



# 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

$$(a, f(a)), (b, f(b))$$

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  $(a, f(a))$  与  $(b, f(b))$  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  $\bar{x} = \frac{a+b}{2}$ .

- 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。

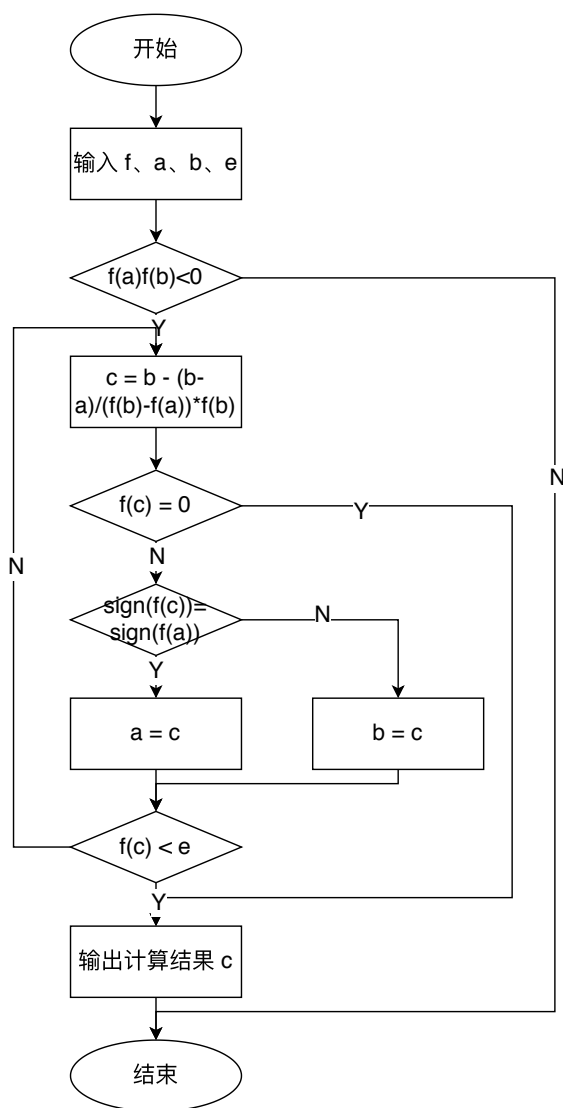


图 2.1 试值法流程图

### Newton 法同二分法结合流程

图2.2 为试值法算法流程图。

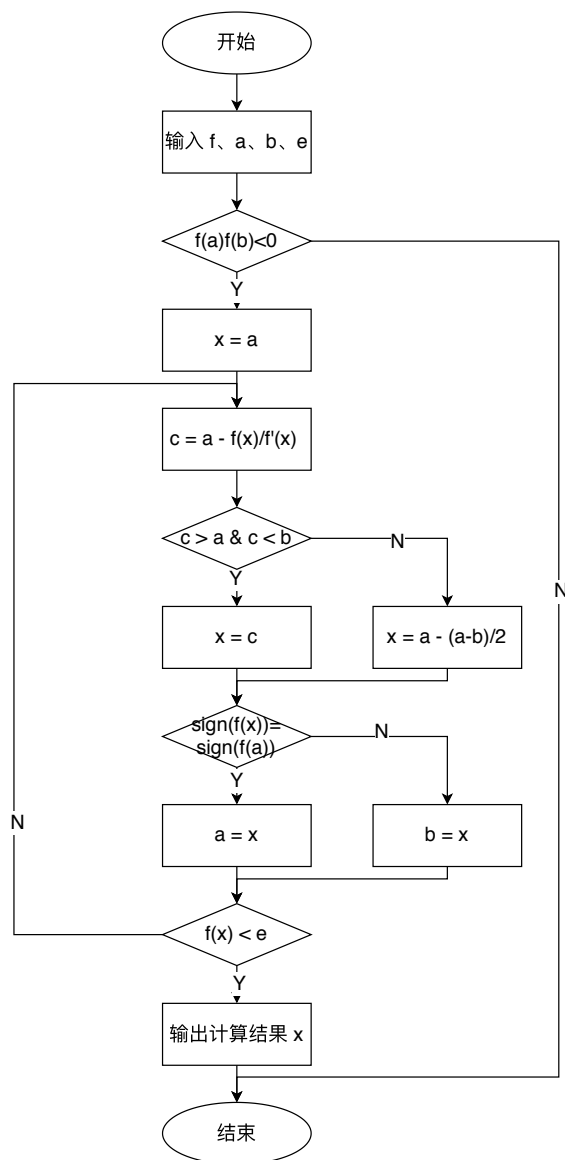


图 2.2 Newton 法同二分法结合流程

### 三、 程序

#### 试值法

```

1 def TrailValue(expr, a, b, e):
2     """
3     试值法
4     F@函数
5     A@区间下限
6     B@区间上限
7     E@容忍误差限
8     """
9     f = _func(expr)

```

```

10     fa_0 = f.value(a)
11     fb_0 = f.value(b)
12     res = 0
13     count = 0
14     if abs(fa_0) < e:
15         res = a
16     elif abs(fb_0) < e:
17         res = b
18     elif sympy.sign(fa_0) == sympy.sign(fb_0):
19         print('f(a) and f(b) 同号')
20         sys.exit()
21     else:
22         while True:
23             count = count + 1
24             fa = f.value(a)
25             fb = f.value(b)
26             c = b - ((b-a)/(fb - fa))*fb
27             fc = f.value(c)
28
29             # 更新有根区间
30             if sympy.sign(fa) == sympy.sign(fc):
31                 a = c
32             else:
33                 b = c
34
35             # 判断计算结束
36             if abs(f.value(c)) < e:
37                 res = c
38                 break
39
40     return res, count

```

---

## Newton 法同二分法结合

---

```

1 def Newton(expr, a, b, e):
2     """
3     牛顿法与二分法结合
4     F@函数
5     A@区间下限
6     B@区间上限
7     E@容忍误差限
8     """
9     f = _func(expr)
10    fa_0 = f.value(a)
11    fb_0 = f.value(b)
12    res = 0
13    count = 0
14    if abs(fa_0) < e:

```

```

15     res = a
16 elif abs(fb_0) < e:
17     res = b
18 elif sympy.sign(fa_0) == sympy.sign(fb_0):
19     print('f(a) and f(b) 同号')
20     sys.exit()
21 else:
22     x = a
23     while True:
24         count = count + 1
25         c = x - f.value(x)/f.diff_value(x)
26
27         # NEWTON与二分法结合, 找下一个点
28         if (c > a) and (c < b):
29             x = c
30         else:
31             x = a+(b-a)/2
32
33         # 更新有根区间
34         if sympy.sign(f.value(a)) == sympy.sign(f.value(x)):
35             a = x
36         else:
37             b = x
38
39         # 判断计算结束
40         if abs(f.value(x)) < e:
41             res = x
42             break
43
44     return res, count

```

---

## 四、算例

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

---

1	试值法	根: 1.11416, 迭代次数: 3
2	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  $1 \times 10^{-5}$ 。

---

1	试值法	根: 2.23607, 迭代次数: 7
2	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  $1 \times 10^{-5}$ 。

---

1	试值法	根: -2.00000, 迭代次数: 11
2	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。

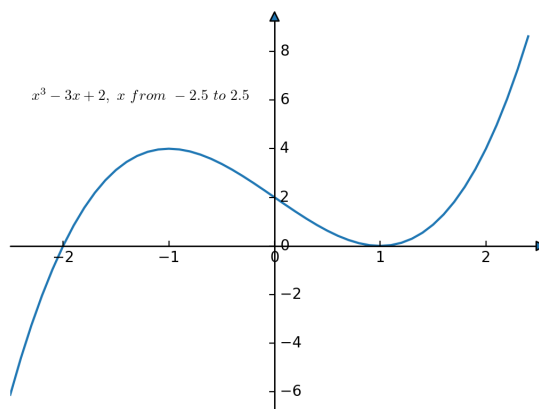


图 2.3  $f(x) = x^3 - 3x + 2$



## 附录 A 第二章代码

q2-1.py

---

```
1 import sys
2 import sympy
3
4 def func_1_expr():
5     """
6      $x \sin(x) - 1 = 0$ 
7     """
8     x = sympy.symbols('x')
9     return x*sympy.sin(x)-1.0
10
11 def func_2_expr():
12     """
13      $x^2 - 5 = 0$ 
14     """
15     x = sympy.symbols('x')
16     return x**2.0 - 5.0
17
18 def func_3_expr():
19     """
20      $x^3 - 3x + 2 = 0$ 
21     """
22     x = sympy.symbols('x')
23     return x**3.0 - 3.0*x +2
24
25 class _func():
26     """
27     计算一元函数值及导数值
28     """
29     def __init__(self, expr, eff=15):
30         """
31         初始化计算表达式
32         EXPR@表达式
33         EFF@有效数字位数
34         """
35         self._expr = expr()
36         self._eff = eff
37         self._x = list(self._expr.free_symbols)[0]
38         self._diff_expr = sympy.diff(self._expr, self._x)
39
```

```

40     def value(self, x):
41         """
42         计算 FUNC 值
43         """
44         expr = self._expr
45         return expr.subs('x', x).evalf(self._eff)
46
47     def diff_value(self, x):
48         """
49         计算导数值
50         """
51         expr = self._diff_expr
52         return expr.subs('x', x).evalf(self._eff)
53
54
55
56 def TrailValue(expr, a, b, e):
57     """
58     试值法
59     F@函数
60     A@区间下限
61     B@区间上限
62     E@容忍误差限
63     """
64     f = _func(expr)
65     fa_0 = f.value(a)
66     fb_0 = f.value(b)
67     res = 0
68     count = 0
69     if abs(fa_0) < e:
70         res = a
71     elif abs(fb_0) < e:
72         res = b
73     elif sympy.sign(fa_0) == sympy.sign(fb_0):
74         print('f(a) and f(b) 同号')
75         sys.exit()
76     else:
77         while True:
78             count = count + 1
79             fa = f.value(a)
80             fb = f.value(b)
81             c = b - ((b-a)/(fb - fa))*fb
82             fc = f.value(c)
83
84             # 更新有根区间
85             if sympy.sign(fa) == sympy.sign(fc):
86                 a = c
87             else:

```

```

88         b = c
89
90         # 判断计算结束
91         if abs(f.value(c)) < e:
92             res = c
93             break
94
95     return res, count
96
97 def Newton(expr, a, b, e):
98     """
99     牛顿法与二分法结合
100    F@函数
101    A@区间下限
102    B@区间上限
103    E@容忍误差限
104    """
105    f = _func(expr)
106    fa_0 = f.value(a)
107    fb_0 = f.value(b)
108    res = 0
109    count = 0
110    if abs(fa_0) < e:
111        res = a
112    elif abs(fb_0) < e:
113        res = b
114    elif sympy.sign(fa_0) == sympy.sign(fb_0):
115        print('f(a) and f(b) 同号')
116        sys.exit()
117    else:
118        x = a
119        while True:
120            count = count + 1
121            c = x - f.value(x)/f.diff_value(x)
122
123            # NEWTON与二分法结合，找下一个点
124            if (c > a) and (c < b):
125                x = c
126            else:
127                x = a+(b-a)/2
128
129            # 更新有根区间
130            if sympy.sign(f.value(a)) == sympy.sign(f.value(x)):
131                a = x
132            else:
133                b = x
134
135        # 判断计算结束

```

```

136         if abs(f.value(x)) < e:
137             res = x
138             break
139
140     return res, count
141
142 def main():
143     """
144     MAIN
145     """
146     res_t, count_t = TrailValue(func_1_expr, 1, 2, 1.0/sympy.Pow(10, 5))
147     print('试值法 func 1 根: %.5f, 迭代次数: %d' % (res_t, count_t))
148
149     res_n, count_n = Newton(func_1_expr, 1, 2, 1.0/sympy.Pow(10, 5))
150     print('牛顿法 func 1 根: %.5f, 迭代次数: %d' % (res_n, count_n))
151
152     res_t, count_t = TrailValue(func_2_expr, 2, 3, 1.0/sympy.Pow(10, 5))
153     print('试值法 func 2 根: %.5f, 迭代次数: %d' % (res_t, count_t))
154
155     res_n, count_n = Newton(func_2_expr, 2, 3, 1.0/sympy.Pow(10, 5))
156     print('牛顿法 func 2 根: %.5f, 迭代次数: %d' % (res_n, count_n))
157
158     res_t, count_t = TrailValue(func_3_expr, -2.5, -1.5, 1.0/sympy.Pow(10, 5))
159     print('试值法 func 3 根: %.5f, 迭代次数: %d' % (res_t, count_t))
160
161     res_n, count_n = Newton(func_3_expr, -2.5, -1.5, 1.0/sympy.Pow(10, 5))
162     print('牛顿法 func 3 根: %.5f, 迭代次数: %d' % (res_n, count_n))
163
164 if __name__ == "__main__":
165     main()

```

---