The second schoolwork of Computational Physics

万炫均 物理 1701 U201710170

Description of this chapter:

For this chapter, we try various of computing methods to find out roots of functions, which is a quite basic problem in mathematic.

- **Description of the problem**

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

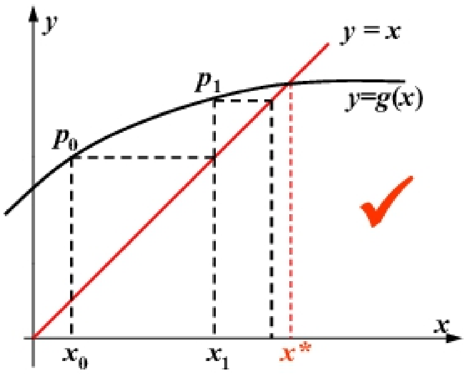
$$f(x)=\frac{x^3}{3}-x=0$$

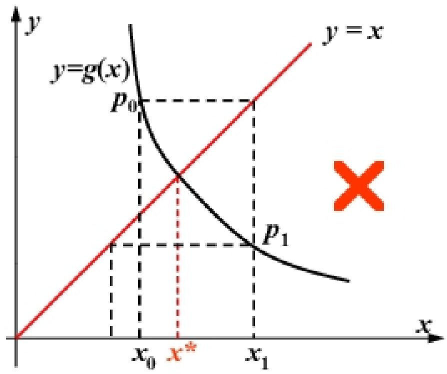
and compare their performances (speed and error)

- **Formula to use**

  Here we will use four methods in total:
  Jacobi method,
  Newtown downhill method,
  Post acceleration method,
  Aitken method.

Jacobi Iteration $\quad x_{k+1} = g(x_k)$





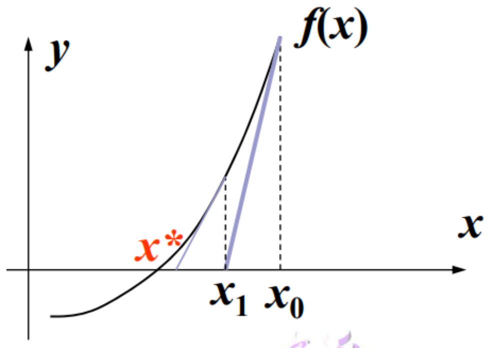
4 *Newton* Iteration Method

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \cdots$$

$$\approx f(x_0) + f'(x_0)(x - x_0) = 0$$

$$x = x_0 - \frac{f(x_0)}{f'(x_0)}$$

$$\boxed{x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}}$$



Post Accelerating Method

$$x_{k+1} = g(x_k), \quad x^* = g(x^*)$$

$$x_{k+1} - x^* = g(x_k) - g(x^*)$$
$$= g'(\xi)(x_k - x^*)$$
$$\approx L(x_k - x^*)$$

$$x^* = \frac{x_{k+1} - Lx_k}{1-L} \quad \Rightarrow \quad \boxed{x'_{k+1} = \frac{x_{k+1} - Lx_k}{1-L}}$$

CPCM 22

Aitken Accelerating Method

$$x_{k+1} - x^* = L(x_k - x^*)$$

$$x_{k+2} - x^* = L(x_{k+1} - x^*)$$

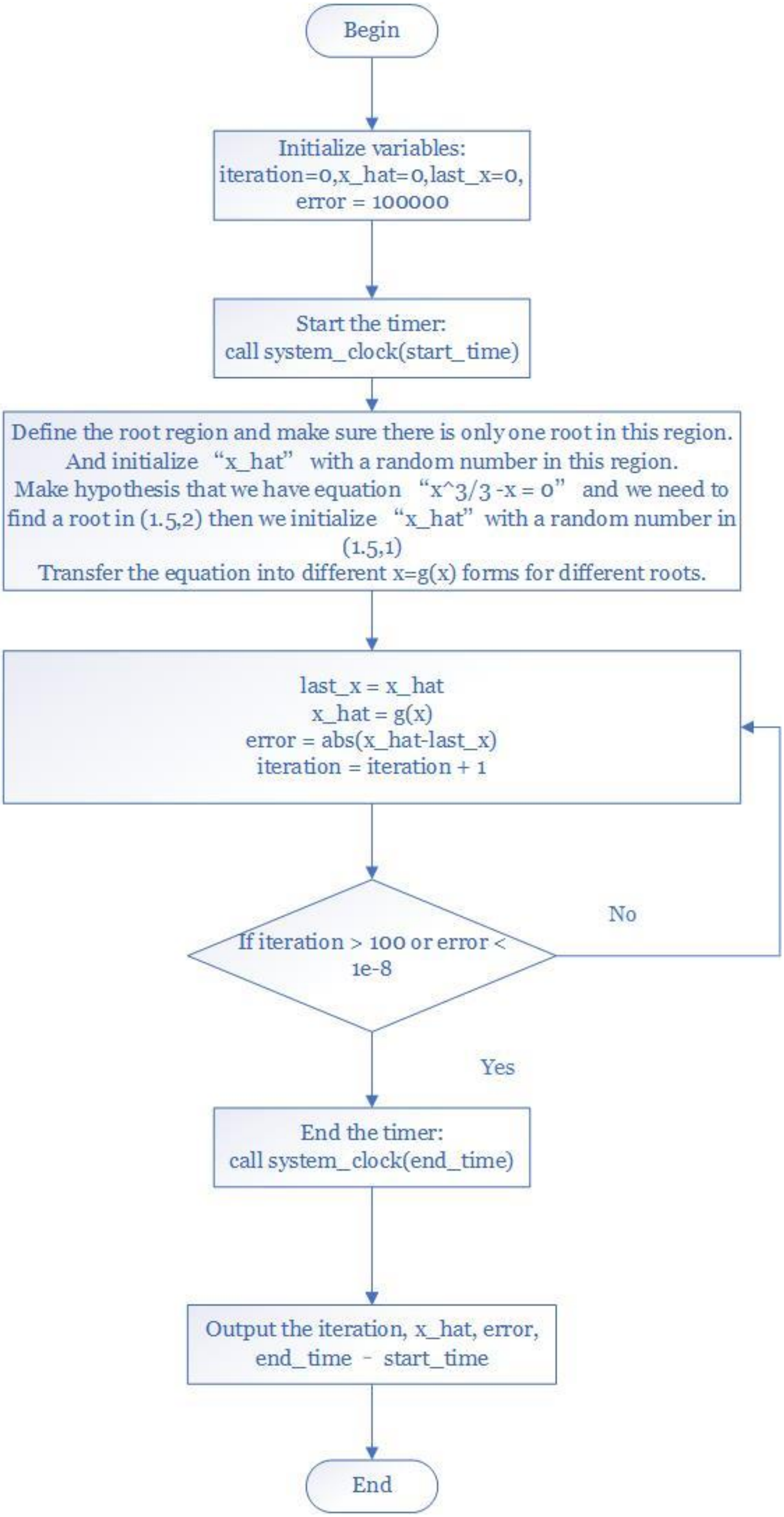
$$\frac{x_{k+1} - x^*}{x_{k+2} - x^*} = \frac{x_k - x^*}{x_{k+1} - x^*}$$

$$\boxed{x'_{k+1} = x^* = x_{k+2} - \frac{\left(x_{k+2} - x_{k+1}\right)^2}{x_{k+2} - 2x_{k+1} + x_k}}$$

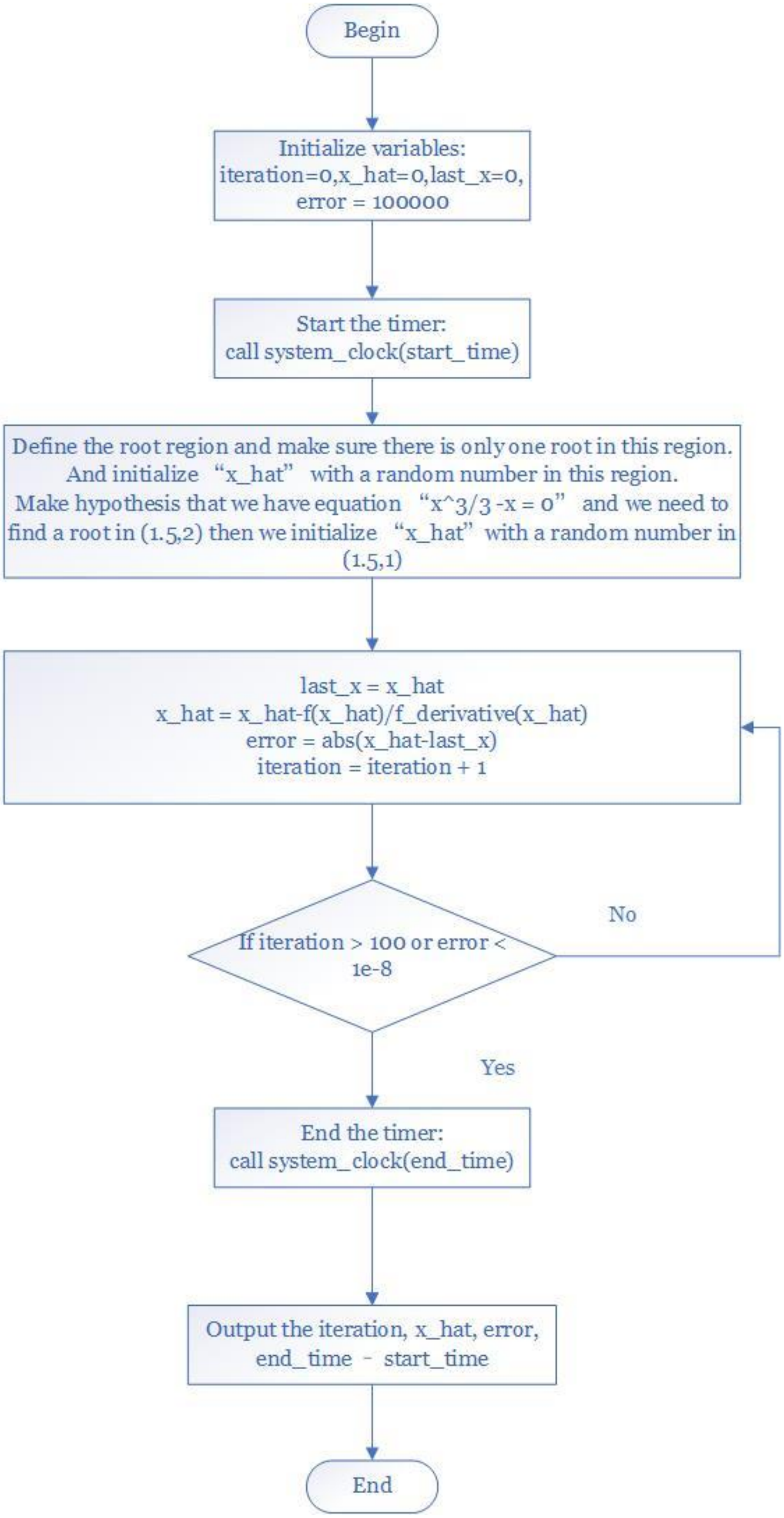
here $x_{k+1} = g(x_k) \quad x_{k+2} = g(x_{k+1})$

CPCM 24

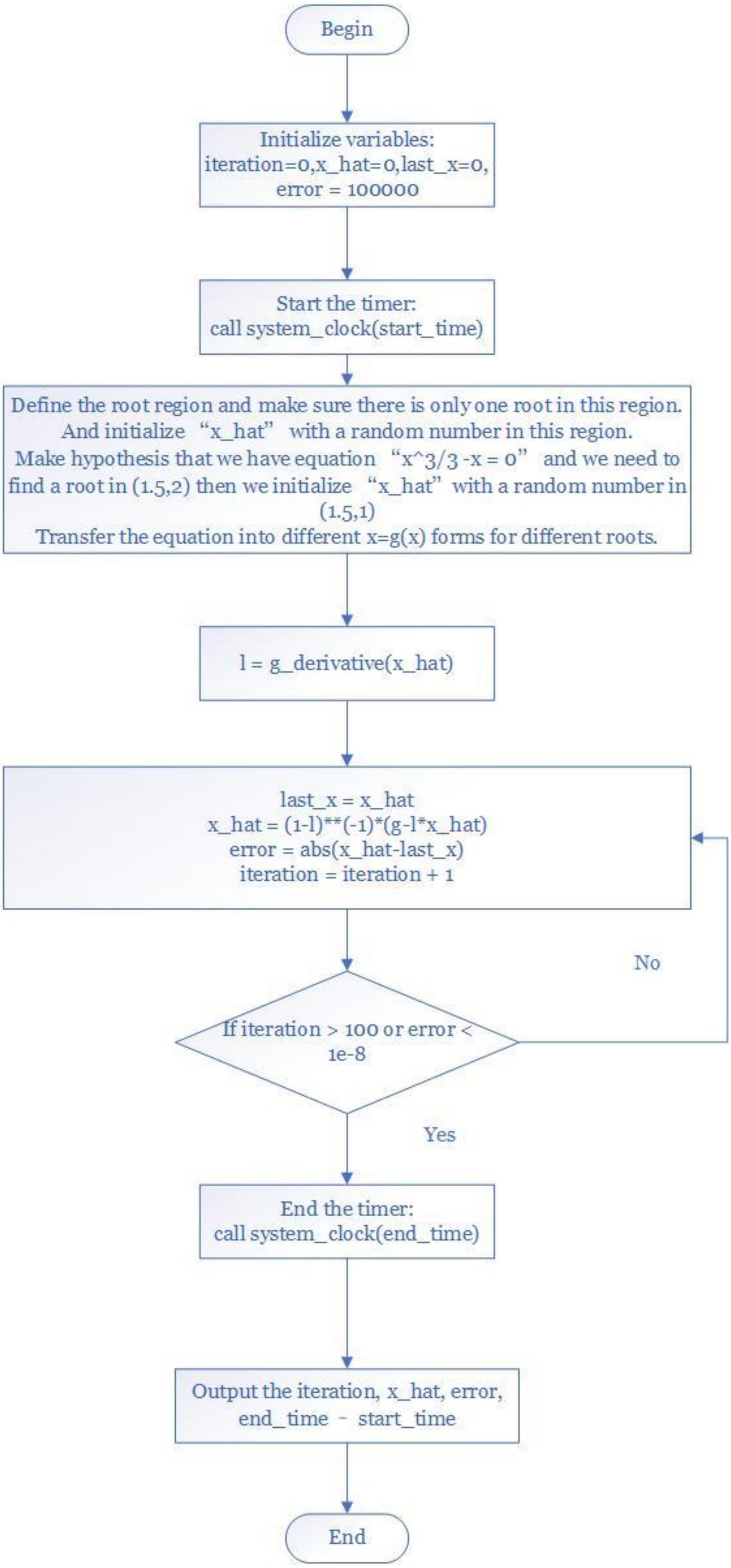
- **Flow chart**
  1. **Jacobi Method Flow Chart**


Begin
Initialize variables:
iteration=0,x_hat=0,last_x=0,
error = 100000
Start the timer:
call system_clock(start_time)
Define the root region and make sure there is only one root in this region.
And initialize "x_hat" with a random number in this region.
Make hypothesis that we have equation "x^3/3 -x = 0" and we need to find a root in (1.5,2) then we initialize "x_hat" with a random number in (1.5,1)
Transfer the equation into different x=g(x) forms for different roots.
last_x = x_hat
x_hat = g(x)
error = abs(x_hat-last_x)
iteration = iteration + 1
If iteration > 100 or error < 1e-8
No
Yes
End the timer:
call system_clock(end_time)
Output the iteration, x_hat, error,
end_time - start_time
End

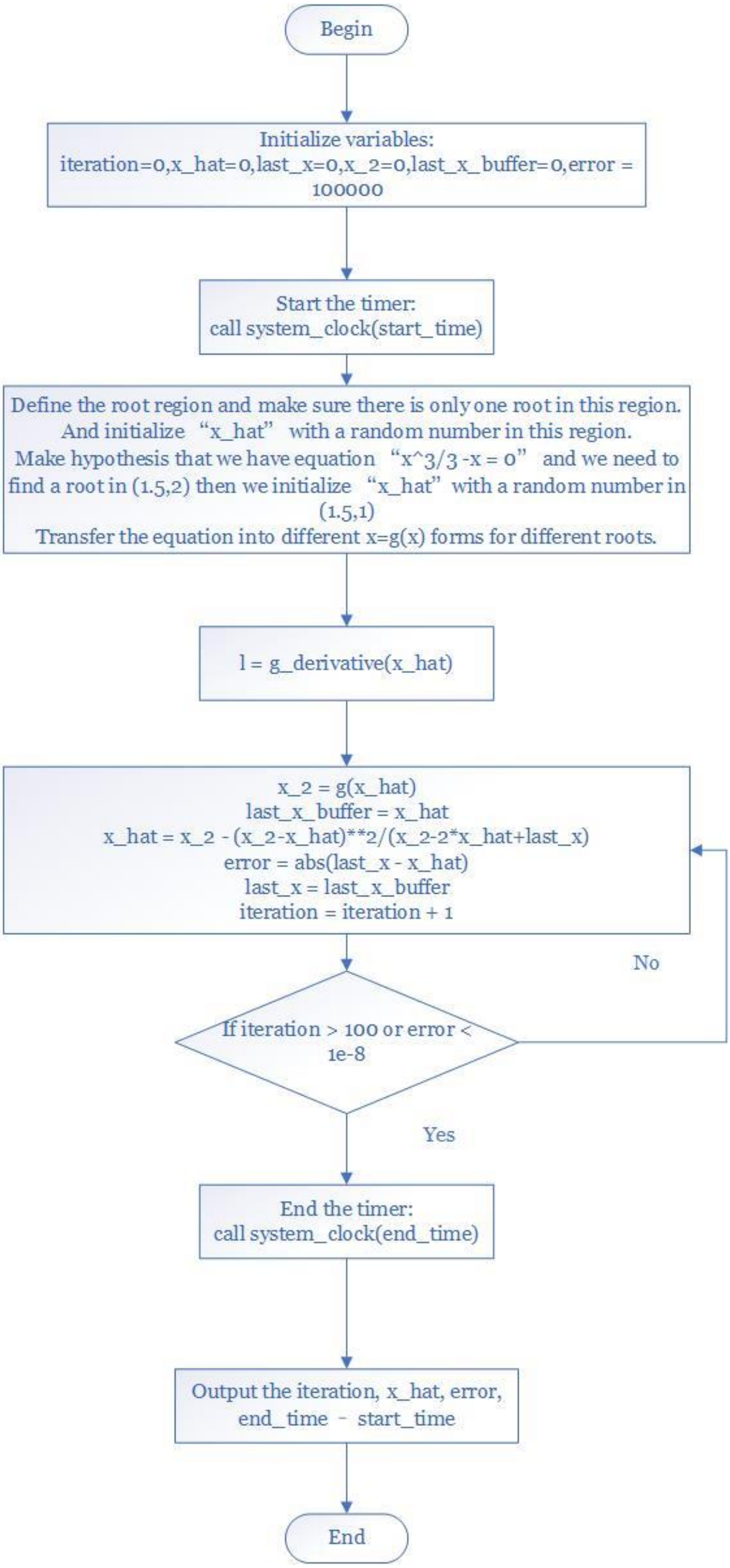
2. Newton Downhill Method Flow Chart



3. Post Accelerating Method Flow Chart



4. Aitken Method Flow Chart



- **Source Code**

  Here is the code. I have compressed the four methods of finding all the three roots into one program so it is easy to run the program.

```
program Root
    real*8 :: x_hat,error
    integer :: iteration

    !Clock to measure
    integer*8 :: start_time,end_time

    !Procedures for different methods
    do j = 1,4
        !Start the system timer
        call system_clock(start_time)
        !Loop for different roots
        print *,"=================================="
        do i = 1,3
            select case(j)
                case(1)
                    call JacobiCalculate(x_hat,error,iteration,i)
                    if (i==1) then
                        print "(a,/)","#Jacobi Method"
                    end if

                case(2)
                    call NewtonCalculate(x_hat,error,iteration,i)

                    if (i==1) then
                        print "(a,/)","#Newton Method"
                    end if
                case(3)
                    call AccCalculate(i, x_hat,error,iteration)
                    if (i==1) then
                        print "(a,/)","#Post Acceleration Method"
                    end if
                case(4)
                    call AitkenCalculate(i, x_hat,error,iteration)
                    if (i==1) then
                        print "(a,/)","#Aitken Method"
                    end if
                case default
                    print "(a,/)","Unknown procedure id!!!"
```

```fortran
            end select

            !Stop the system timer
            call system_clock(end_time)

            print "(a,i8)","Root id",i
            print "(a,f8.4,/,a,es16.3,/,a,i4,/a,es16.3)","Root:",x_hat,"Error:",error,"Iteration:",iteration
            print "(a,i16,/)","Used time:",(end_time-start_time)
        end do
    end do

end program Root


!The core calculation subroutine
subroutine JacobiCalculate(x_hat,error,iteration,root_id)
    implicit none
    real*8 :: g1
    real*8 :: g2
    integer,intent(in):: root_id
    real*8,intent(out) :: x_hat,error
    integer,intent(out) :: iteration

    real*8 :: last_x

    iteration = 0
    x_hat = 0
    last_x = 0
    !Set a large number
    error = 1000000

    do while(abs(error)>1d-8 .and. iteration < 100)
        last_x = x_hat


        !Choose different functions for different roots
        select case(root_id)
            case (1)
                x_hat = g1(x_hat)
                !Set the x a random number in (-1,0)
                if ( iteration == 0 ) then
                    call random_number(x_hat)
                    x_hat = x_hat*(-1)
```

```
                end if
            case (2)
                x_hat = g2(x_hat)
                !Set the x a random number in (1,2)
                if ( iteration == 0 ) then
                    call random_number(x_hat)
                    x_hat = x_hat+1
                end if
            case (3)
                x_hat = g2(x_hat)
                !Set the x a random number in (-2,-1)
                if ( iteration == 0 ) then
                    call random_number(x_hat)
                    x_hat = x_hat*(-1)-1
                end if
        end select

        error = abs(x_hat-last_x)
        iteration = iteration + 1
    end do

end subroutine JacobiCalculate

subroutine NewtonCalculate(x_hat,error,iteration,root_id)
    integer,intent(in) :: root_id
    real*8,intent(out) ::x_hat,error
    integer,intent(out) :: iteration

    real*8 :: f,f_derivative

    real*8 :: last_x


    !Initialize the x with a random number in different regions
    select case(root_id)
            case (1)
                !Set the x a random number in (0,1)
                call random_number(x_hat)
            case (2)
                !Set the x a random number in (1,2)
                call random_number(x_hat)
                x_hat = x_hat+1
            case (3)
                !Set the x a random number in (-2,-1)
                call random_number(x_hat)
```

```fortran
                x_hat = (x_hat+1)*(-1)
    end select
    iteration = 0
    !Set a large number
    error = 10000000

    !Main calculation loop
    do while(abs(error)>1d-8 .and. iteration < 100)
        last_x = x_hat
        x_hat = x_hat-f(x_hat)/f_derivative(x_hat)
        error = abs(last_x - x_hat)
        iteration = iteration + 1
    end do

end subroutine

!Post Acceleration Method
subroutine AccCalculate(root_id, x_hat,error,iteration)
    implicit none
    integer,intent(in) :: root_id
    real*8,intent(out) :: x_hat,error
    integer,intent(out) :: iteration

    real*8 :: g,g1,g2,g1_derivative,g2_derivative,l
    real*8 :: last_x

    !Initialize the x with a random number in different regions
    !Initialize the factor l
    select case(root_id)
            case (1)
                !Set the x a random number in (0,1)
                call random_number(x_hat)
                l = g1_derivative(x_hat)
            case (2)
                !Set the x a random number in (1,2)
                call random_number(x_hat)
                x_hat = x_hat+1
                l = g2_derivative(x_hat)
            case (3)
                !Set the x a random number in (-2,-1)
                call random_number(x_hat)
                x_hat = x_hat*(-1)-1
                l = g2_derivative(x_hat)
    end select
```

```fortran

    iteration = 0
    !Set a large number
    error = 10000000

    do while(abs(error)>1d-8 .and. iteration < 100)
        last_x = x_hat
        select case(root_id)
            case(1)
                g = g1(x_hat)
            case default
                g = g2(x_hat)
        end select
        x_hat = (1-l)**(-1)*(g-l*x_hat)
        error = abs(last_x - x_hat)
        iteration = iteration + 1
    end do
end subroutine AccCalculate

!Aitken Calculation
subroutine AitkenCalculate(root_id, x_hat,error,iteration)
    implicit none
    integer,intent(in) :: root_id
    real*8,intent(out) :: x_hat,error
    integer,intent(out) :: iteration

    real*8 :: g,g1,g2,g1_derivative,g2_derivative,l
    real*8 :: last_x,last_x_buffer,x_2

    !Initialize the x with a random number in different regions
    !Initialize the factor l
    select case(root_id)
            case (1)
                !Set the x a random number in (0,1)
                call random_number(x_hat)
                l = g1_derivative(x_hat)
            case (2)
                !Set the x a random number in (1,2)
                call random_number(x_hat)
                x_hat = x_hat+1
                l = g2_derivative(x_hat)
            case (3)
                !Set the x a random number in (-2,-1)
                call random_number(x_hat)
```

```
                   x_hat = x_hat*(-1)-1
                   l = g2_derivative(x_hat)
       end select

       select case(root_id)
           case(1)
               last_x = g1(x_hat)
               x_hat = g1(last_x)
           case default
               last_x = g2(x_hat)
               x_hat = g2(last_x)
       end select
       iteration = 0
       !Set a large number
       error = 10000000


       !Main loop
       do while(abs(error)>1d-8 .and. iteration < 100)

           select case(root_id)
           case(1)
               x_2 = g1(x_hat)
           case default
               x_2 = g2(x_hat)
           end select
           last_x_buffer = x_hat
           x_hat = x_2 - (x_2-x_hat)**2/(x_2-2*x_hat+last_x)
           error = abs(last_x - x_hat)
           last_x = last_x_buffer
           iteration = iteration + 1
       end do

   end subroutine AitkenCalculate



   !Functions definitions
   function f(x)
       implicit none
       real*8 :: x
       real*8 :: f
```

```fortran
    f = x**3/3-x
end function

function f_derivative(x)
    implicit none
    real*8 :: x
    real*8 :: f_derivative
    f_derivative = x**2-1
end function
!g1 = x**3/3
function g1(x)
    implicit none
    real*8 :: x
    real*8 :: g1
    g1 = x**3/3
end function
function g1_derivative(x)
    implicit none
    real*8 :: x
    real*8 :: g1_derivative
    g1_derivative = x**2
end function
!g2 = x**(1.0/3)/3
function g2(x)
    implicit none
    real*8 :: x
    real*8 :: g2
    if ( x>0 ) then
        g2 = (3*x)**(1.0/3)
    else
        g2 = sign(abs(3*x)**(1.0/3.0),x)
    end if
end function
!g2 = x**(1.0/3)/3
function g2_derivative(x)
    implicit none
    real*8 :: x
    real*8 :: g2_derivative
    if ( x>0 ) then
        g2_derivative = (3*x)**(dble(-2.0)/dble(3.0))
    else
        g2_derivative = sign(abs(3*x)**(dble(-2.0)/dble(3.0)),x)
    end if
end function
```

- **Results and examples:**

```
#Jacobi Method

Root id       1
Root: -0.0000
Error:       6.748E-20
Iteration:   6

Used time:           55867us

Root id       2
Root:  1.7321
Error:       8.068E-09
Iteration:  17

Used time:          110718us

Root id       3
Root: -1.7321
Error:       3.397E-09
Iteration:  18

Used time:          234540us

#Newton Method

Root id       1
Root:  0.0000
Error:       1.594E-17
Iteration:   6

Used time:           22932us

Root id       2
Root:  1.7321
Error:       5.922E-13
Iteration:   6

Used time:          127540us

Root id       3
Root: -1.7321
Error:       5.281E-10
Iteration:   5

Used time:          176664us
```

```
#Post Acceleration Method

Root id        1
Root:  0.0000
Error:         8.089E-09
Iteration:    4

Used time:             54524us

Root id        2
Root:  1.7321
Error:         9.940E-09
Iteration:   15

Used time:             89251us

Root id        3
Root: -1.7321
Error:         9.006E-09
Iteration:   27

Used time:            155515us

#Aitken Method

Root id        1
Root:  0.0000
Error:         4.734E-11
Iteration:    3

Used time:             36630us

Root id        2
Root:  1.7321
Error:         4.801E-10
Iteration:    9

Used time:            122745us

Root id        3
Root: -1.7321
Error:         1.289E-09
Iteration:    9

Used time:            235672us
```

Iteration Comparison

Method	**X1**	**X2**	**X3**
Jacobi	6	17	18
Newton Downhill	6	6	5
Post Acc	4	15	27
Aitken Acc	3	9	9

Execution Time Comparison (unit = us)

Method	**X1**	**X2**	**X3**
Jacobi	55867us	110718us	234540us
Newton Downhill	22932us	127540us	176664us

Post Acc	54524us	89251us	155515us
Aitken Acc	36630us	122745us	235672us

- **Demo:**
  Check the folder "Root" in the directory.
  Run the program "a.exe"