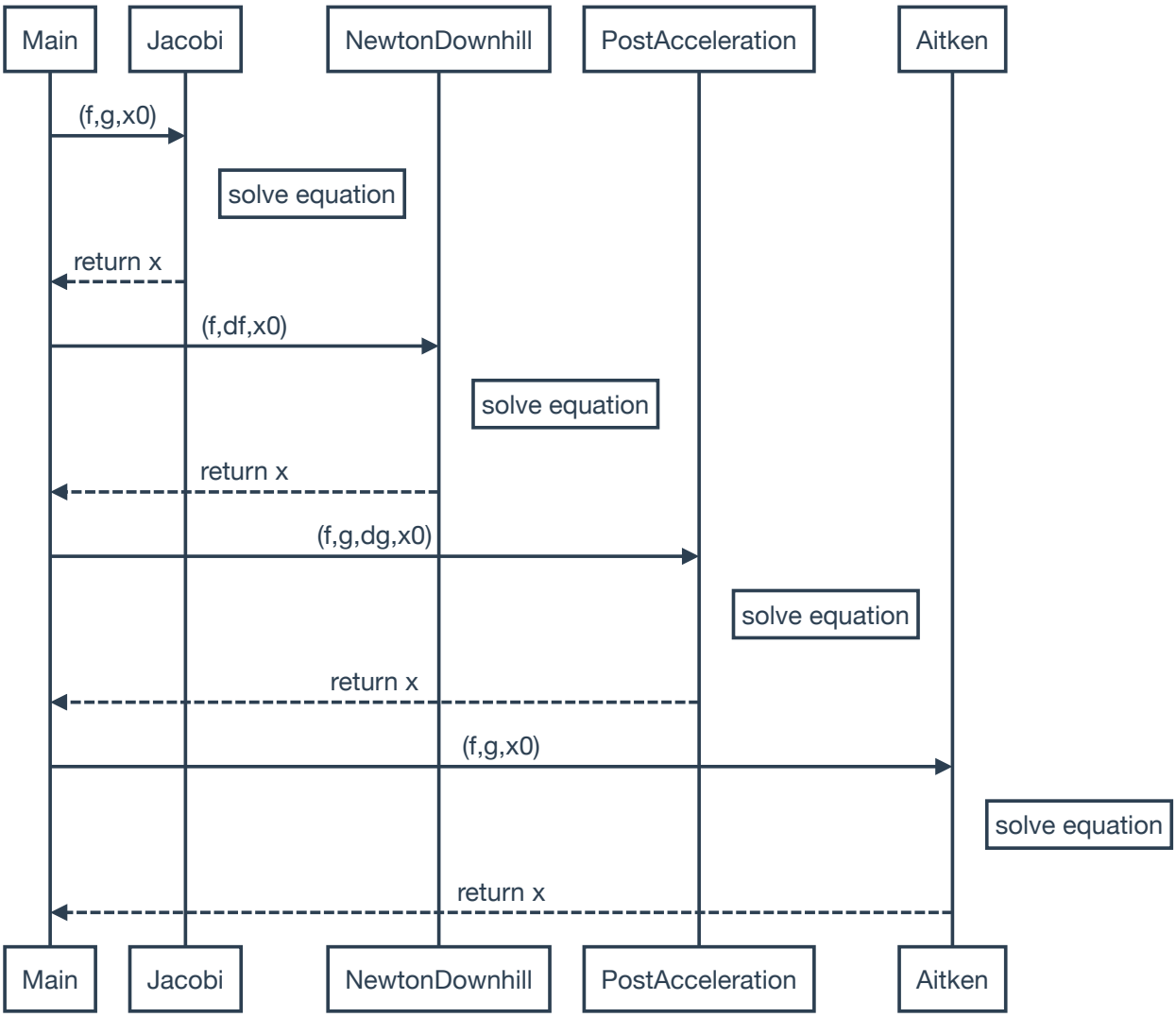
计算物理作业报告

报告主要内容

> 分别用雅科比迭代，牛顿下山法迭代，事后加速法迭代以及Aitken迭代等四种方法计算函数
>
> $$f(x)=\frac{x^3}{3}-x=0$$
>
> 的根。

Main program:

```
program main
implicit none
integer,parameter :: ikind=selected_real_kind(p=15)
real (kind=ikind),external :: f,df,g1,g2
real,external :: dg1
call Jacobi(f,g1,1.5)
call Jacobi(f,g2,1.5)
call Jacobi(f,g2,-1.5)
call NewtonDownhill(f,df,1.5)
call NewtonDownhill(f,df,0.5)
call PostAcceleration(f,g1,dg1,0.5)
call Aitken(f,g1,1.5)
end program main
```



The equations:

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

$$f'(x) = x^2 - 1$$

$$g_1(x) = \frac{x^3}{3}$$

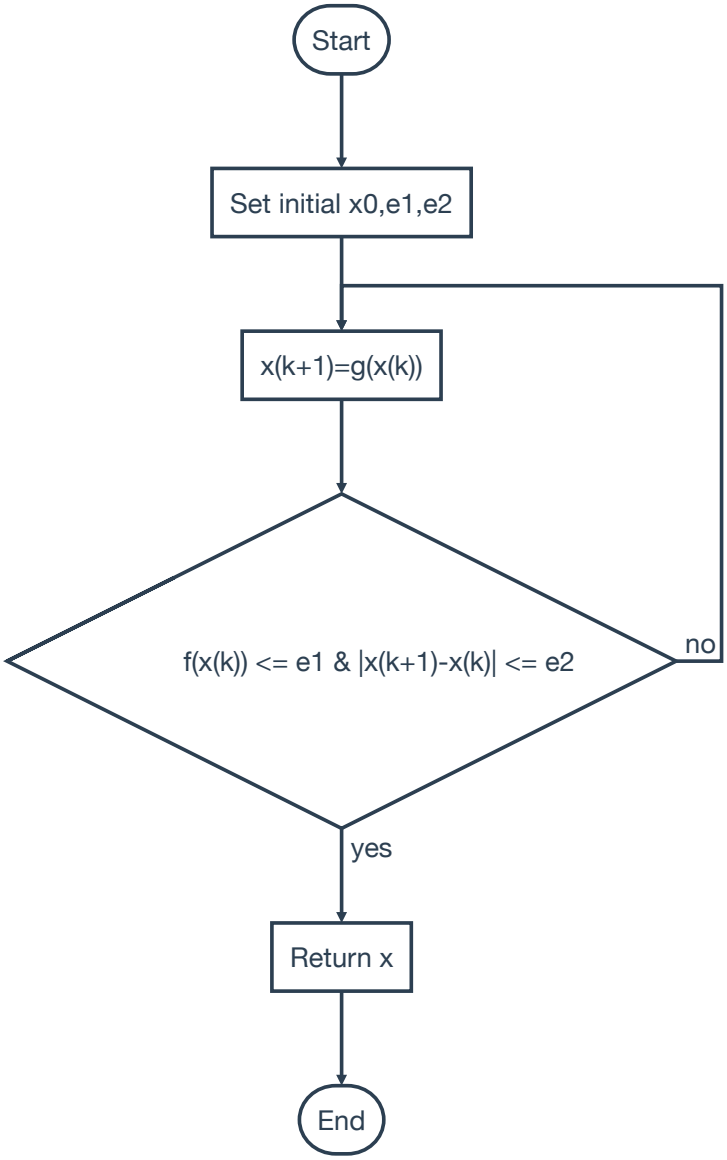
$$g_2(x) = \sqrt[3]{3x}$$

$$g_1'(x) = x^2$$

```
!================= Functions =================
function f(x)
implicit none
integer,parameter :: ikind=selected_real_kind(p=15)
real (kind=ikind) :: x,f
f=x*x*x/3.0_ikind-x
end function f

function df(x)
implicit none
integer,parameter :: ikind=selected_real_kind(p=15)
real (kind=ikind) :: x,df
df=x*x-1
end function df

function g1(x)
implicit none
integer,parameter :: ikind=selected_real_kind(p=15)
real (kind=ikind) :: x,f,g1
g1=f(x)+x
end function g1

function dg1(x)
implicit none
real :: x,dg1
dg1=x*x
end function dg1

function g2(x)
implicit none
integer,parameter :: ikind=selected_real_kind(p=15)
real (kind=ikind) :: x,g2
g2=sign( abs(3.0_ikind*x)**(1.0_ikind/3._ikind), x)
end function g2
```

Jacobi Iteration Method

流程图


Start
Set initial x0,e1,e2
x(k+1)=g(x(k))
f(x(k)) <= e1 & |x(k+1)-x(k)| <= e2
no
yes
Return x
End

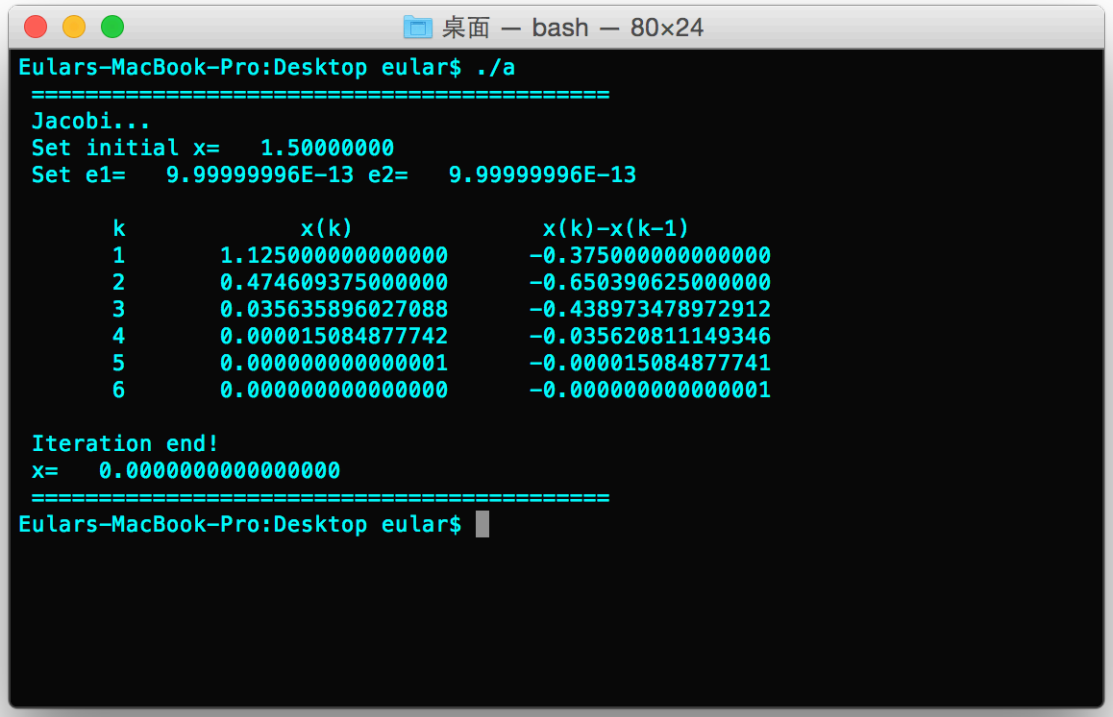
The code

```
!=============== Subroutines ===============
! Jacobi Method
! f: functions
! g: functions
! x0: initial value x
subroutine Jacobi(f,g,x0)
implicit none
integer,parameter :: ikind=selected_real_kind(p=15)
real (kind=ikind),external :: f,g
real (kind=ikind) :: x,x_
real :: x0,e1,e2
integer :: k,k_max
print *,'=========================================='
print *,'Jacobi...'
!Select proper initial value x0
x=x0
print *,'Set initial x=',x0
e1=0.000000000001
e2=0.000000000001
print *,'Set e1=',e1,'e2=',e2
print *
k=0
k_max=100
write(*,"(5x,'  k',5x,'        x(k)',10x,'    x(k)-x(k-1)')")
do
    k=k+1
    x_=x
    x=g(x)
    write(*,100) k,x,x-x_
    100 format(5x,i3,5x,f19.15,5x,f19.15)

    if (f(x)<=e1 .and. abs(x-x_)<=e2) then
        print *
        print *,'Iteration end!'
        print *,'x=',x
        exit
    else if (k>=k_max) then
        write(*,"(5x,'...')")
        write(*,"(5x,'Too many iterations!')")
        exit
    end if
end do
end subroutine Jacobi
```

The result

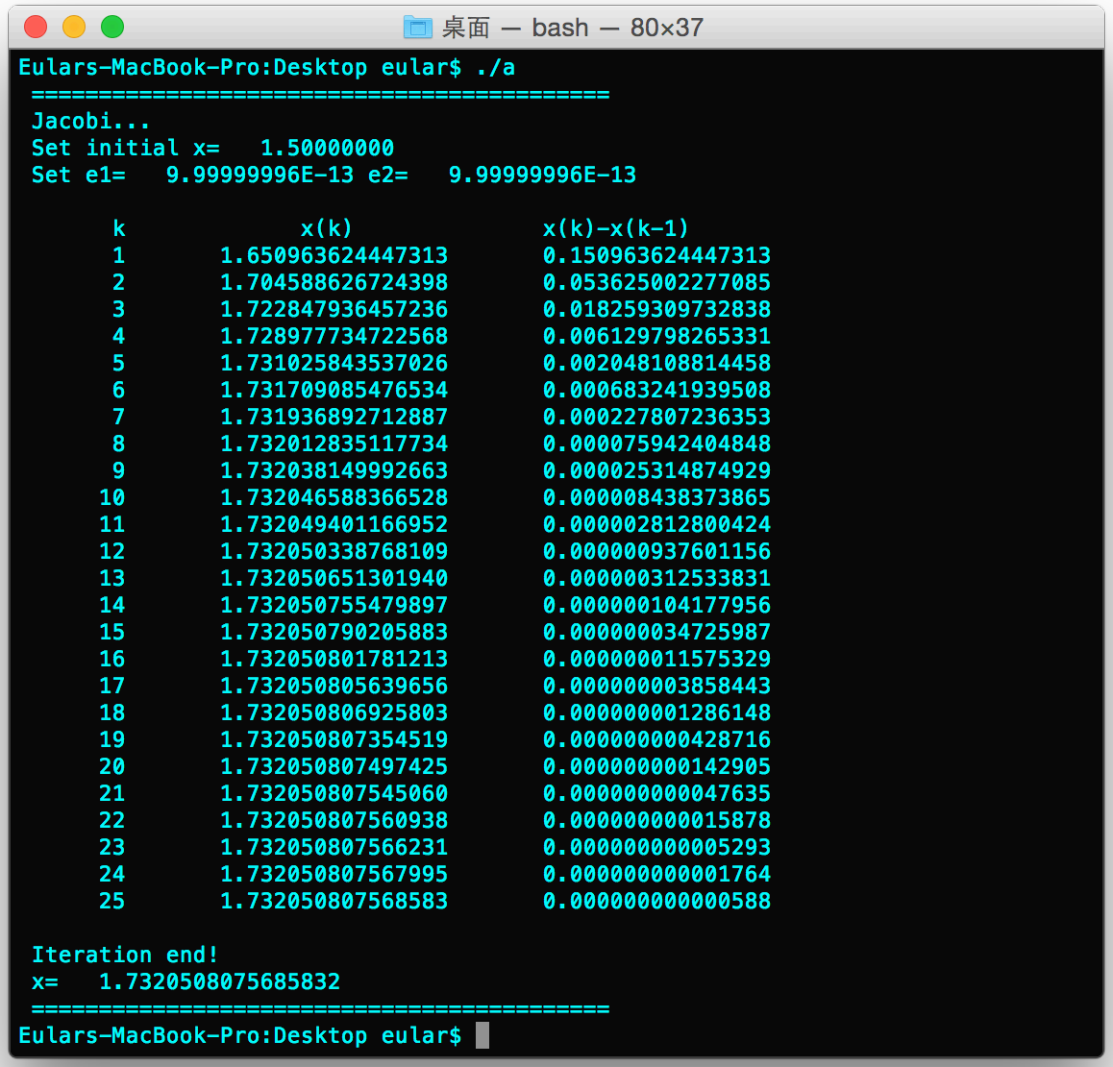
$$x_0 = 1.5$$

$$g(x) = \frac{x^3}{3}$$



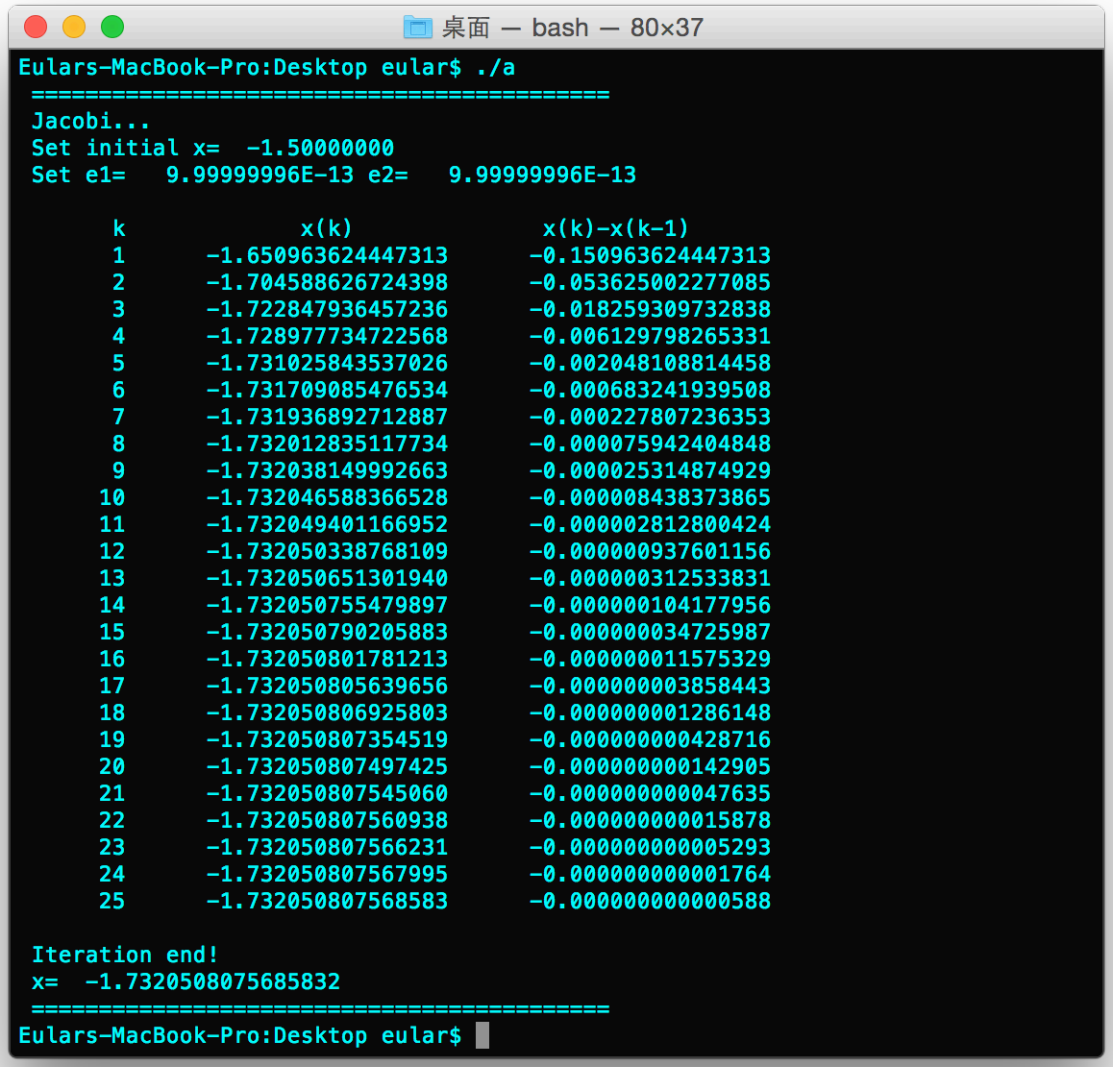
$$x_0 = 1.5$$

$$g(x) = \sqrt[3]{3x}$$



$$x_0 = -1.5$$

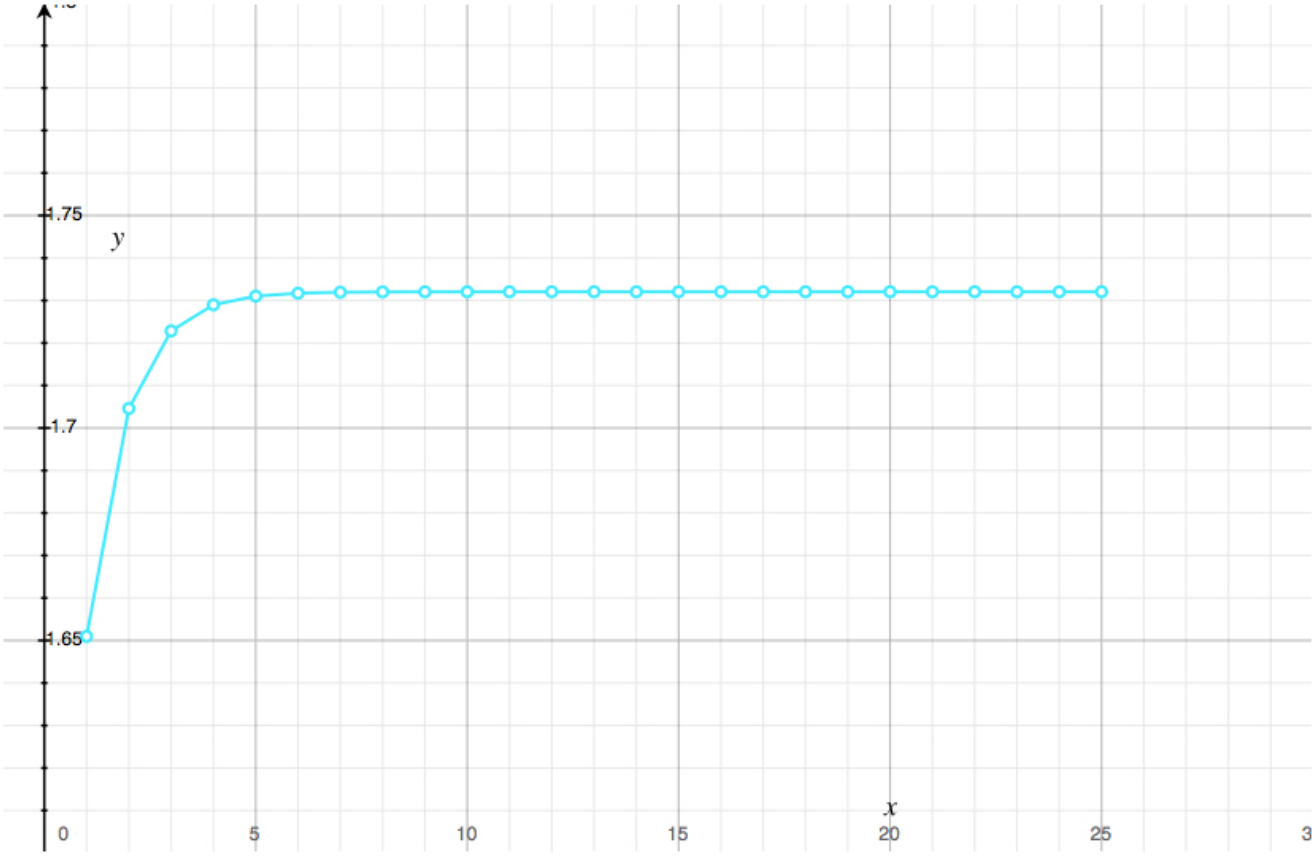
$$g(x) = \sqrt[3]{3x}$$



图例:

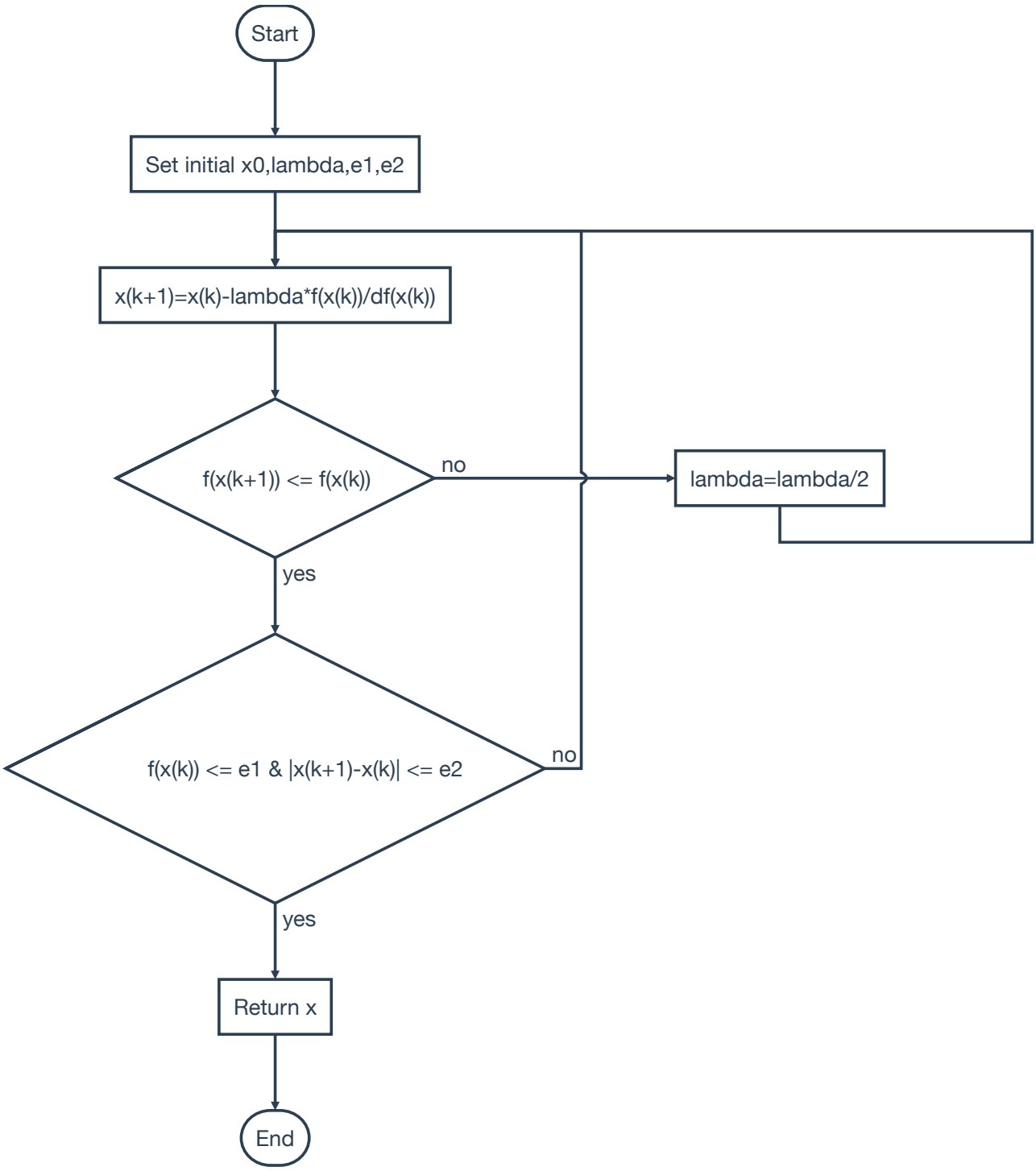
$$x_0 = 1.5$$

$$g(x) = \sqrt[3]{3x}$$



Newton Downhill Method

流程图


Start
Set initial x0,lambda,e1,e2
x(k+1)=x(k)-lambda*f(x(k))/df(x(k))
f(x(k+1)) <= f(x(k))
no
lambda=lambda/2
yes
f(x(k)) <= e1 & |x(k+1)-x(k)| <= e2
no
yes
Return x
End

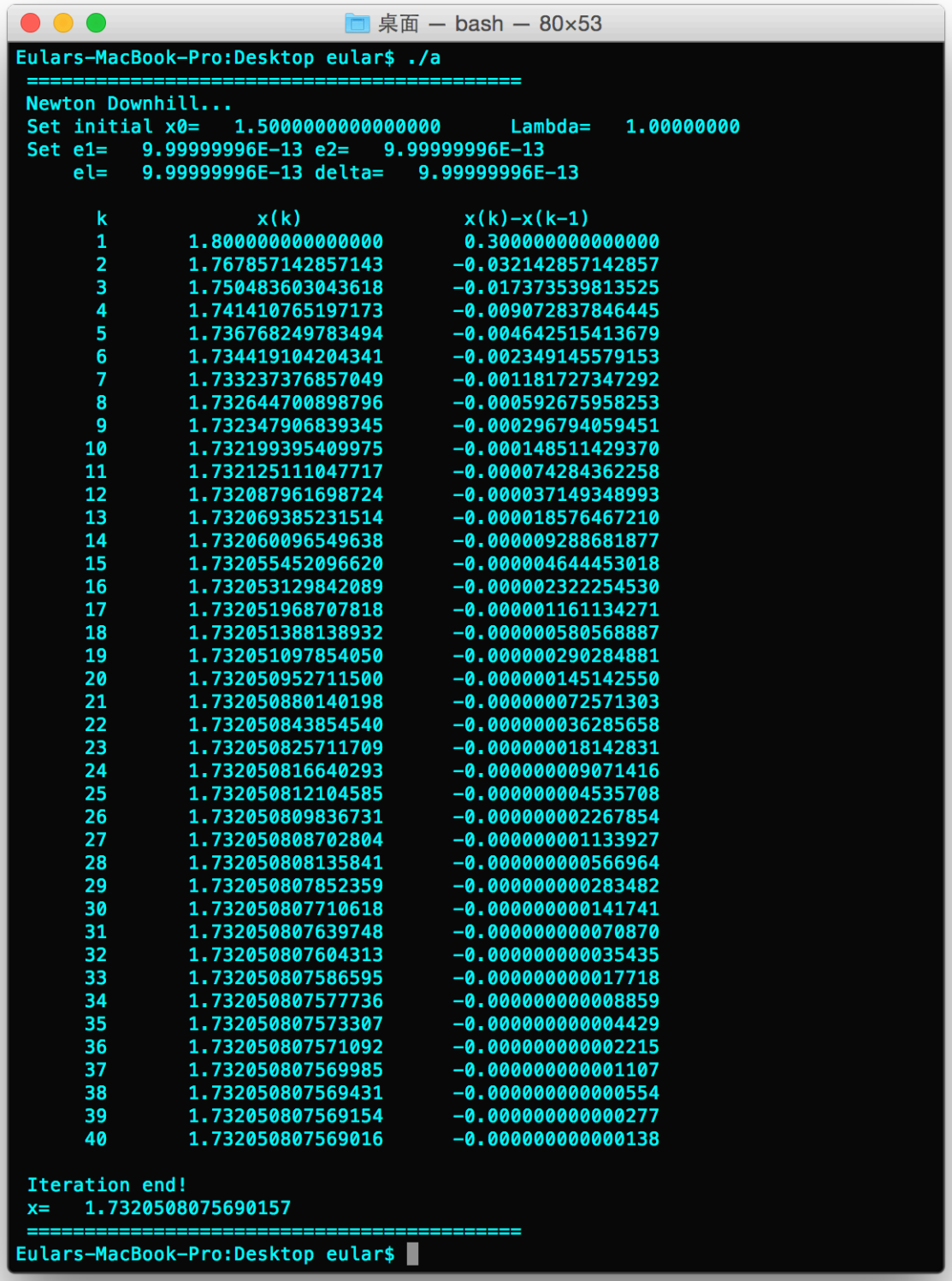
The code

```
!=========================================
! Newton Downhill Method
! f: functions
! df: functions
! x0: initial value x
subroutine NewtonDownhill(f,df,x0)
implicit none
integer,parameter :: ikind=selected_real_kind(p=15)
real (kind=ikind),external :: f,df
real (kind=ikind) :: x,x_
real :: x0,delta,lambda,e1,e2,el
integer :: k
print *,'========================================='
print *,'Newton Downhill...'
x=x0
lambda=1
print *,'Set initial x0=',x,'Lambda=',lambda
e1=0.000000000001
e2=0.000000000001
el=0.000000000001
delta=0.000000000001
print *,'Set e1=',e1,'e2=',e2
print *,'    el=',el,'delta=',delta
print *
k=0
write(*,"(5x,'  k',5x,'        x(k)',10x,'    x(k)-x(k-1)')")
do
    k=k+1
    x_=x
    x=x-lambda*f(x)/df(x)
    write(*,100) k,x,x-x_
    100 format(5x,i3,5x,f19.15,5x,f19.15)

    if (f(x)<=f(x_)) then
        if (f(x_)<=e1 .and. abs(x-x_)<=e2) then
            print *
            print *,'Iteration end!'
            print *,'x=',x
            exit
        end if
    else
        if (lambda > el) then
            lambda=lambda/2
        else
            x=x_+delta
        end if
    end if
end do
print *,'========================================='
end subroutine NewtonDownhill
```

The result

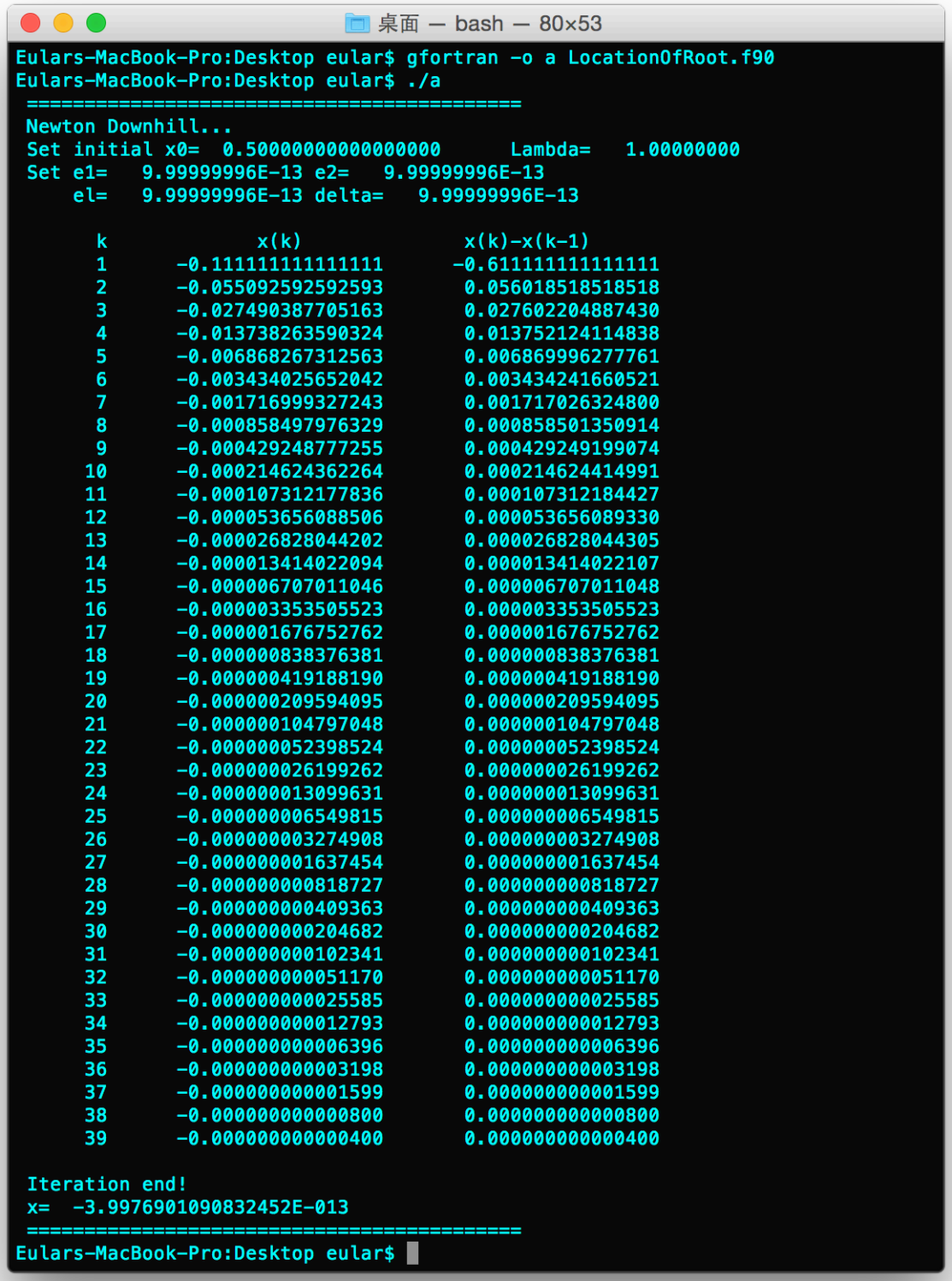
$$x_0 = 1.5$$

$\lambda = 1$

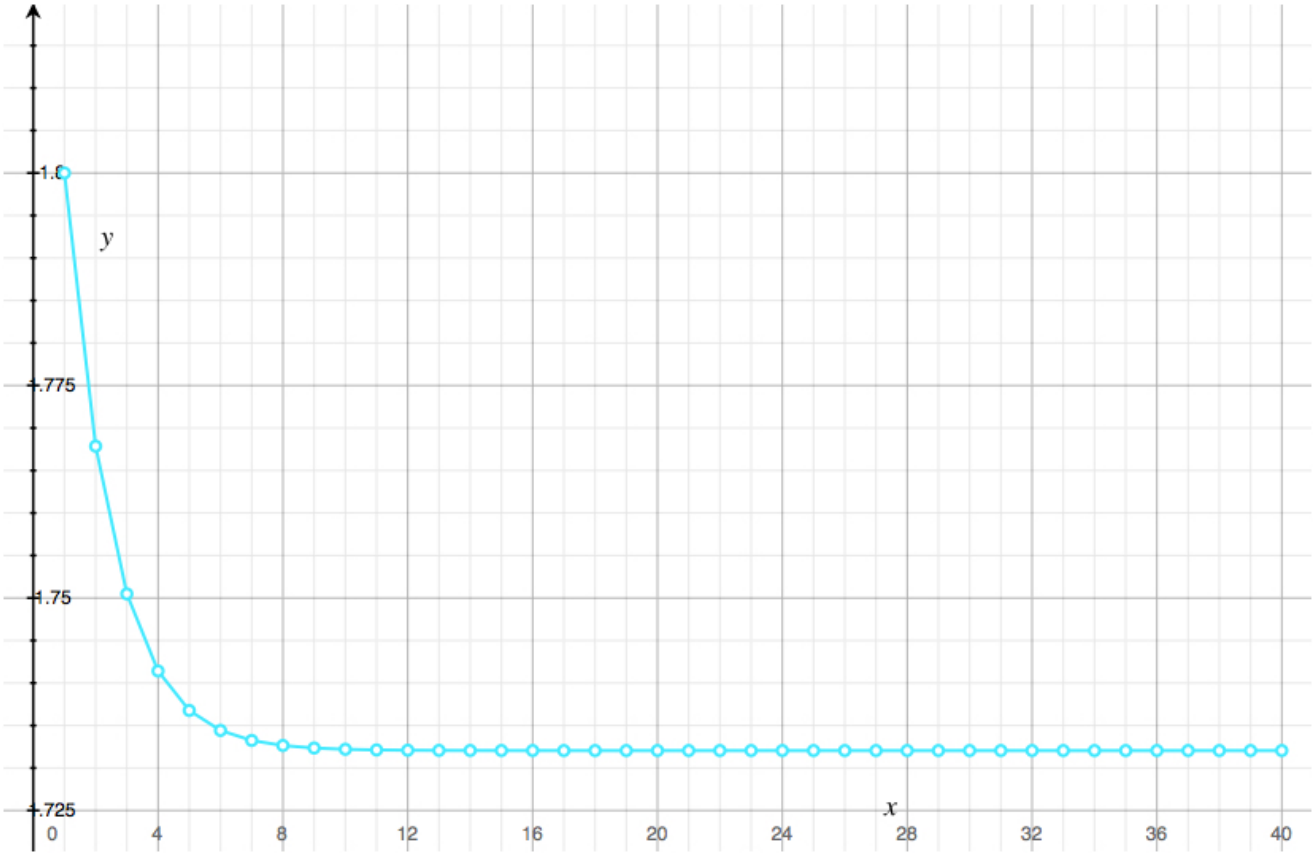


$x_0 = 0.5$

$\lambda = 1$

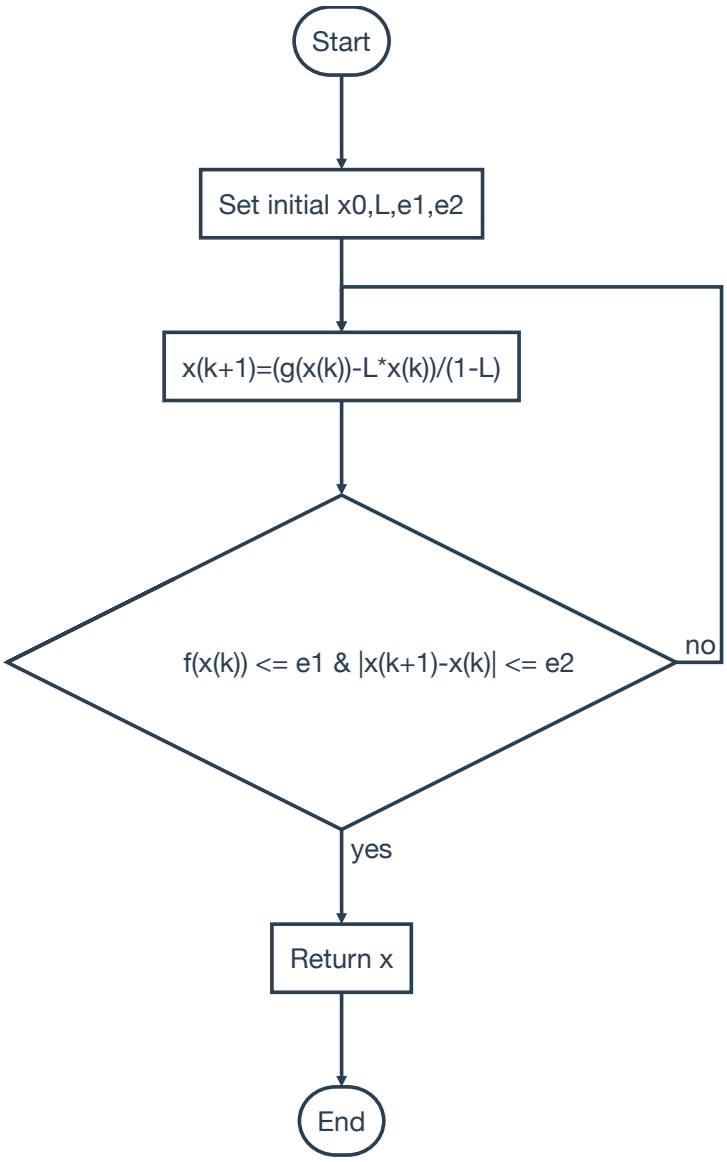

桌面 — bash — 80×53
Eulars-MacBook-Pro:Desktop eular$ gfortran -o a LocationOfRoot.f90
Eulars-MacBook-Pro:Desktop eular$ ./a
 ==
 Newton Downhill...
 Set initial x0= 0.50000000000000000   Lambda=   1.00000000
 Set e1=   9.99999996E-13 e2=   9.99999996E-13
     el=   9.99999996E-13 delta=   9.99999996E-13
       k   x(k)   x(k)-x(k-1)
       1   -0.111111111111111   -0.611111111111111
       2   -0.055092592592593   0.056018518518518
       3   -0.027490387705163   0.027602204887430
       4   -0.013738263590324   0.013752124114838
       5   -0.006868267312563   0.006869996277761
       6   -0.003434025652042   0.003434241660521
       7   -0.001716999327243   0.001717026324800
       8   -0.000858497976329   0.000858501350914
       9   -0.000429248777255   0.000429249199074
      10   -0.000214624362264   0.000214624414991
      11   -0.000107312177836   0.000107312184427
      12   -0.000053656088506   0.000053656089330
      13   -0.000026828044202   0.000026828044305
      14   -0.000013414022094   0.000013414022107
      15   -0.000006707011046   0.000006707011048
      16   -0.000003353505523   0.000003353505523
      17   -0.000001676752762   0.000001676752762
      18   -0.000000838376381   0.000000838376381
      19   -0.000000419188190   0.000000419188190
      20   -0.000000209594095   0.000000209594095
      21   -0.000000104797048   0.000000104797048
      22   -0.000000052398524   0.000000052398524
      23   -0.000000026199262   0.000000026199262
      24   -0.000000013099631   0.000000013099631
      25   -0.000000006549815   0.000000006549815
      26   -0.000000003274908   0.000000003274908
      27   -0.000000001637454   0.000000001637454
      28   -0.000000000818727   0.000000000818727
      29   -0.000000000409363   0.000000000409363
      30   -0.000000000204682   0.000000000204682
      31   -0.000000000102341   0.000000000102341
      32   -0.000000000051170   0.000000000051170
      33   -0.000000000025585   0.000000000025585
      34   -0.000000000012793   0.000000000012793
      35   -0.000000000006396   0.000000000006396
      36   -0.000000000003198   0.000000000003198
      37   -0.000000000001599   0.000000000001599
      38   -0.000000000000800   0.000000000000800
      39   -0.000000000000400   0.000000000000400
 Iteration end!
 x= -3.9976901090832452E-013
 ==
Eulars-MacBook-Pro:Desktop eular$

图例



Post Accelerating Method

流程图


Start
Set initial x0,L,e1,e2
x(k+1)=(g(x(k))-L*x(k))/(1-L)
f(x(k)) <= e1 & |x(k+1)-x(k)| <= e2
no
yes
Return x
End

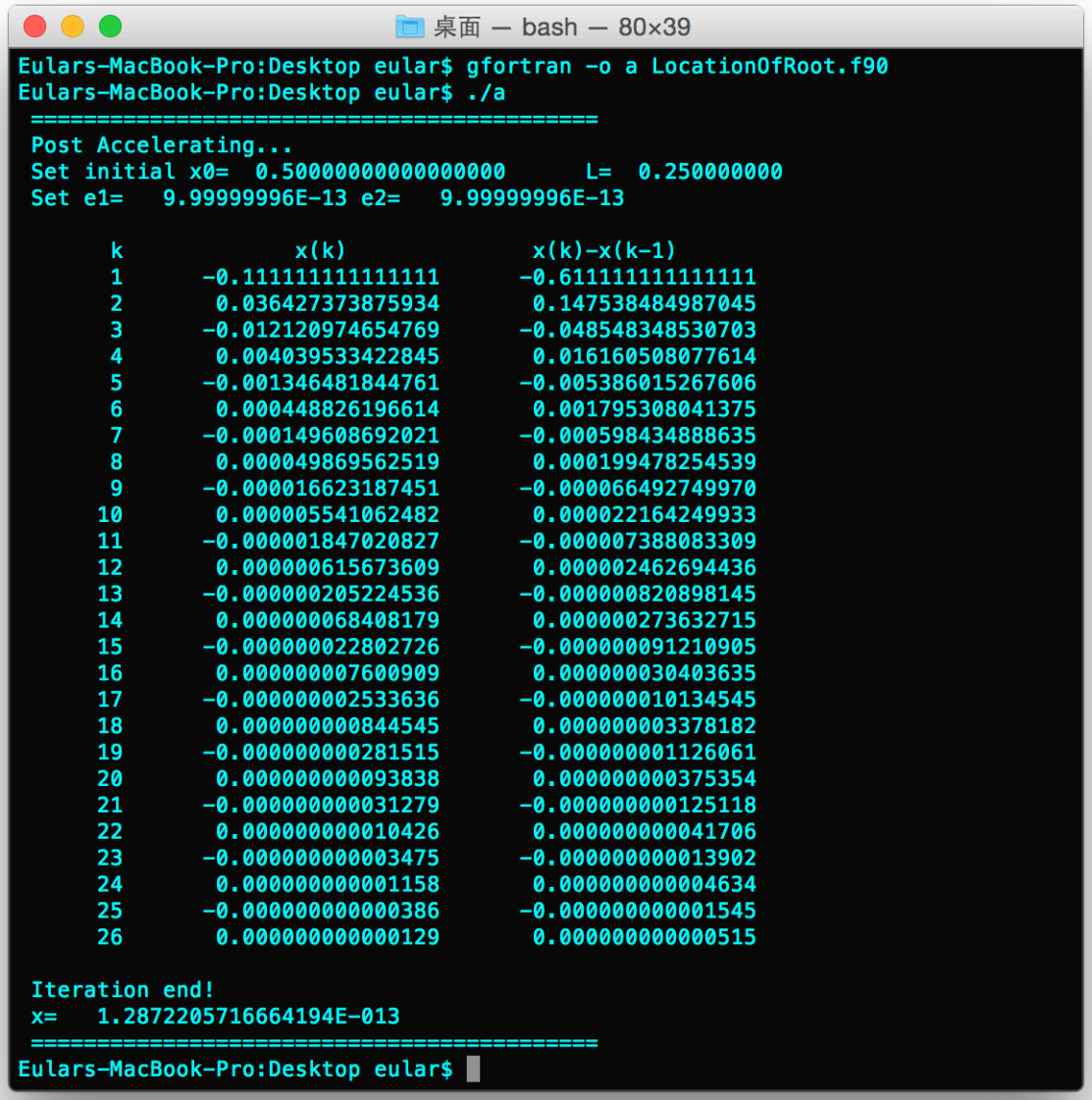
The code:

```
!=========================================
! Post Accelerating Method
! f: functions
! g: functions
! dg: functions
! x0: initial value x
subroutine PostAcceleration(f,g,dg,x0)
implicit none
integer,parameter :: ikind=selected_real_kind(p=15)
real (kind=ikind),external :: f,g
real (kind=ikind) :: x,x_
real,external :: dg
real :: x0,e1,e2,L
integer :: k,k_max
print *,'========================================='
print *,'Post Accelerating...'
x=x0
L=dg(x0)
print *,'Set initial x0=',x,'L=',L
e1=0.000000000001
e2=0.000000000001
print *,'Set e1=',e1,'e2=',e2
print *
k=0
k_max=100
write(*,"(5x,'  k',5x,'        x(k)',10x,'    x(k)-x(k-1)')")
do
    k=k+1
    x_=x
    x=(g(x)-L*x)/(1-L)
    write(*,100) k,x,x-x_
    100 format(5x,i3,5x,f19.15,5x,f19.15)

    if (f(x_)<=e1 .and. abs(x-x_)<=e2) then
        print *
        print *,'Iteration end!'
        print *,'x=',x
        exit
    else if (k>=k_max) then
        write(*,"(5x,'...')")
        write(*,"(5x,'Too many iterations!')")
        exit
    end if
end do
print *,'========================================='
end subroutine PostAcceleration
```

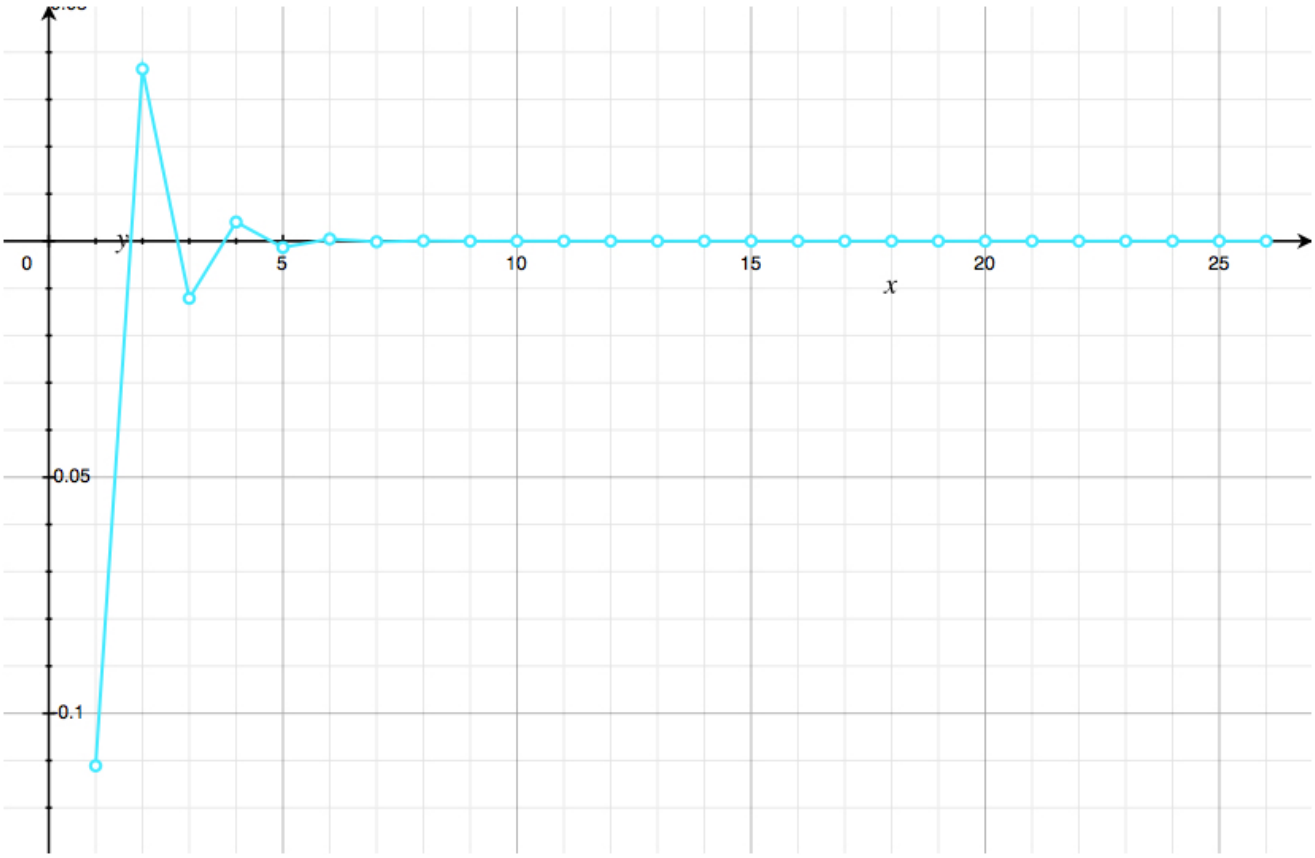
The result

$$x_0 = 0.5$$

$$L = 0.25$$

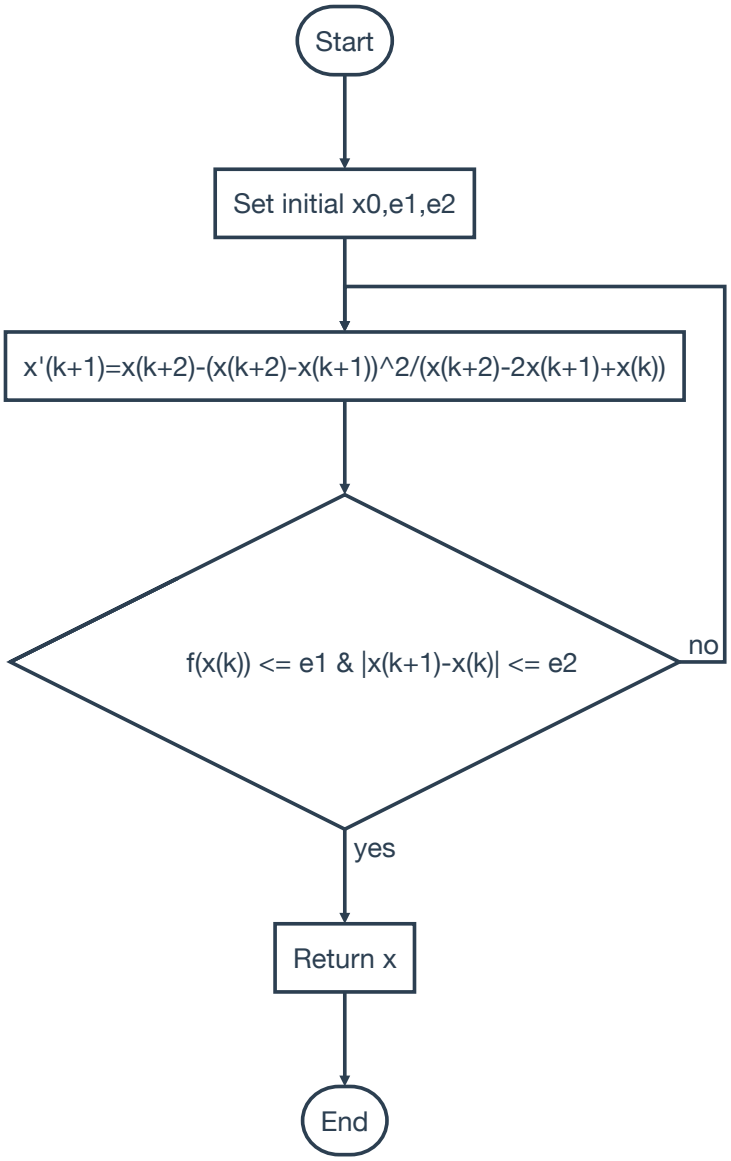

桌面 — bash — 80×39
Eulars-MacBook-Pro:Desktop eular$ gfortran -o a LocationOfRoot.f90
Eulars-MacBook-Pro:Desktop eular$ ./a
 ===
 Post Accelerating...
 Set initial x0= 0.50000000000000000   L= 0.250000000
 Set e1=   9.99999996E-13 e2=   9.99999996E-13
       k   x(k)   x(k)-x(k-1)
       1   -0.111111111111111   -0.611111111111111
       2   0.036427373875934   0.147538484987045
       3   -0.012120974654769   -0.048548348530703
       4   0.004039533422845   0.016160508077614
       5   -0.001346481844761   -0.005386015267606
       6   0.000448826196614   0.001795308041375
       7   -0.000149608692021   -0.000598434888635
       8   0.000049869562519   0.000199478254539
       9   -0.000016623187451   -0.000066492749970
      10   0.000005541062482   0.000022164249933
      11   -0.000001847020827   -0.000007388083309
      12   0.000000615673609   0.000002462694436
      13   -0.000000205224536   -0.000000820898145
      14   0.000000068408179   0.000000273632715
      15   -0.000000022802726   -0.000000091210905
      16   0.000000007600909   0.000000030403635
      17   -0.000000002533636   -0.000000010134545
      18   0.000000000844545   0.000000003378182
      19   -0.000000000281515   -0.000000001126061
      20   0.000000000093838   0.000000000375354
      21   -0.000000000031279   -0.000000000125118
      22   0.000000000010426   0.000000000041706
      23   -0.000000000003475   -0.000000000013902
      24   0.000000000001158   0.000000000004634
      25   -0.000000000000386   -0.000000000001545
      26   0.000000000000129   0.000000000000515
 Iteration end!
 x=   1.2872205716664194E-013
 ===
Eulars-MacBook-Pro:Desktop eular$

图例



Aitken Method

流程图


Start
Set initial x0,e1,e2
x'(k+1)=x(k+2)-(x(k+2)-x(k+1))^2/(x(k+2)-2x(k+1)+x(k))
f(x(k)) <= e1 & |x(k+1)-x(k)| <= e2
no
yes
Return x
End

The code:

```
!=========================================
! Aitken Method
! f: functions
! g: functions
! x0: initial value x
subroutine Aitken(f,g,x0)
implicit none
integer,parameter :: ikind=selected_real_kind(p=15)
real (kind=ikind),external :: f,g
real (kind=ikind) :: x,x_
real :: x0,e1,e2
integer :: k,k_max
print *,'========================================='
print *,'Aitken...'
x=x0
print *,'Set initial x0=',x
e1=0.000000000001
e2=0.000000000001
print *,'Set e1=',e1,'e2=',e2
print *
k=0
k_max=100
write(*,"(5x,'  k',5x,'        x(k)',10x,'    x(k)-x(k-1)')")
do
    k=k+1
    x_=x
    x=g(x_)
    x=g(x)-(g(x)-x)**2/(g(x)-2*x+x_)
    write(*,100) k,x,x-x_
    100 format(5x,i3,5x,f19.15,5x,f19.15)

    if (f(x)<=e1 .and. abs(x-x_)<=e2) then
        print *
        print *,'Iteration end!'
        print *,'x=',x
        exit
    else if (k>=k_max) then
        write(*,"(5x,'...')")
        write(*,"(5x,'Too many iterations!')")
        exit
    end if
end do
print *,'========================================='
end subroutine Aitken
```

The result

$$x_0 = 1.5$$

```
桌面 — bash — 80×25
Eulars-MacBook-Pro:Desktop eular$ ./a
 ==========================================
 Aitken...
 Set initial x0=   1.5000000000000000
 Set e1=   9.99999996E-13 e2=   9.99999996E-13

       k             x(k)              x(k)-x(k-1)
       1       2.010638297872340       0.510638297872340
       2       1.859069249599888      -0.151569048272452
       3       1.765106642447591      -0.093962607152298
       4       1.734704213224771      -0.030402429222820
       5       1.732068997319653      -0.002635215905118
       6       1.732050808428462      -0.000018188891191
       7       1.732050807568877      -0.000000000859585
       8       1.732050807568877       0.000000000000000

 Iteration end!
 x=   1.7320508075688772
 ==========================================
Eulars-MacBook-Pro:Desktop eular$
```

图例