数值分析上机报告

第二章

院 (系) 名 称：　微电子学院

学 生 姓 名：　周玉乾

学　　　　号：　220205764

二〇二〇年十一月

第二章

一、　问题

试值法或者 Newton 法同二分法结合

问题 1

3.2.4　试值法(The Method of False Position)

由于二分法的收敛速度相对较慢,因此有些方法**尝试对它进行改进**.

二分法选择区间的中点进行下一次迭代,而所谓试值法选择

$$\big(a, f(a)\big),\ \big(b, f(b)\big)$$

的连线同 x 轴的交点的横坐标作为下一个迭代点.

理论分析

假设 $f(a) \cdot f(b) < 0$.经过点 $\big(a, f(a)\big)$ 与 $\big(b, f(b)\big)$ 的直线方程为:

$$y = \frac{f(b) - f(a)}{b - a}(x - b) + f(b),$$

令 $y = 0$, 求出

$$c = b - \frac{b - a}{f(b) - f(a)} \cdot f(b).$$

接下来有**三种可能性**:

- $f(c)$ 与 $f(a)$ 符号相反,则下一个有根区间为 $[a, c]$.
- $f(c)$ 与 $f(b)$ 符号相反,则下一个有根区间为 $[c, b]$.
- $f(c) = 0$,计算结束.

第三种情况直接得结果,否则根区间得到压缩.同二分法类似,可以构造一个 $\{[a_n, b_n]\}$ 的序列,其中每个区间都包含零点,零点 x^* 的近似值选为:

$$c_n = b_n - \frac{b_n - a_n}{f(b_n) - f(a_n)} \cdot f(b_n).$$

如果 $f(x)$ 是连续函数,可以证明这个算法一定收敛.若 $f(x)$ 是线性的,该方法一步就得到根.但是,有时候该方法的收敛速度甚至比二分法还要慢.二分法的有根区间的长度 $b_n - a_n$ 趋近于 0,试值法里 $b_n - a_n$ 会越来越小,但可能不趋近于 0.因此,**该方法的终止判据应选择** $|f(c_n)| \leq \epsilon$.

问题 2

Newton法同二分法的结合

为了提升Newton法的稳定性,减少初值对它的影响,可以把它与二分法相结合.具体的策略是:

- 假设 $a < b,\ f(a)f(b) < 0$,从 $x = a$ 或者 $x = b$ 开始迭代.
- 如果
$$\bar{x} = x - \frac{f(x)}{f'(x)} \in (a, b),$$
接受它,否则取 $\bar{x} = \dfrac{a+b}{2}$.
- 根据函数值的正负号,选取 $[a, \bar{x}]$ 或者 $[\bar{x}, b]$ 作为新的有根区间.
- 重复前面的过程,并在 $|f(\bar{x})|$ 足够小时终止迭代.

二、 分析

试值法流程

图2.1 为试值法算法流程图。

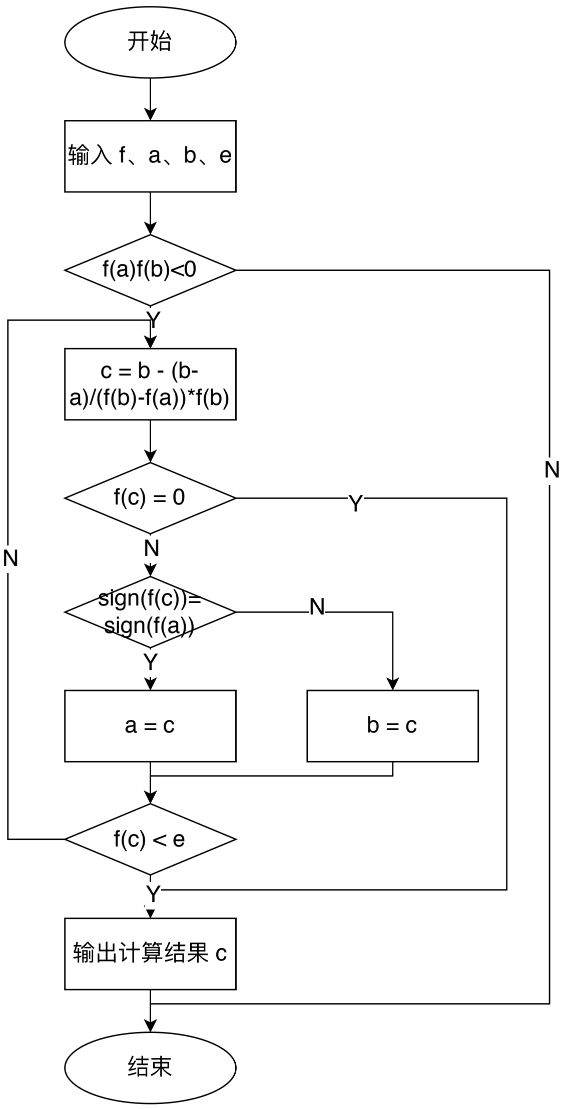

图 2.1　试值法流程图

Newton 法同二分法结合流程

图2.2 为试值法算法流程图。

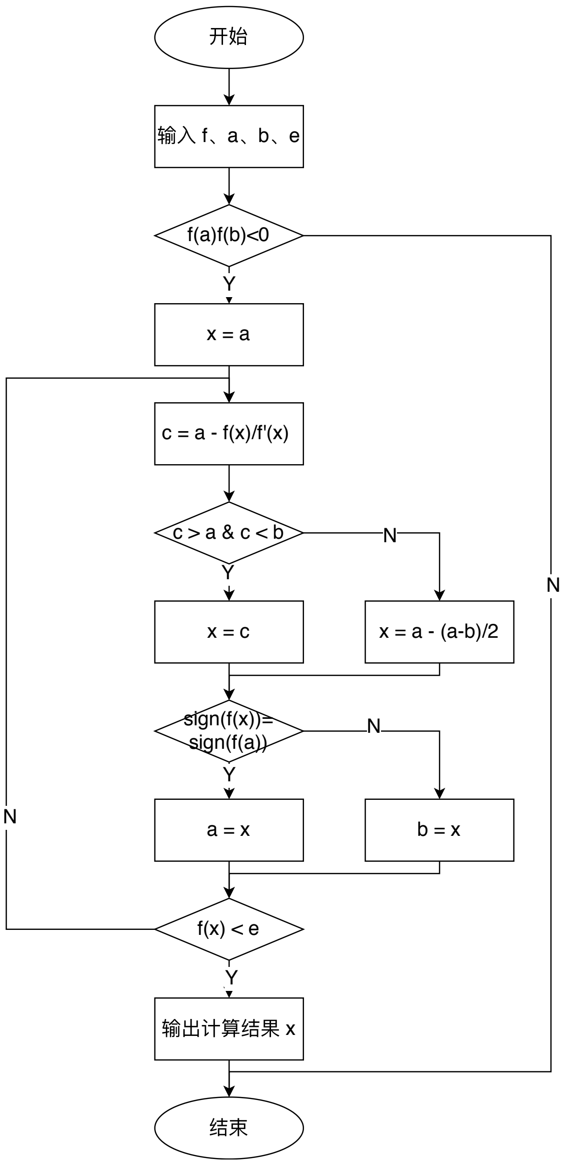

图 2.2　Newton 法同二分法结合流程

三、　程序

试值法

```
def TrailValue(expr, a, b, e):
    """
    试值法
    F@函数
    A@区间下限
    B@区间上限
    E@容忍误差限
    """
    f = _func(expr)
```

```python
    fa_0 = f.value(a)
    fb_0 = f.value(b)
    res = 0
    count = 0
    if abs(fa_0) < e:
        res = a
    elif abs(fb_0) < e:
        res = b
    elif sympy.sign(fa_0) == sympy.sign(fb_0):
        print('f(a) and f(b) 同号')
        sys.exit()
    else:
        while True:
            count = count + 1
            fa = f.value(a)
            fb = f.value(b)
            c = b - ((b-a)/(fb - fa))*fb
            fc = f.value(c)

            # 更新有根区间
            if sympy.sign(fa) == sympy.sign(fc):
                a = c
            else:
                b = c

            # 判断计算结束
            if abs(f.value(c)) < e:
                res = c
                break

    return res, count
```

Newton 法同二分法结合

```python
def Newton(expr, a, b, e):
    """
    牛顿法与二分法结合
    F@函数
    A@区间下限
    B@区间上限
    E@容忍误差限
    """
    f = _func(expr)
    fa_0 = f.value(a)
    fb_0 = f.value(b)
    res = 0
    count = 0
    if abs(fa_0) < e:
```

```python
            res = a
        elif abs(fb_0) < e:
            res = b
        elif sympy.sign(fa_0) == sympy.sign(fb_0):
            print('f(a) and f(b) 同号')
            sys.exit()
        else:
            x = a
            while True:
                count = count + 1
                c = x - f.value(x)/f.diff_value(x)

                # Newton与二分法结合，找下一个点
                if (c > a) and (c < b):
                    x = c
                else:
                    x = a+(b-a)/2

                # 更新有根区间
                if sympy.sign(f.value(a)) == sympy.sign(f.value(x)):
                    a = x
                else:
                    b = x

                # 判断计算结束
                if abs(f.value(x)) < e:
                    res = x
                    break

        return res, count
```

四、 算例

1. $x \times sin(x) - 1 = 0$，有根区间为 $(1, 2)$，误差限为 1×10^{-5}。

```
试值法 根: 1.11416, 迭代次数: 3
牛顿法 根: 1.11416, 迭代次数: 2
```

2. $x^2 - 5 = 0$，有根区间为 $(2, 3)$，误差限为 1×10^{-5}。

```
试值法 根: 2.23607, 迭代次数: 7
牛顿法 根: 2.23607, 迭代次数: 3
```

3. $x^3 - 3x + 2 = 0$，有根区间为 (-2.5, -1.5)，误差限为 1×10^{-5}。

```
试值法 根: -2.00000, 迭代次数: 11
牛顿法 根: -2.00000, 迭代次数: 4
```

五、 结论

1. 相比于试值法，Newton 与二分法结合的算法收敛速度更快；
2. Newton 与二分法结合提升了原始 Newton 法的稳定性；
3. 无论是试值法还是 Newton 与二分法结合的算法，都只能求解函数穿过 x 轴的根，不能求解函数与 x 轴相切的根，例如在算例 3 中，无法求解 $x = 1.0$ 的根。

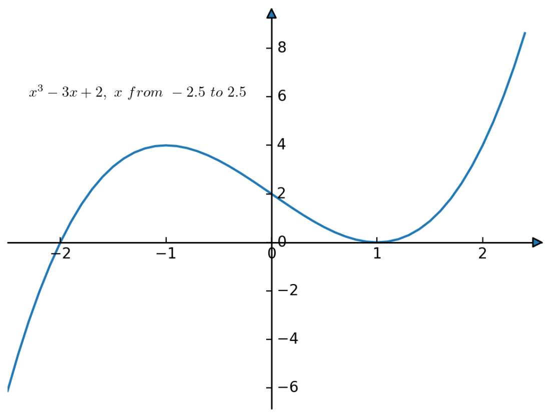

图 2.3 $f(x) = x^3 - 3x + 2$

附录 A　第二章代码

q2-1.py

```python
import sys
import sympy

def func_1_expr():
    """
    x*sin(x) - 1 = 0
    """
    x = sympy.symbols('x')
    return x*sympy.sin(x)-1.0

def func_2_expr():
    """
    x^2 - 5 = 0
    """
    x = sympy.symbols('x')
    return x**2.0 - 5.0

def func_3_expr():
    """
    x^3 -3x + 2 = 0
    """
    x = sympy.symbols('x')
    return x**3.0 - 3.0*x +2

class _func():
    """
    计算一元函数值及导数值
    """
    def __init__(self, expr, eff=15):
        """
        初始化计算表达式
        expr@表达式
        eff@有效数字位数
        """
        self._expr = expr()
        self._eff = eff
        self._x = list(self._expr.free_symbols)[0]
        self._diff_expr = sympy.diff(self._expr, self._x)

```

```python
    def value(self, x):
        """
        计算 FUNC 值
        """
        expr = self._expr
        return expr.subs('x', x).evalf(self._eff)

    def diff_value(self, x):
        """
        计算导数值
        """
        expr = self._diff_expr
        return expr.subs('x', x).evalf(self._eff)



def TrailValue(expr, a, b, e):
    """
    试值法
    F@函数
    A@区间下限
    B@区间上限
    E@容忍误差限
    """
    f = _func(expr)
    fa_0 = f.value(a)
    fb_0 = f.value(b)
    res = 0
    count = 0
    if abs(fa_0) < e:
        res = a
    elif abs(fb_0) < e:
        res = b
    elif sympy.sign(fa_0) == sympy.sign(fb_0):
        print('f(a) and f(b) 同号')
        sys.exit()
    else:
        while True:
            count = count + 1
            fa = f.value(a)
            fb = f.value(b)
            c = b - ((b-a)/(fb - fa))*fb
            fc = f.value(c)

            # 更新有根区间
            if sympy.sign(fa) == sympy.sign(fc):
                a = c
            else:
```

```python
                b = c

            # 判断计算结束
            if abs(f.value(c)) < e:
                res = c
                break

    return res, count

def Newton(expr, a, b, e):
    """
    牛顿法与二分法结合
    f@函数
    a@区间下限
    b@区间上限
    e@容忍误差限
    """
    f = _func(expr)
    fa_0 = f.value(a)
    fb_0 = f.value(b)
    res = 0
    count = 0
    if abs(fa_0) < e:
        res = a
    elif abs(fb_0) < e:
        res = b
    elif sympy.sign(fa_0) == sympy.sign(fb_0):
        print('f(a) and f(b) 同号')
        sys.exit()
    else:
        x = a
        while True:
            count = count + 1
            c = x - f.value(x)/f.diff_value(x)

            # Newton与二分法结合，找下一个点
            if (c > a) and (c < b):
                x = c
            else:
                x = a+(b-a)/2

            # 更新有根区间
            if sympy.sign(f.value(a)) == sympy.sign(f.value(x)):
                a = x
            else:
                b = x

            # 判断计算结束
```

```python
            if abs(f.value(x)) < e:
                res = x
                break

    return res, count

def main():
    """
    MAIN
    """
    res_t, count_t = TrailValue(func_1_expr, 1, 2, 1.0/sympy.Pow(10, 5))
    print('试值法 func 1 根: %.5f, 迭代次数: %d' % (res_t, count_t))

    res_n, count_n = Newton(func_1_expr, 1, 2, 1.0/sympy.Pow(10, 5))
    print('牛顿法 func 1 根: %.5f, 迭代次数: %d' % (res_n, count_n))

    res_t, count_t = TrailValue(func_2_expr, 2, 3, 1.0/sympy.Pow(10, 5))
    print('试值法 func 2 根: %.5f, 迭代次数: %d' % (res_t, count_t))

    res_n, count_n = Newton(func_2_expr, 2, 3, 1.0/sympy.Pow(10, 5))
    print('牛顿法 func 2 根: %.5f, 迭代次数: %d' % (res_n, count_n))

    res_t, count_t = TrailValue(func_3_expr, -2.5, -1.5, 1.0/sympy.Pow(10, 5))
    print('试值法 func 3 根: %.5f, 迭代次数: %d' % (res_t, count_t))

    res_n, count_n = Newton(func_3_expr, -2.5, -1.5, 1.0/sympy.Pow(10, 5))
    print('牛顿法 func 3 根: %.5f, 迭代次数: %d' % (res_n, count_n))

if __name__ == "__main__":
    main()
```