Computional Homework10 Report

Student: 纪浩正, jihz2023@mail.sustech.edu.cn

1 Bisection Method

1.1 Introduction

The Bisection Method is a numerical technique for finding a root of a continuous function $f(x) = 0$ in a given interval $[a, b]$, provided that $f(a)f(b) < 0$. This guarantees that there is at least one root between a and b by the Intermediate Value Theorem.

In the implemented Python class `BisectionMethod`, the method works as follows:

1. Compute the midpoint of the interval:

$$m = \frac{a+b}{2}$$

   and store it in `self.points` for tracking iterations.

2. Check if the interval is sufficiently small:

$$|b-a| < 2 \cdot \text{tol}$$

3. Determine in which subinterval the root lies:

$$\text{if } f(a)f(m) < 0, \text{ then the root is in } [a,m] \text{else if } f(m)f(b) < 0, \text{ then the root is in } [m,b]$$

4. Repeat the above steps recursively until the interval satisfies the tolerance condition.

```python
class BisectionMethod:
    def __init__(self,f,tol):
        self.f=f
        self.tol=tol
        self.points=[]
        self.roots=[]
```

```python
def _bi(self,a,b):
    m=(a+b)/2
    self.points.append(m)
    if (b-a)<2*self.tol:
        if self.f(a)*self.f(b)<0:
            self.roots.append(m)
            return m
        else:
            raise ValueError("please change the interval")
    else:
        if abs(self.f(m))<self.tol:
            self.roots.append(m)
            return m
        elif self.f(a)*self.f(m)<0:
            return self._bi(a,m)
        elif self.f(m)*self.f(b)<0:
            return self._bi(m,b)
        else:
            raise ValueError("please change the interval")

def bisection(self,a,b):
    self.points=[]
    self.roots=[]
    if self.f(a)==0:
        self.roots.append(a)
    if self.f(b)==0:
        self.roots.append(b)
    if self.f(a)*self.f(b)<0:
        self._bi(a,b)
        return np.array(sorted(self.roots)), np.array(self.points)
    else:
        raise ValueError("please change the interval")
```

1.2 Result

The following table summarizes the iterations of the Bisection Method applied to $f(x) = x^3 - e^{-x}$:

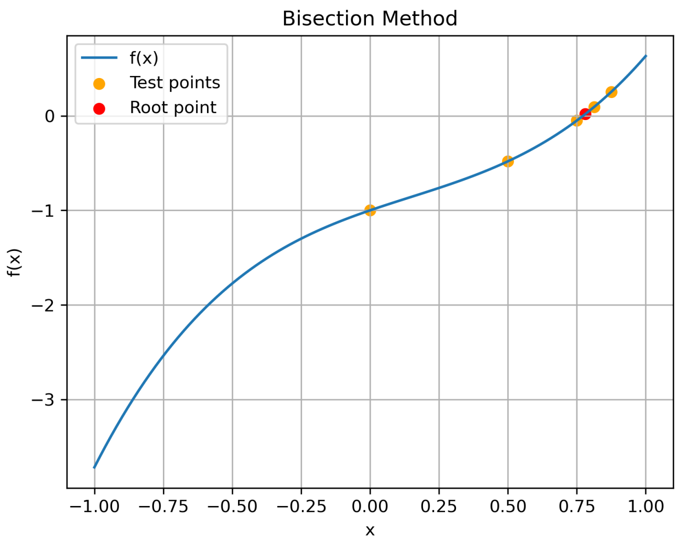

Figure 1 Visualization of the Bisection Method iterations, showing the convergence of test points to the final root.

Iteration	x_n	$f(x_n)$
0	0.000000	-1.000000e+00
1	0.500000	-4.815307e-01
2	0.750000	-5.049155e-02
3	0.875000	2.530599e-01
4	0.812500	9.262964e-02
5	0.781250	1.900380e-02
Final	0.765625	-1.624787e-02

Table 1 Iteration results for the Bisection Method. The final approximation of the root is $x \approx 0.765625$ with $f(x) \approx -1.624787 \times 10^{-2}$.

```
def bisection_method():
    def f(x):
        return x**3-np.exp(-x)
    S=BisectionMethod(f,0.02)
    r,n =S.bisection(-1,1)
    print("############# Bisection Method #############")
    for count in range(len(n)):
```

```
    print(f"Iteration {count}: x = {n[count]:.6f}, f(x)={f(n[count]):.6e}")
  print(f"Final fixed point:{r[0]:.6f} with value {f(r[0]):.6e}")
```

2 Fixed-point Method

2.1 Introduction

The Fixed-point Method is an iterative technique used to find a solution x^* to an equation $f(x) = 0$. In this homework, the original equation is:

$$f(x) = x^3 - e^{-x} = 0.$$

To apply the Fixed-point Method, we rewrite it in the form:

$$x = f(x) + x = g(x),$$

so that the iteration can be written as:

$$x_{n+1} = \varphi(x_n) = x_n + f(x_n).$$

For better convergence, we can transform the original equation into a different fixed-point form. For example:

$$x^3 - e^{-x} = 0 \quad \text{can be rewritten as} \quad e^{-x/3} - x = 0.$$

In the iteration, we can define:

$$f(x) = e^{-x/3} - x, \quad f'(x) = -\frac{1}{3}e^{-x/3} - 1.$$

A critical requirement for convergence is the Lipschitz (Lei) condition:

$$|f'(x) + 1| < 1,$$

which ensures that the iterative process will converge. If this condition is not satisfied, the iteration may diverge. By choosing a suitable transformation $\varphi(x)$, we can reduce $|\varphi'(x)|$ near the root and satisfy the Lipschitz condition.

```
class FixedPointMethod():
  def __init__(self,f,f1,x0,tol):
```

```
        self.f=f
        self.f1=f1
        self.x0=x0
        self.tol=tol
        self.points=[]

    def _fixed_point(self,x0,count=0):
        print(f"Interation {count}:x = {x0:.6f}, f(x0)={self.f(x0):.6e}")
        if abs((self.f1(x0)+1))>1:
            raise ValueError("enter the correct condition to satisfy the lipschitz condition")
        x_new=x0+self.f(x0)
        self.points.append(x_new)
        if abs(x_new-x0)<self.tol:
            return x_new
        else:
            return self._fixed_point(x_new,count+1)

    def fixed_point(self):
        x0=self.x0
        self.points=[]
        return self._fixed_point(x0),np.array(sorted(self.points))
```

2.2 Result

The iteration results of the Fixed-point Method are summarized in Table 2. The final fixed point is $x^* = 0.773854$ with function value $f(x^*) = -1.221010 \times 10^{-3}$.

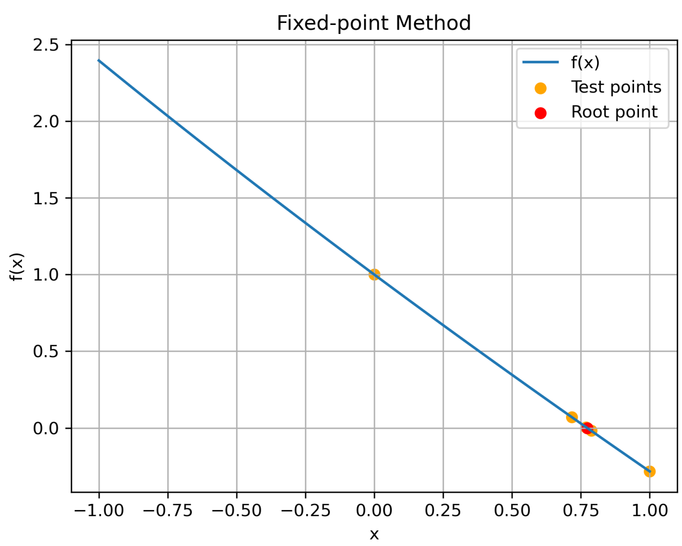

Figure 2 **Visualization of the Fixed-point Method iterations, showing the convergence of the iteration points towards the final root.**

Table 2 Fixed-point Method Iteration Results

Iteration	x_n	$f(x_n)$
0	100.000000	-1.000000e+02
1	0.000000	1.000000e+00
2	1.000000	-2.834687e-01
3	0.716531	7.100660e-02
4	0.787538	-1.842126e-02
5	0.769117	4.737230e-03
Final	0.773854	-1.221010e-03

```
def fixed_point_method():
    def f(x):
        return np.exp(-x/3)-x
    def f1(x):
        return -1/3*np.exp(-x/3)-1
    x0,tol=100,1e-2
    print("\n############# Fixed-point Method #############")
    S=FixedPointMethod(f,f1,x0,tol)
    r,n=S.fixed_point()
    print(f"Final fixed point:{r:.6f} with value {f(r):.6e}")
```

3 Newton Method

3.1 Introduction

The Newton Method is an iterative technique to find a root x^* of a function $f(x) = 0$. The method uses both the function and its derivative $f'(x)$ to iteratively improve the approximation of the root.

Starting from an initial guess x_0, the iteration formula is:

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

In the provided Python implementation (`Newton_method`), all intermediate points are stored in the `points` list for visualization and analysis. The convergence check is based on the change of successive iterates:

$$|x_{n+1} - x_n| < \text{tol}.$$

```python
def Newton_method(f,f1,x0,tol,points=None,count=0):
    if points is None:
        points = []
    points.append(x0)
    print(f"Interation {count}:x = {x0:.6f}, f(x)={f(x0):.6e}")

    x_new=x0-f(x0)/f1(x0)
    if abs(x_new-x0)<tol:
        points.append(x_new)
        print(f"Final fixed point: {x_new:.6f}, f(x)={f(x_new):.6e}")
        return x_new,np.array(points)
    else:
        return Newton_method(f,f1,x_new,tol,points,count+1)
```

3.2 Result

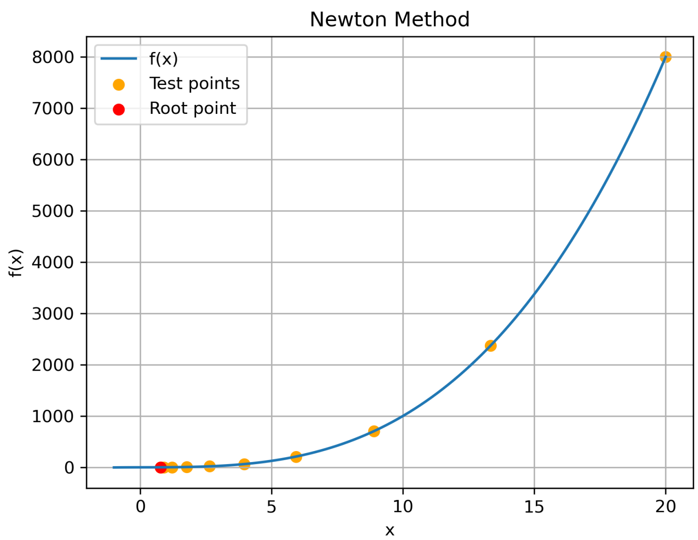

Figure 3 **Convergence of the Newton Method. The orange points indicate the iteration steps, and the red point is the final root.**

Table 3 Iteration results for the Newton Method. The final approximation of the root is $x \approx 0.772883$ with $f(x) \approx 3.759 \times 10^{-8}$.

Iteration	x_n	$f(x_n)$
0	20.000000	8.000000×10^3
1	13.333333	2.370370×10^3
2	8.888889	7.023318×10^2
3	5.925928	2.080959×10^2
4	3.950694	6.164313×10^1
5	2.634748	1.821841×10^1
6	1.762946	5.307662
7	1.203979	1.445248
8	0.893086	3.029361×10^{-1}
9	0.784979	2.756830×10^{-2}
10	0.773017	3.024780×10^{-4}
Final	0.772883	3.759421×10^{-8}

```
def newton_method():
    def f(x):
        return x**3-np.exp(-x)
    def f1(x):
        return 3*x**2+np.exp(-x)

    x0,tol=20,1e-2
    print("\n############ Newton Method ############")
    r,n=Newton_method(f,f1,x0,tol)
```

4 Secant Method

4.1 Introduction

The Secant Method is an iterative root-finding algorithm that uses two initial guesses x_0 and x_1 to approximate a root of $f(x) = 0$. Unlike the Newton Method, it does not require the derivative of $f(x)$, and instead approximates it using the slope of the line connecting $(x_0, f(x_0))$ and $(x_1, f(x_1))$:

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

The iteration continues until convergence is reached, i.e., $|x_{n+1} - x_n| < \text{tol}$.

```
def Secant_method(f,x0,x1,tol,points=None,count=0):
    if points is None:
        points=[]
    points.append(x1)
    print(f"Interation {count}:x = {x0:.6f}, f(x)={f(x0):.6e}")

    x_new=x1-(x1-x0)*f(x1)/(f(x1)-f(x0))
    if abs(x_new-x1)<tol:
        points.append(x_new)
        print(f"Final fixed point: {x_new:.6f}, f(x)={f(x_new):.6e}")
        return x_new,np.array(points)
    else:
        return Secant_method(f,x1,x_new,tol,points,count+1)
```

4.2 Result

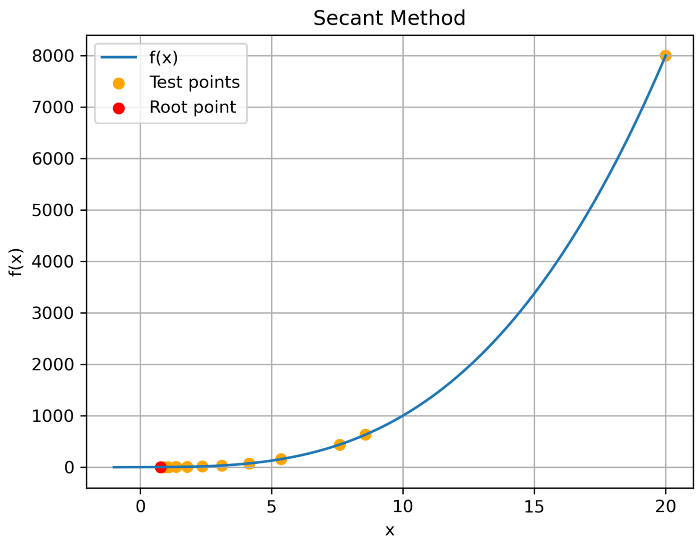

Figure 4 Convergence of the Secant Method. Orange points show iteration steps and the red point indicates the final root.

Table 4 Iteration results for the Secant Method. The final approximation of the root is $x \approx 0.772937$ with $f(x) \approx 1.215 \times 10^{-4}$.

Iteration	x_n	$f(x_n)$
0	10.000000	1.000000×10^3
1	20.000000	8.000000×10^3
2	8.571429	6.297374×10^2
3	7.594937	4.380988×10^2
4	5.362612	1.542112×10^2
5	4.149986	7.145688×10^1
6	3.102905	2.982991×10^1
7	2.352566	1.292531×10^1
8	1.778855	5.460040
9	1.359246	2.254423
10	1.064148	8.600296×10^{-1}
11	0.882137	2.725524×10^{-1}
12	0.797696	5.722401×10^{-2}
Final	0.772937	1.215491×10^{-4}