Homework Chapter 01

Find_root,刘玖阳,应用物理1301,U201310209

Question

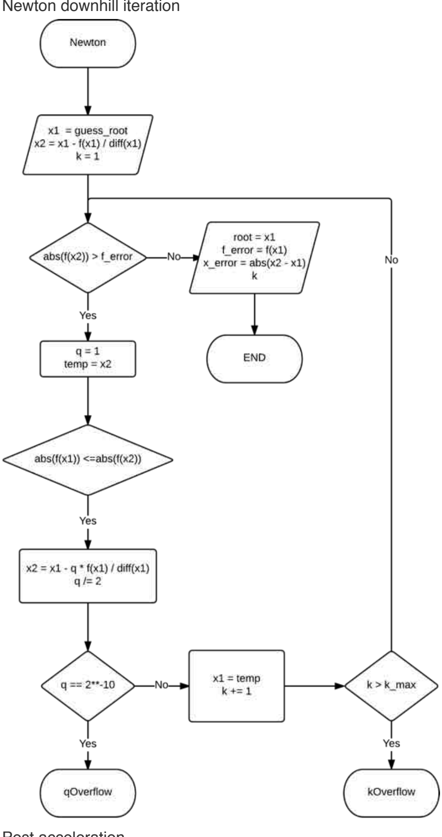
Use *Newton downhill iteration*, *post acceleration* and *Aitken iteration* method to find the roots of the equation

$$f(x) = \frac{x^3}{3} - x = 0$$

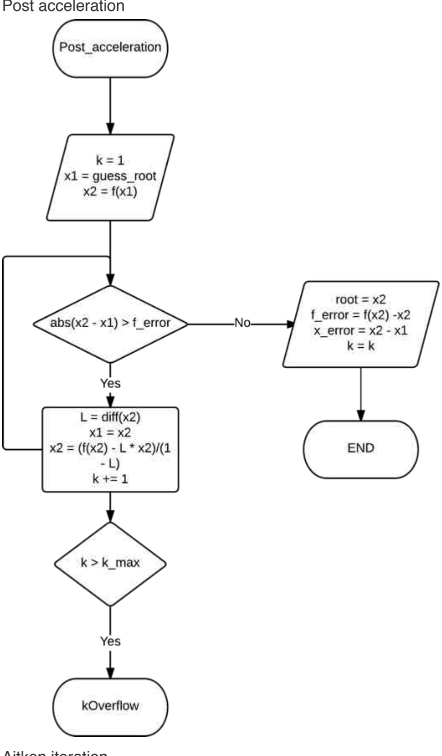
and compare their performances (speed and error)

Used algorithm

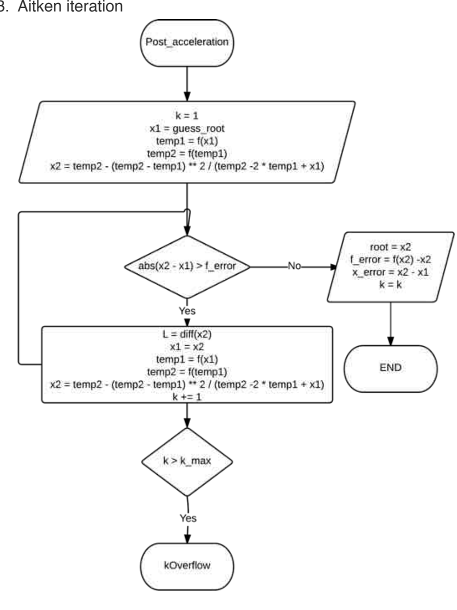
1. Newton downhill iteration



2. Post acceleration



3. Aitken iteration

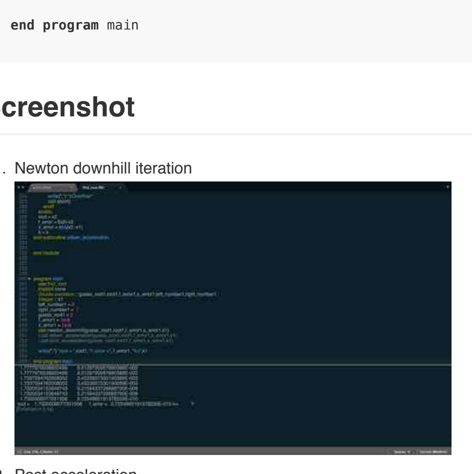


Source Code

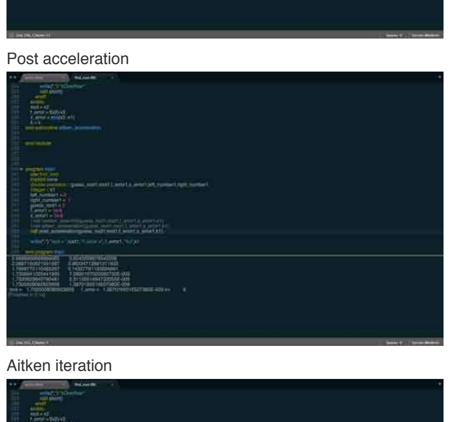
```
module find_root
!----------
!   author:
!       Sequencer
!   description:
!       计算物理非线型方程寻根模块(HUST PHY 2013 第一次作业)
!       MIT协议
!   contains:
!       bolzano(left_number,right_number,root,f_error,x_error,k)    二分法
!       picard(guess_root,root,f_error,x_error,k)   不动点法
!       newton(guess_root,root,f_error,x_error,k)   牛顿法
!       newton_downhill(guess_root,root,f_error,x_error,k)  牛顿下山法
!       post_acceleration(guess_root,root,f_error,x_error,k)    post加速法
!       aitken_acceleration(guess_root,root,f_error,x_error,k)  aitken加速法
!   exception:
!       kOverflow
!       fDivergence
!----------
contains

double precision function f(x)
    !----------
    !   define function
    !----------
    implicit none
    double precision :: x
    f = x**3/3
end function f

double precision function diff(x)
    implicit none
    double precision :: delta,x
    delta = x/100d0
    diff = (f(x+delta)-f(x-delta))/(2d0*delta)
end function diff


subroutine bolzano(left_number,right_number,root,f_error,x_error,k)
    double precision,intent(in) :: left_number, right_number
    double precision,intent(out) :: root
    double precision,intent(inout) :: f_error,x_error
    integer,intent(out) :: k
    double precision temp,x1,x2
    integer k_max
    k = 0
    k_max =100
    if (f(left_number)*f(right_number).gt.0d0) then
        write(*,*) "fDivergence"
        call abort()
    endif
    x1 = left_number
    x2 = right_number
    do while(abs(f(x1)*f(x2)).gt.f_error**2)
        temp = (x1+x2)/2
        if (f(x1)*f(temp).gt.0d0) then
            x1 = temp
        else
            x2 = temp
        endif
        k=k+1
        if (k.gt.k_max) then
            write(*,*) "kOverflow"
            call abort()
        endif
        end do
    if (abs(f(x1)).lt.f_error) then
        if (abs(f(x2)).lt.f_error) then
            root = (x1+x2)/2
            else
            root = x1
        endif
    else
        root = x2
    endif
    ! write(*.*) root
    f_error = f(root)
    x_error = abs(x1-x2)
end subroutine bolzano


subroutine picard(guess_root,root,f_error,x_error,k)
    double precision, intent(in) :: guess_root
    double precision, intent(out) :: root
    double precision, intent(inout) :: f_error,x_error
    integer,intent(out) :: k
    double precision :: x1,x2
    integer k_max
    k_max = 100
    k = 1
    x1 = guess_root
    x2 = f(x1)
    do while(abs(x2-x1).gt.f_error)
        x1 = x2
        x2 = f(x2)
        k=k+1
        write(*,*) k
        if (k.gt.k_max) then
            write(*,*) "kOverflow"
            call abort()
        endif
    enddo
    root = x2
    f_error = f(root)
    x_error = abs(x2-x1)
end subroutine picard


subroutine newton(guess_root,root,f_error,x_error,k)
    double precision,intent(in) :: guess_root
    double precision,intent(out) :: root
    double precision,intent(inout) :: f_error,x_error
    integer,intent(out) :: k
    double precision :: x1,x2,delta
    integer k_max
    k_max = 100
    x1 =guess_root
    k = 1
    do while(abs(f(x1)).gt.f_error)
        x2 = x1
        delta = x1/1000
        x1 = x1 - f(x1)/ diff(x1)
        k = k + 1
        if(k.gt.k_max) then
            write(*,*) "kOverflow"
            call abort()
        endif
    enddo
    root = x1
    f_error = f(x1)
    x_error = abs(x2-x1)
end subroutine newton


subroutine newton_downhill(guess_root,root,f_error,x_error,k)
    double precision,intent(in) :: guess_root
    double precision,intent(out) :: root
    double precision,intent(inout) :: f_error,x_error
    integer,intent(out) :: k
    double precision :: x1,x2,delta,temp,q
    integer k_max
    k_max = 100
    k = 1
    x1 = guess_root
    x2 = x1 - f(x1)/diff(x1)
    write(*,*) x2,f(x2)
    do while (abs(f(x2)) .gt. f_error)
        q = 1
        temp = x2
        do while (abs(f(x1)) .le. abs(f(x2)))
            x2 = x1 - q * f(x1)/diff(x1)
            q = q/2
            if (q .lt. 2**(-10)) then
                write(*,*) "qOverflow"
            endif
        enddo
        write(*,*) x2,f(x2)
        x1 = temp
        k = k + 1
        if (k .gt. k_max) then
            write(*,*) "kOverflow"
            call abort()
        endif
    enddo
    root = x2
    f_error = f(x2)
    x_error = abs(x2-x1)
    k = k
end subroutine newton_downhill


subroutine post_acceleration(guess_root,root,f_error,x_error,k)
    double precision, intent(in) :: guess_root
    double precision, intent(out) :: root
    double precision, intent(inout) :: f_error,x_error
    integer,intent(out) :: k
    double precision :: x1,x2,L
    integer k_max
    k_max = 100
    k = 1
    x1 = guess_root
    x2 = f(x1)
    write(*,*) x2,f(x2)-x2

    do while (abs(f(x2) - x2) .gt. f_error)
        L = diff(x2)
        x1 = x2
        x2 = (f(x2)-L*x2)/(1-L)
        write(*,*) x2,f(x2)-x2
        k = k+1
        if (k > k_max) then
            write(*,*) k,"kOverflow"
            call abort()
        endif
    enddo
    root = x2
    f_error = f(x2) - x2
    x_error = abs(x2-x1)
    k = k
end subroutine post_acceleration


subroutine aitken_acceleration(guess_root,root,f_error,x_error,k)
    double precision, intent(in) :: guess_root
    double precision, intent(out) :: root
    double precision, intent(inout) :: f_error,x_error
    integer,intent(out) :: k
    double precision :: x1,x2,temp1,temp2
    integer k_max
    k_max = 100
    k = 1
    x1 = guess_root
    temp1 = f(x1)
    temp2 = f(temp1)
    x2 = temp2 - (temp2 - temp1) ** 2 / (temp2 - 2 * temp1 + x1)
    write(*,*) x2,f(x2)-x2
    do while (abs(x2 - x1) .gt. f_error)
        x1 = x2
        temp1 = f(x1)
        temp2 = f(temp1)
        x2 = temp2 - (temp2 - temp1) ** 2 / (temp2 - 2 * temp1 + x1)
        write(*,*) x2,f(x2)-x2
        k = k+1
        if (k .gt. k_max) then
            write(*,*) "kOverflow"
            call abort()
        endif
    enddo
    root = x2
    f_error = f(x2)-x2
    x_error = abs(x2 -x1)
    k = k
end subroutine aitken_acceleration
end module


program main
    use find_root
    implicit none
    double precision :: guess_root1,root1,f_error1,x_error1,left_number1,rig
    guess_root1 = ht_number1
    integer :: k1
    left_number1 = 0
    right_number1 =  1
    f_error1 = 1e-8
    x_error1 = 1e-8
    ! call bolzano(left_number1,right_number1,root1,f_error1,x_error1,k1)
    ! call newton_downhill(guess_root1,root1,f_error1,x_error1,k1)
    call aitken_acceleration(guess_root1,root1,f_error1,x_error1,k1)
    ! call post_acceleration(guess_root1,root1,f_error1,x_error1,k1)

    ! write(*,*) "root = ",root1, "f_error =", f_error1, "k=",k1

end program main
```

Screenshot

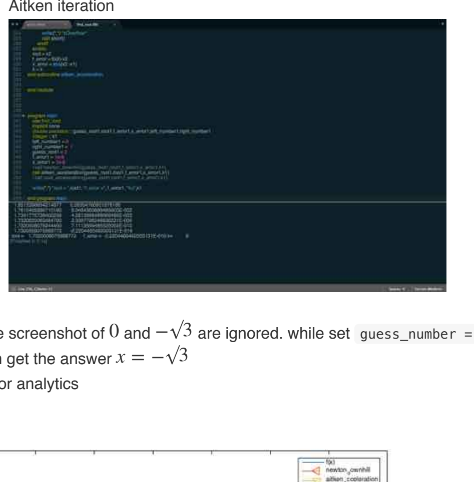
1. Newton downhill iteration



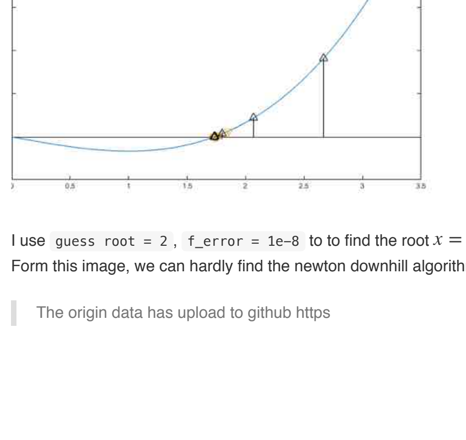
2. Post acceleration



3. Aitken iteration



The screenshot of 0 and $-\sqrt{3}$ are ignored. while set `guess_number = 1` can get the answer $x = 0$, and `guess_number = -2` can get the answer $x = -\sqrt{3}$
Error analytics
—



I use `guess root = 2`, `f_error = 1e-8` to to find the root $x = \sqrt{3}$
Form this image, we can hardly find the newton downhill algorithm. It shows newton downhill is really fast

> The origin data has upload to github https