom: Log-Log plot of f_2

3 Appendix

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


def my_fun(x, type):
    if type == 0:
        f = (1/2)*np.sin((x-1)**3)
    if type == 1:
        f = 6**(-x)-2
    return f


def my_fun_deriv(x, n, h=1e-8):
    # argument n is type number
    der = (my_fun(x+h, n)-my_fun(x, n))/h
    return der


# show plots for equations
val = np.linspace(-2, 2)
plt.plot(val, my_fun(val, 0))
plt.plot(val, my_fun(val, 1))
plt.title("Equation Plots")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend(["f1", "f2"], loc ="upper right")
plt.show()


def bisection(a, b, n, e, max):
    i = 1
    fa = my_fun(a, n)
    pn = []
    abs_err_array = []
    while i <= max:
        p = a+((b-a)/2)
        abs_err = abs(my_fun(p, n))
        abs_err_array.append(abs_err)
        pn.append(p)
        fp = my_fun(p, n)
        if fp == 0 or (b-a)/2 < e:
            return [p, pn, abs_err_array]
        i += 1
        if fa*fp > 0:
            a = p
            fa = fp
        else:
            b = p


def fixed_point(p0, n, e, max):
    i = 1
    pn = []
    abs_err_array = []
    while i <= max:
        # updated p because original function does not converge by fixed point.
        p = p0 - my_fun(p0, n)/my_fun_deriv(p0, n)
        pn.append(p)
        abs_err = abs(my_fun(p, n))
        abs_err_array.append(abs_err)
        pn.append(p)
        if abs(p - p0) < e:
            return [p, pn, abs_err_array]
        i += 1
        p0 = p


def newton(p0, n, e, max):
    i = 1
    pn = []
    abs_err_array = []
    while i <= max:
        p = p0 - my_fun(p0, n)/my_fun_deriv(p0, n)
        pn.append(p)
        abs_err = abs(my_fun(p, n))
        abs_err_array.append(abs_err)
        pn.append(p)
        if abs(p-p0) < e:
            return [p, pn, abs_err_array]
        i += 1
        p0 = p


def secant(p0, p1, n, e, max):
    i = 2
    pn = []
    abs_err_array = []
    q0 = my_fun(p0, n)
    q1 = my_fun(p1, n)
    while i <= max:
        p = p1-(q1*(p1-p0)/(q1-q0))
        pn.append(p)
        abs_err = abs(my_fun(p, n))
        abs_err_array.append(abs_err)
        pn.append(p)
        if abs(p-p1) < e:
            return [p, pn, abs_err_array]
        i += 1
        p0 = p1
        q0 = q1
        p1 = p
        q1 = my_fun(p, n)


# show convergence plots for f1 and f2
_, _, bis_err0 = bisection(-0.3, 2, 0, 1e-8, 1000)
_, _, fix_err0 = fixed_point(0, 0, 1e-8, 1000)
_, _, new_err0 = newton(-0.3, 0, 1e-8, 1000)
_, _, sec_err0 = secant(-0.3, 2, 0, 1e-8, 1000)
plt.figure(dpi=150)
x1 = np.arange(len(bis_err0))
x2 = np.arange(len(fix_err0))
x3 = np.arange(len(new_err0))
x4 = np.arange(len(sec_err0))

# original plots
plt.plot(x1, bis_err0)
plt.plot(x2, fix_err0)
plt.plot(x3, new_err0)
plt.plot(x4, sec_err0)
plt.legend(["Bisection Method for f1", "Fixed Point Method for f1",
            "Newton's Method for f1", "Secant Method for f1"], loc="upper right")
plt.title("Convergence Methods for f1")
plt.xlabel("Iteration")
plt.ylabel("Absolute Error")
plt.show()

# loglog plots
plt.figure(dpi=150)
plt.loglog(x1, bis_err0)
plt.loglog(x2, fix_err0)
plt.loglog(x3, new_err0)
plt.loglog(x4, sec_err0)
plt.legend(["Bisection Method for f1", "Newton's Method for f1", "Secant Method for f1"], loc="upper ri
plt.title("Convergence Methods for f1 in LogLog")
plt.xlabel("Iteration")
plt.ylabel("Absolute Error")
plt.show()


_, _, bis_err1 = bisection(-2, 1, 1, 1e-8, 1000)
```

```
_, _, fix_err1 = fixed_point(-2, 1, 1e-8, 1000)
_, _, new_err1 = newton(-5, 1, 1e-8, 1000)
_, _, sec_err1 = secant(-2, 1, 1, 1e-8, 1000)
plt.figure(dpi=150)
x1 = np.arange(len(bis_err1))
x2 = np.arange(len(fix_err1))
x3 = np.arange(len(new_err1))
x4 = np.arange(len(sec_err1))

# original plots
plt.plot(x1, bis_err1)
plt.plot(x2, fix_err1)
plt.plot(x3, new_err1)
plt.plot(x4, sec_err1)
plt.legend(["Bisection Method for f2", "Fixed Point Method for f2",
            "Newton's Method for f2", "Secant Method for f2"], loc="upper right")
plt.title("Convergence Methods for f2")
plt.xlabel("Iteration")
plt.ylabel("Absolute Error")
plt.show()

# loglog plots
plt.figure(dpi=150)
plt.loglog(x1, bis_err1)
plt.loglog(x2, fix_err1)
plt.loglog(x3, new_err1)
plt.loglog(x4, sec_err1)
plt.legend(["Bisection Method for f2", "Fixed Point Method for f2",
            "Newton's Method for f2", "Secant Method for f2"], loc="upper right")
plt.title("Convergence Methods for f2 in LogLog")
plt.xlabel("Iteration")
plt.ylabel("Absolute Error")
plt.show()
```