1 问题描述

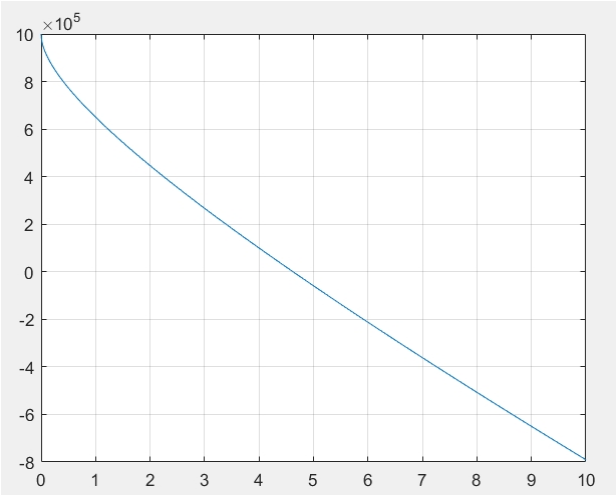
在一个混合均匀的湖中，质量平衡方程可以写为 $v\frac{dc}{dt} = w - Qc - kV\sqrt{c}$
其中$V = 1 \times 10^6 m^3$，$Q = 1 \times 10^5 m^3/yr$，$w = 1 \times 10^6 g/yr$，
$k = 0.25 g^{0.5}/(m^{1.5} yr)$，分别用二分法、试位法、不动点迭代、Newton-Raphson法和割线法求解稳定状态的浓度$c(g/m^3)$，并对各种方法进行比较。

2 问题分析

题目中给的方程是动态方程，要求求解稳定状态的c，稳定状态意味着$\frac{dc}{dt} = 0$，即将问题转化为$w - Qc - kV\sqrt{c} = 0$的非线性方程求根问题。代入数值后
$1000000 - 100000c - 250000\sqrt{c} = 0$。
初步分析一下根的情况，将$\sqrt{c}$换元为x，即求根$-100000x^2 - 250000x + 1000000 = 0$，由韦达定理$x1 + x2 = -2.5 < 0$，$x1x2 = -10$，故一个正根一个负根，且$\sqrt{c} > 0$舍去负根，故方程只有一个根。

```
x=0:0.01:10;
f=@(x)(1000000-100000*x-250000*sqrt(x));
y1_value=f(x);
plot(x, y1_value);
grid();
```

用matlab画图



确实是一个根，而且根大致处于(4,5)区间。

3 代码实现

3.1 二分法

由图可以设初始边界值为4和5。函数单调递减，设下界x_l上界x_u，思路如下：

1. 起始$f(x_l)f(x_u) < 0$
2. 取中间点$x_r = \frac{x_l + x_u}{2}$

3. 如果$f(x_l)f(x_r)<0$落在左区间，令$x_u=x_r$；如果$f(x_l)f(x_r)>0$落在右区间，令$x_l=x_r$；如果$f(x_l)f(x_r)=0$，正好为根。

其中迭代次数可求，为使结果准确，均采用双精度，同时为了防止误差的终止容限过于小使迭代次数过多或者陷入迭代循环，令最终期望绝对误差为双精度机器精度eps的十倍，初始区间长度为1，$n=\frac{log(1/10eps)}{log2}$

```
function [c,err,yc,k]=bisect(f,a,b,delta)
ya=feval(f,a);
yb=feval(f,b);
if(ya*yb>0)
    return;
end
max1=1+round((log(b-a)-log(delta))/log(2)); %防止round舍去小数部分导致迭代次数不够，所以加一
for k=1:max1
    c=(a+b)/2;
    yc=feval(f,c);
    if yc==0
        a=c;
        b=c;
    elseif ya*yc<0
        b=c;
        yb=yc;
    else
        a=c;
        ya=yc;
    end
    if b-a<delta
        break;
    end
end
c=(a+b)/2;
err=abs(b-a)/2;
yc=feval(f,c);
end
```

其中输入部分：f是函数，a、b是左右边界值，delta是容限
输出部分：c是最终估计的零点，yc是f(c)值，err是最终值与真实值绝对误差的最小上界，k是总共迭代的次数

3.2 试位法

思路如下：取两边界点$(x_l,f(x_l)),(x_u,f(x_u))$，直线与x轴交点作为新根估计值，$x_r=x_u-\frac{f_u(x_l-x_u)}{f_l-f_u}$。对新值的处理与二分法一致。

```
function [c,err,yc,k]=regular(f,a,b,delta,epsilon,max1)
ya=feval(f,a);
```

```
yb=feval(f,b);
if ya*yb>0
    return;
end
for k=1:max1
    dx=yb*(b-a)/(yb-ya);
    c=b-dx;
    ac=c-a;
    yc=feval(f,c);
    if yc==0
        break;
    elseif yb*yc>0
        b=c;
        yb=yc;
    else
        a=c;
        ya=yc;
    end
    dx=min(abs(dx),ac);
    if abs(dx)<delta
        break;
    end
    if abs(yc)<epsilon
        break;
    end
end
dx=yb*(b-a)/(yb-ya);
c=b-dx;
err=abs(b-a);
yc=feval(f,c);
end
```

输入-f是函数，a、b是左右边界点，delta是相邻两次迭代值之间误差的容限，epsilon是当前点的函数值与0之间差的容限，max1是最　大迭代次数
输出-c是零点，yc是f(c)，err是最终值与真实值绝对误差的最小上界，k是当前迭代次数。

为了防止迭代次数过多，收敛性差，也考虑用修正后的试位法：修正思路为添加两个计数变量，分别计数两个边界值的不变次数，当某个边界值连续有两次迭代不变，那么此边界量对应的函数值减半，以加快收敛速度。

```
function [c,err,yc,k]=regularnew(f,a,b,delta,epsilon,max1)
ya=feval(f,a);
yb=feval(f,b);
count_a=0;%用于计数边界值不变的次数
count_b=0;
if ya*yb>0
    return;
end
for k=1:max1
```

```
    dx=yb*(b-a)/(yb-ya);
    c=b-dx;
    ac=c-a;
    yc=feval(f,c);
    if yc==0
        break;
    elseif yb*yc>0
        count_a=count_a+1;
        count_b=0;%连续两次累积不变触发
        b=c;
        yb=yc;
    else
        count_b=count_b+1;
        count_a=0;
        a=c;
        ya=yc;
    end
    dx=min(abs(dx),ac);
    if count_a>=2    %当某一边界值由两次迭代不变将此点的函数值缩小一半
        ya=ya/2;
        count_a=0;
    end
    if count_b>=2
        yb=yb/2;
        count_b=0;
    end
    if abs(dx)<delta
        break;
    end
    if abs(yc)<epsilon
        break;
    end
end
err=abs(b-a);
yc=feval(f,c);
end
```

3.3 不动点迭代法

思路：把原方程经过约分移项转化为$c=10-2.5\sqrt{c}=g(c)$，然后用$c_k=g(c_{k-1})$，进行迭代，其中可令初始值x0=5

```
function [k,p,err,P]=fixpt(g,p0,tol,max1)
P(1)=p0;
for k=2:max1
    P(k)=feval(g,P(k-1));
    err=abs(P(k)-P(k-1));%绝对误差
    relerr=err/abs(P(k)+eps);%相对误差
    p=P(k);
    if (err<tol)||(relerr<tol),break;end
end
end
```

输出-g是迭代函数，p0是不动点初始值，tol是相邻两迭代值之差的容限，max1是最大迭代次数
输出-k是当前迭代的次数，err是最终估计值误差，p是不动点最终结果，P是p迭代的序列

3.4 Newton-Raphson法

思路：
已知$f(x_{i+1}) \approx f(x_i) + f^{'}(x_i)(x_{i+1} + x_i)$
假设x_{i+1}就是f(x)的零点，函数值为0，那么则有$0 \approx f(x_i) + f^{'}(x_i)(x_{i+1} + x_i)$
即$x_{i+1} = x_i - \frac{f(x_i)}{f^{'}(x_i)}$，就是寻找$g(x) = x - \frac{f(x)}{f^{'}(x)}$的不动点

```
function [p0,err,k,y]=newton(f,df,p0,delta,epsilon,max1)
for k=1:max1
    p1=p0-feval(f,p0)/feval(df,p0);
    err=abs(p1-p0);%绝对误差
    relerr=2*err/(abs(p1)+delta);%相对误差
    p0=p1;
    y=feval(f,p0);
    if(err<delta)||(relerr<delta)||(abs(y)<epsilon),break;end
end
end
```

输入-f是函数，df是函数一阶导，p0是初始值，delta迭代两次差的容限，epsilon是迭代当前结果函数值与0之间差的容限，max1最大迭代次数
输出-p0是最后结果，k是当前迭代次数，y是f(p0)函数值，err是最终估计值误差

3.5 割线法

思路：$x_{i+1} = x_i - f(x_i)\frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}$，用两个初始估计值依次估计。

```
function [p1,err,k,y]=secant(f,p0,p1,delta,epsilon,max1)
for k=1:max1
    p2=p1-feval(f,p1)*(p1-p0)/(feval(f,p1)-feval(f,p0));
    err=abs(p2-p1);%绝对误差
    relerr=2*err/(abs(p2)+delta);%相对误差
    p0=p1;
    p1=p2;
    y=feval(f,p1);
    if (err<delta)||(relerr<delta)||(abs(y)<epsilon),break;end
end
end
```

输入-f是函数，p0、p1是初始值，delta是相邻两项的容限，epsilon是y的容限，max1是最大迭代次数
输出-p1是估计值，err是最终估计值误差，y是函数值

也可尝试改进割线法，使用扰动小量来估计：$x_{i+1}=x_i-\frac{\delta x_i f(x_i)}{f(x_i+\delta x_i)-f(x_i)}$

```
function [p1,err,k,y]=secantnew(f,p0,varepsilon,delta,epsilon,max1)
for k=1:max1
    p1=p0-varepsilon*p0*feval(f,p0)/(feval(f,p0+varepsilon*p0)-
feval(f,p0));
    err=abs(p1-p0);%绝对误差
    relerr=2*err/(abs(p1)+delta);%相对误差
    p0=p1;
    y=feval(f,p0);
    if(err<delta)||(relerr<delta)||(abs(y)<epsilon),break;end
end
end
```

输入-f是函数，p0是初始值，varepsilon是扰动小量的倍数，delta是相邻两项的容限，epsilon是y的容限，max1是最大迭代次数
输出-p1是估计值，err是误差，y是函数值，k是迭代次数

3.6 用matlab内置的求根函数

```
fzero(f,4)
```

其中f是待求零点的函数，4是估计的初始值。

4 结果分析

每种方法虽然都返回了误差值，但是均不是准确度预测值与真值的误差，一般均为符合跳出函数的容限条件时的最后两次迭代的值的差或者是估计区间的最小值，虽然有一定参考价值，但是由于算法的收敛速度特点不同，迭代情况不同，最终跳出函数的情况也不同，并不能以此作为评估算法精度的标准。同时在后续结果中我们也看到即使最终得到相同的估计值的算法，其

返回的err值也不一定相同，因此我们只能把返回的err值看做真值与估计值之间绝对误差的一个最小上界。

为了比较各算法间的精度，将原方程可以转化为$-c-2.5\sqrt{c}+10=0$，用一元二次方程的求根公式可求得$\sqrt{c}$，再平方可得c，用计算器计算的值我们可以视为真值，为
$c=4.624080932040347909356289142 4641$

4.1 初始值确定

```
f=@(x)(1000000-100000*x-250000*sqrt(x));
g=@(x)(10-2.5*sqrt(x));
df=@(x)(-100000-125000/sqrt(x));
xl=4;
xu=5;
xl=double(xl);
xu=double(xu);
x0=5;
x0=double(x0);
```

以上包括了所有算法会用到的原函数，不动点的迭代函数，导函数，两个边界值或者是两个初始值，单个初始值。均为双精度。
至于对每个算法的误差容限(终止条件)，在代码实现中我们发现，如果将容限设置为机器精度eps，N-R算法与改进割线法都会陷入无限迭代的循环，始终无法到达跳出函数的误差要求，这样不利于我们分析各个算法的特点，于是选用eps的十倍作为所有的容限要求，且经实验这种程度的改变对最终估计值的影响几乎为零，只是对err有些许影响。

4.2.1 二分法

```
[c1,err1,yc1,k1]=bisect(f,xl,xu,10*eps);
fprintf("二分法结果%.16f\n",c1);
fprintf("二分法误差上界%.28f\n",err1);
fprintf("%.28f\n",yc1);
disp(k1);
```

结果如下（自上而下依次为估计值，误差上界，对应函数值，迭代次数）

```
二分法结果4.6240809320403473
二分法误差上界0.0000000000000008881784197001
0.0000000001164153218269348145
    49
```

4.2.2 试位法

```
[c2,err2,yc2,k2]=regular(f,xl,xu,10*eps,10*eps,1000);
fprintf("试位法结果%.16f\n",c2);
fprintf("试位法误差上界%.28f\n",err2);
fprintf("%.28f\n",yc2);
disp(k2);
```

结果如下

```
试位法结果4.6240809320403482
试位法误差上界0.000000000000000888178419700l
-0.00000000011641532182693481459
     9
```

4.2.3试位法修正版

```
[c3,err3,yc3,k3]=regularnew(f,xl,xu,10*eps,10*eps,1000);
fprintf("试位法修正版结果%.16f\n",c3);
fprintf("试位法修正版误差上界%.28f\n",err3);
fprintf("%.28f\n",yc3);
disp(k3);
```

结果如下

```
试位法修正版结果4.6240809320403464
试位法修正版误差上界0.00000000000000177635683940O3
0.00000000011641532182693481459
     6
```

4.2.4 不动点迭代法

```
[k4, c4, err4, P] = fixpt(g, x0, 10*eps, 1000);
fprintf("不动点迭代法结果%.16f\n",c4);
fprintf("不动点迭代法误差上界%.28f\n",err4);
fprintf("%.28f\n",f(c4));
disp(k4);
```

结果如下

```
不动点迭代法结果4.6240809320403509
不动点迭代法误差上界0.000000000000007993605777301l
-0.00000000046566128730773925781
    61
```

4.2.5 Newton-Raphson法

```
[c5,err5,k5,yc5]=newton(f,df,x0,10*eps,10*eps,1000);
fprintf("N-R法结果%.16f\n",c5);
fprintf("N-R法结果误差上界%.28f\n",err5);
fprintf("%.28f\n",yc5);
disp(k5);
```

结果如下

```
N-R法结果4.6240809320403482
N-R法结果误差上界0.00000000000000008881784197001
-0.00000000011641532182693481 45
     4
```

4.2.6 割线法

```
[c6,err6,k6,yc6]=secant(f,xl,xu,10*eps,10*eps,1000);
fprintf("割线法结果%.16f\n",c6);
fprintf("割线法结果误差上界%.28f\n",err6);
fprintf("%.28f\n",yc6);
disp(k6);
```

结果如下

```
割线法结果4.6240809320403482
割线法结果误差上界0.00000000000000017763568394003
-0.00000000011641532182693481 45
     5
```

4.2.7 改进割线法

```
[c7,err7,k7,yc7]=secantnew(f,x0,0.01,10*eps,10*eps,1000);%选择varepsilon为
0.01
fprintf("改进割线法结果%.16f\n",c7);
fprintf("改进割线法结果误差上界%.28f\n",err7);
fprintf("%.28f\n",yc7);
disp(k7);
```

结果如下

```
改进割线法结果4.6240809320403482
改进割线法结果误差上界0.00000000000000266453525910 04
-0.00000000011641532182693481 45
     6
```

4.2.8 内置函数

```
fprintf("%.16f\n",fzero(f,4));
```

结果如下

```
4.6240809320403464
```

4.3 分析

通过与计算器得到的真值比较，以10eps为误差容限估计值的准确度为

试位法=N-R法=割线法=改进割线法>(略大于,可忽略)二分法>试位法修正版=内置函数法>不动点迭代法
其中前面两种估计值的准确度区别极小，可以忽略不计

但是我对试位法修正版与不动点迭代法的准确度存疑，因为本身10eps的误差容限选择是由于N-R的算法限制而选的，而试位法修正版与不动点迭代法可能本身可以接受eps的误差容限，且只有在eps的误差容限下所得的估计值准确度才能最高、最能体现算法准确度，因此将误差容限改为eps重新实验，结果如下。

```
试位法修正版结果4.6240809320403473
试位法修正版误差上界0.000000000000000888178419700l
0.000000000116415321826934814 5
    10

不动点迭代法结果4.6240809320403473
不动点迭代法误差上界0.000000000000000888178419700l
0.000000000116415321826934814 5
    64
```

发现迭代次数变多了，估计值与其他算法结果一致，对应函数值也一致，说明两者受制于容限设置，没能达到最大准确度。（剩下的算法也修改eps做了实验，但是估计值均没有变化）

这个现象从算法原理角度解释：N-R算法涉及到了导数，改进割线法涉及到了微小扰动以及分母的减性抵消，这些都可能导致陷入迭代死循环无法达到跳出循环的容限要求。而不动点迭代法与修正试位法，一个只涉及迭代函数与y=x交点，一个只涉及两端逼近，可以达到容限要求，并且精度越高其结果越准确。
至于其他的算法也类似，而为什么没有当10eps改为eps时估计值产生较大波动，这个则与收敛速度有关，以试位法与试位法修正法为例，修正法添加了对静滞点的调整，可能在某次迭代中突然增加了与真实值的跨度，这时如果容限太大可能会导致提前跳出函数，只有更精的容限才能约束其迭代到准确值。

对于迭代次数，二分法的迭代次数可以根据容限计算；试位法修正法比试位法多可能是在某个边界函数值减半的下一次迭代中产生了不合理的跨度，使其需要更多的迭代次数来重新返回更精确的值；不动点迭代法最多，但是只要保证迭代函数的收敛性，其得到的值是可靠的，而且不用担心精度设置；割线法与改进割线法的迭代次数均相当少，其收敛速度都相当优秀，但是迭代次数少说明每次迭代跨度大，因此返回的误差上界都较大；对于N-R法，其在保证最少的迭代次数的同时也有最小的误差上界，性能及其优秀，但是容易受各种特殊情况影响，从本实验可以看出受误差容限影响，从其定义也可看出受局部极值点、根附近拐点、导数为零等等情况影响。

matlab的内置函数的准确度明显不如其他算法，可能是其内部误差容限设置过大了。

5 个人心得

通过本次作业熟悉了五种求非线性方程根的算法及其改进变体，了解了其各自的优缺点、准确度、迭代次数、受误差容限影响的程度等等，为以后遇到具体问题时该选用哪一种代码、选用何种精度作为误差容限最为合适等等都提供了一份参考。