第七章 非线性方程的数值解法

7.1 二分法

- 考虑$f(x)=0$,若$f(a)f(b)<0$,则$f(x)$在(a,b)上必有零点
- 具体做法
  - 计算$f\left(\frac{a+b}{2}\right)$,记$a_1=a,b_1=b$,$\Rightarrow\begin{cases}(1)\text{若}f\left(\frac{a+b}{2}\right)=0,\frac{a+b}{2}\text{就是解,结束}\\(2)\text{若}f\left(\frac{a+b}{2}\right)\text{与}f(a_1)\text{符号相同},a_2=\frac{a_1+b_1}{2},b_2=b_1\\(3)\text{若}f\left(\frac{a+b}{2}\right)\text{与}f(a_1)\text{符号不同},a_2=a_1,b_2=\frac{a_1+b_1}{2}\end{cases}$

    则(a_2,b_2)成为新的有根区间,$(a_2,b_2)\subset(a_1,b_1)$,$b_2-a_2=\frac{b_2-a_2}{2}$,反复进行上述过程，可得一系列有根区间序列$\{(a_n,b_n)\}_{n=1}^{\infty}$
  - (1)输入：函数$f(x)$有根区间端点a,b，最大容许迭代次数N,误差允许限TOL，机器精度ε
(2)$n=1$
(3)$x_n=\frac{a+b}{2}$
(4)若$|f(x_n)|<\varepsilon$或$\frac{b-a}{2}<TOL$,输出x_n为结果,结束
(5)若$n>N$,输出达到最大容许迭代次数，失败，结束
(6)若$f(a)f(x_n)<0$,则$b=x_n$,否则$a=x_n$
(7)令$n=n+1$,转(3)

```python
def bisection_method(f, a, b, N, TOL, epsilon):
    """
    二分法求解方程 f(x) = 0 的根
    参数:
    f: 函数对象
    a, b: 初始区间端点
    N: 最大允许迭代次数
    TOL: 误差允许限
    epsilon: 机器精度
    """
    # 检查初始区间是否有效
    if f(a) * f(b) >= 0:
        print("初始区间[a, b]可能不包含根或包含多个根")
        return None

    n = 1  # 初始化迭代次数

    while n <= N:
        # 计算区间中点
        x_n = (a + b) / 2

        # 检查是否满足终止条件
        if abs(f(x_n)) < epsilon or (b - a) / 2 < TOL:
            print(f"在第 {n} 次迭代时找到近似解 x_n = {x_n}")
            return x_n

        # 判断区间缩小方向
        if f(a) * f(x_n) < 0:
            b = x_n  # 根在左区间
        else:
            a = x_n  # 根在右区间

        # 更新迭代次数
        n += 1

    # 如果超过最大迭代次数
    print("达到最大允许迭代次数，求解失败")
    return None
```

- 误差估计
  - 设$\in C[a,b]$,且$f(a)f(b)<0$,则二分法生成的数列$\{x_n\}$与f的零点x^*之间的误差为$|x_n-x^*|\leqslant\frac{b-a}{2^n}$
  - 例：用二分法计算$\sqrt{2}$的近似值，确定最小迭代次数，使误差的绝对值不超过10^{-4}
- 定理　在二分法中，$\lim\limits_{n\to\infty}a_n=\lim\limits_{n\to\infty}b_n=\lim\limits_{n\to\infty}x_n$,且极限为$f(x)$的零点

7.2 简单迭代法

- 把$f(x)=0$改写成$x=\varphi(x)$,即把非线性方程的求根问题转化成求$\varphi(x)$的不动点问题
- 具体做法
  - 取$x_0=[a,b]$,由$x_{k+1}=\varphi(x_k),k=0,1,\ldots$可得序列$\{x_n\}_{n=0}^{\infty}$,若序列收敛,记$\alpha=\lim\limits_{n\to\infty}x_n$
对上式两边取极限$k\to\infty,\alpha=\varphi(Q)$,则$\alpha$是$\varphi(x)$的不动点
  - (1)输入：函数$\varphi(x)$,初始值x_0，最大容许迭代次数N,误差允许限TOL
(2)$n=1$
(3)$x_n=\varphi(x_{n-1})$
(4)若$|x_n-\varphi(x_{n-1})|<TOL$,输出$x_n$为结果,结束
(5)若$n>N$,输出达到最大容许迭代次数，结束
(6)令$n=n+1$,转(3)

```python
def simple_iteration(phi, x0, N, TOL):
    """
    简单迭代法求解方程 phi(x) = x 的固定点
    参数:
    phi: 迭代函数
    x0: 初始值
    N: 最大允许迭代次数
    TOL: 误差允许限
    """
    n = 1  # 初始化迭代次数
    x_n = x0  # 初始化 x_n

    while n <= N:
        # 计算新的迭代值
        x_next = phi(x_n)

        # 检查误差是否满足条件
        if abs(x_next - x_n) < TOL:
            print(f"在第 {n} 次迭代时找到近似解 x_n = {x_next}")
            return x_next

        # 更新 x_n 为新值
        x_n = x_next

        # 更新迭代次数
        n += 1

    # 如果达到最大迭代次数仍未满足条件
    print("达到最大允许迭代次数，求解失败")
    return None
```

  - 例：用简单迭代法计算$x^3+4x^2-10=0$在$[1,2]$上的根
- 不动点算法的收敛性
  - 考虑方程$x=\varphi(x)\in C([a,b])$，若$\begin{cases}1.\text{映内性：若}x\in[a,b]\text{，则}\varphi(x)\in[a,b]\text{；}\\2.\text{压缩性：}\exists L\in(0,1)\text{，s.t.}|\varphi(x_1)-\varphi(x_2)|\le L|x_1-x_2|\text{，}\forall x_1,x_2\in[a,b]\text{。}\\\quad(\text{可放宽为}Lipschitz\text{条件：}|\varphi(x_1)-\varphi(x_2)|\le L|x_1-x_2|,\forall x_1,x_2\in[a,b])\end{cases}$

    则$\forall x_0\in[a,b]$，由$x_{k+1}=\varphi(x_k),k=0,1,2,\ldots$得到的序列$\{x_k\}_{n=0}^{\infty}$收敛到$\varphi(x)$在$[a,b]$上的唯一不动点$x^*$

    且满足以下估计：$|x^*-x_k|\leqslant\frac{L}{1-L}|x_k-x_{k-1}|;|x^*-x_k|\leqslant\frac{L^k}{1-L}|x_k-x_{k-1}|$

    误差界限假设期望误差小于ε，即$|x^*-x_k|<\varepsilon$，则需满足：$|x_{k+1}-x_k|<\frac{\varepsilon(1-L)}{L}$

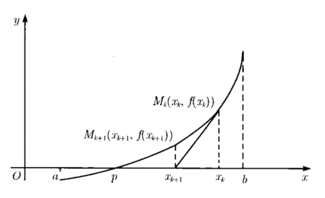
    进一步，由递推公式可以估计需要的最小迭代次数k：$k>\log\frac{|x_1-x_0|}{\varepsilon(1-L)}$,且$\lim\limits_{k\to\infty}\frac{x^*-x_{k+1}}{x^*-x_k}=\varphi(x^*)$
  - 例子

    有根区间为$[1,2]$，$x^*=1.365$
(1)$\varphi_1(x)=x-x^3-4x^2+10,\varphi_1(1)=6$，$\varphi_1(2)=-12\Rightarrow$不映内
(2)$\varphi_2(x)=\sqrt{\frac{10}{x}-4x},\varphi_2(1)=\sqrt{6},\varphi_2(2)=\sqrt{-3}\Rightarrow$不映内
(3)$\varphi_3(x)=\frac{\sqrt{10-x^3}}{2},\varphi_3(1)=\frac{3}{2},\varphi_3(2)=\frac{\sqrt{2}}{2},\varphi_3\left(\frac{3}{2}\right)=\frac{1}{2}\sqrt{\frac{53}{8}}\approx1.3\Rightarrow$在$[1.1.5]$中映内
当$x\in[1,1.5],\varphi_3'(x)=-\frac{3x^2}{4\sqrt{10-x^3}}>\varphi_3'\left(\frac{3}{2}\right)=-\frac{27}{16}\sqrt{\frac{8}{63}}\approx-0.6$,即$|\varphi_3'(x)|<0.6$
(4)$\varphi_4(x)=\sqrt{\frac{10}{4+x}},\varphi_4(1)=\sqrt{2},\varphi_4(2)=\sqrt{\frac{5}{3}}\Rightarrow$映内
$|\varphi_4'(x)|=\left|\frac{1}{2\sqrt{\frac{10}{4+x}}}\left(-\frac{10}{(4+x)^2}\right)\right|=\frac{\sqrt{10}}{2(4+x)^{\frac{3}{2}}}<|\varphi_4'(1)|=\frac{\sqrt{2}}{10}\approx0.14<1\Rightarrow$压缩
- 局部收敛
  - 由于$|\varphi'(x)|<1$不易满足，讨论局部收敛
  - 定义　$\exists\,\varphi(x)$的不动点x^*的一个闭邻域$N(x^*)=[x^*-\delta,x^*+\delta],\delta>0$,s.t.$\forall x_0\in N(x^*)$,简单迭代法产生的序列$\{x_k\}$均收敛于$x^*$
  - 定理　记x^*是$\varphi(x)$的不动点，$\varphi'(x)$在x^*某邻域内连续，且$|\varphi(x^*)|\leqslant L<1$,则简单迭代法局部收敛
- 迭代法的收敛阶
  - 设$\{x_k\}$收敛于x^*,$\begin{cases}e_k=x_k-x^*\\\tilde{e_k}=\tilde{x_k}-x^*\end{cases}$,若$\exists p\geqslant1$及非零常数$c>0,s.t.\lim\limits_{k\to\infty}\frac{|e_{k+1}|}{|e_k|^p}=c$，则称$\{x_k\}$p阶收敛，$c$为渐进误差常熟
  - $p=1$且$0<c<1$时，称为线性收敛
$p>1$时称为超线性收敛
$p=2$时称为平方收敛或二次收敛
  - 例子

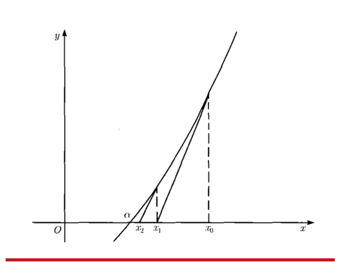
    $\{x_k\}$是线性收敛于x^*,$\{\widetilde{x_k}\}$是二次收敛于x^*,即$\exists c,\tilde{c}$,s.t.$\lim\limits_{k\to\infty}\frac{|e_{k+1}|}{|e_k|}=c,\lim\limits_{k\to\infty}\frac{|\widetilde{e_{k+1}}|}{|\widetilde{e_k}|}=\tilde{c}$
为对比，取$|e_0|=|\widetilde{e_0}|=1,c=\tilde{c}=0.15$,为时误差小于$10^{-8}$,$|e_k|\approx c\cdot|e_{k-1}|\approx\cdots\approx c^{k-1}|e_1|\approx c^k|e_0|$
为使$|e_k|<10^{-8}$,只需令$c^k|e_0|<10^{-8},0.75^k<10^{-8},k>\log_{0.75}10^{-8}=\frac{-8}{\log_{10}0.75}\approx64$, 需65步
$|\widetilde{e_k}|\approx\tilde{c}|\widetilde{e_k}|^2\approx\tilde{c}\cdot\tilde{c}^2|\widetilde{e_{k-2}}|^4\approx\tilde{c}^{2^k-1}|\widetilde{e_{k-2}}|^{2^k}$,为使$|\widetilde{e_{k-2}}|<10^{-8}$,需$\tilde{c}^{2^k-1}|\widetilde{e_{k-2}}|^{2^k}<10^{-8},k>6$,需7步
  - 若$\varphi'(x^*)\neq0$，最多是线性收敛

7.3 Newton类迭代法

- *Newton*法
  - 构造$\varphi(x)=x-f(x)g(x),\varphi'(x)=1-f'(x)g(x)-f(x)g'(x)$
为使$0=\varphi'(x^*)=1-f'(x^*)g(x^*),g(x^*)=\frac{1}{f(x^*)}$,取$g(x)=\frac{1}{f'(x)}$,即$\varphi(x)=x-\frac{f(x)}{f'(x)}$



  - 算法
    - 输入：初始值x_0，误差容限TOL；最大迭代次数m。
输出：近似解p或失败信息。
    - 1. $p_0:=x_0$。
2. 对$i=1,2,\ldots,m$进行步骤3和步骤4。
3. $p:=p_0-\frac{f(p_0)}{f'(p_0)}$。
4. 若$|p-p_0|<\mathrm{TOL}$，则输出p；停机；否则$p_0:=p$。
5. 输出("Method failed")；停机。
在步骤4中的迭代终止标准也可用：$\frac{|p-p_0|}{|p|}<\mathrm{TOL}$,或$\frac{|p-p_0|}{|p|}<\mathrm{TOL}$,且$|f(p)|<\mathrm{TOL}$.
  - 收敛定理
    - 若迭代函数$g(x)$的高阶导数在x^*的邻域内连续，则简单迭代法p阶收敛
$\Updownarrow$
$\varphi(x^*)=x^8,\varphi^{(l)}(x^*)=0,l=1,2,\ldots,p-1,\varphi^{(p)}(x^*)\neq0$,且$\lim\limits_{k\to\infty}\frac{e_{k+1}}{e_k^p}=\frac{1}{p!}\varphi^{(p)}(x^*)$
    - 设函数$f(x)$在有限区间$[a,b]$上存在二阶导数，且满足条件：
1. $f(a)f(b)<0$；
2. $f'(x)\neq0,\ x\in[a,b]$，$f''(x)$在$[a,b]$上不变号；
3. $\left|\frac{f(a)}{f'(a)}\right|<b-a,\quad\left|\frac{f(b)}{f'(b)}\right|<b-a.$
则*Newton*法对任意的初始值$x_0\in[a,b]$都收敛于方程$f(x)=0$的唯一解p，且收敛阶数为2。
- 简化的*Newton*法
  - 为了避免计算导数值，可将$f'(x_n)$取位某个定点处的值，如$f'(x_0)$，此时迭代法为$\varphi(x)=x-\frac{f(x)}{f'(x_0)}$



- *m*重根
  - 平方收敛的修改的*Newton*方法：$x_{n+1}=x_n-m\frac{f(x_n)}{f'(x_n)}$,但这要预先知道$\alpha$的重数
  - 在非单根的情况下，一般不知道重数m。这时也可采用如下修改*Newton*方法计算$f(x)=0$的重根。
由于α为$f(x)$的m重根，这时$f(x)=(x-\alpha)^m g(x)$。假设$g(x)$充分可微，且$g(\alpha)\neq0$。记
$u(x)=\frac{f(x)}{f'(x)}$，　若α为$f(x)$的m重根，则$u(x)=\frac{(x-\alpha)g(x)}{mg(x)+(x-\alpha)g'(x)}$。故$\alpha$为$u(x)$的单重零点。
这时*Newton*方法修改为$x_{n+1}=x_n-\frac{u(x_n)}{u'(x_n)}=x_n-\frac{f(x_n)f'(x_n)}{[f'(x_n)]^2-f(x_n)f''(x_n)}$。
- *Newton*下山法
  - 在迭代格式$x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)}$中引入一个因子$\lambda$，将迭代格式变为$x_{n+1}=x_n-\lambda\frac{f(x_n)}{f'(x_n)}$。
其中，λ称为下山法因子。选择下山法因子的原则是使得下式成立（即$f(x)$的绝对值变小）：$|f(x_{n+1})|<|f(x_n)|,\quad n=0,1,\ldots$
  - 这种做法扩大了初始值的选取范围，并使得迭代序列有比较快的收敛速度。

    在整个计算过程中，下山法因子λ是变化的，一般可以按如下方法进行。
首先取$\lambda=1$，由初始值x_0算出x_1。如果$|f(x_1)|<|f(x_0)|$，则可以将λ扩大一倍，应用$x_2=x_n-2\frac{f(x_1)}{f'(x_1)}$计算$x_2$；
如果$|f(x_1)|>|f(x_0)|$，逐次地将λ减半，直到条件$|f(x_1)|<|f(x_0)|$满足为止，并利用最后算出的x_1进行下一步迭代。
如此进行下去，这种做法保证每次迭代都使$|f(x_n)|$的值严格下降，同时在下降的前提下尽可能增大下山法因子λ。
  - 算法

```python
def newton_downhill(f, f_prime, x0, tol=1e-6, max_iter=100):

    f_lambdified = sp.lambdify(x, f)
    f_prime_lambdified = sp.lambdify(x, f_prime)

    xn = x0
    lambda_factor = 1  # 初始步长 λ 从 1 开始

    for n in range(max_iter):
        fxn = f_lambdified(xn)
        f_prime_xn = f_prime_lambdified(xn)

        # 计算 x_new, 并验证下降条件
        while True:
            x_new = xn - lambda_factor * fxn / f_prime_xn
            if abs(f_lambdified(x_new)) < abs(fxn):  # 满足下降条件
                break
            lambda_factor /= 2  # 如果不满足, 减小 λ

        print(f"x_{n} = {xn:.16f}, lambda_{n} = {lambda_factor:.6f}, f(x_{n}) = {fxn:.16f}")

        # 检查是否满足收敛条件
        if abs(x_new - xn) < tol and abs(f_lambdified(x_new)) < tol:
            print(f"收敛到零点: x_{n+1} = {x_new:.16f}, 共进行了 {n + 1} 次迭代")
            return x_new

        # 如果满足下降条件, 尝试在下一次迭代中增大 λ (翻倍)
        lambda_factor *= 2  # 每次翻倍尝试更大的步长
        xn = x_new  # 更新当前点

    print("超过最大迭代次数, 未找到零点。")
    return None

# 参数
x0 = 0.1  # 初始值
tolerance = 1e-6  # 收敛容差

# 求解零点
root = newton_downhill(f, f_prime, x0, tol=tolerance)
print(f"方程的零点为: {root:.16f}")
```