Computer Science 1510

Lecture 14

Lecture Outline

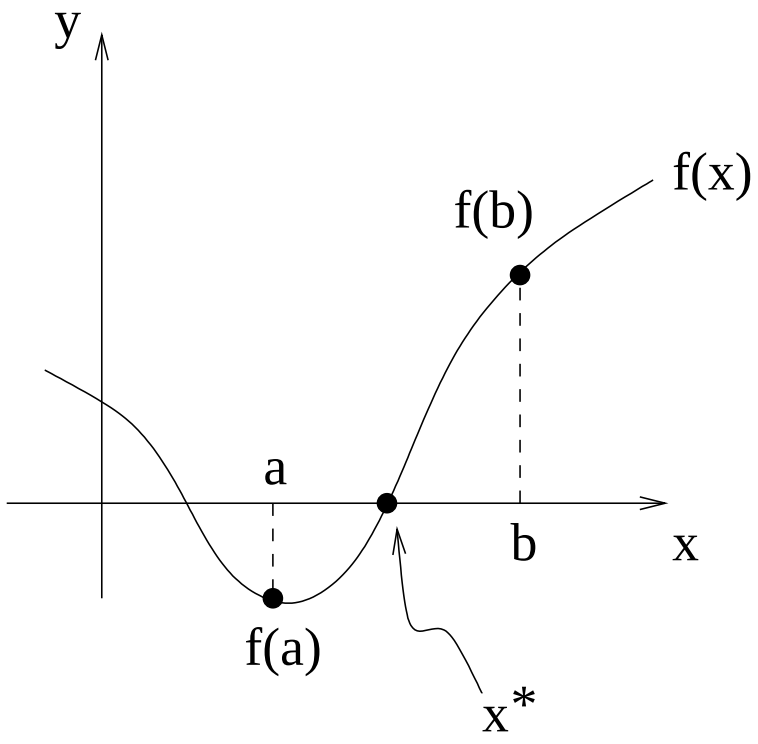
- Scientific Computing Example – Rootfinding

The rootfinding problem

- Problem: Given a function $y = f(x)$, find a value x^* such that $f(x^*) = 0$.

- Such values are called the *roots* of the function.

- Intermediate value theorem:
  If a function $f(x)$ is continuous on $[a, b]$ and $f(a)f(b) < 0$ then $f(x) = 0$ for at least one $x = x^*$ on $[a, b]$.



Rootfinding methods

- Algorithms to solve the rootfinding problem are iterative, that is, they produce a sequence of guesses, that hopefully will converge to the desired root.

- One rootfinding method is called the *Bisection algorithm*.

- In the bisection algorithm it is assumed that we know an interval $[a, b]$ where $f(a)$ and $f(b)$ have different signs (ie. $f(a)f(b) < 0$).

- In this case we know, by the intermediate value theorem, that there is at least one root in the interval $[a, b]$.

Bisection algorithm

- To find a root of $f(x)$ we first find an interval $[a, b]$ where $f(a)f(b) < 0$.

- We proceed to find the root via the following steps:

  1. Compute the midpoint of the interval $[a, b]$,

$$m = \frac{a+b}{2}.$$

  2. If $f(m) = 0$ then stop, we have reached a root.
  3. Otherwise, one of the following is true:
     - $f(a)f(m) < 0$, or
     - $f(b)f(m) < 0$.

     In the first case, there is a root in the interval $[a, m]$ so we set $b = m$ and go to step 1.
     In the second case, there is a root in the interval $[m, b]$ so we set $a = m$ and go to step 1.

Bisection algorithm

- On each iteration we are cutting the interval in half, hence the name *bisection*.

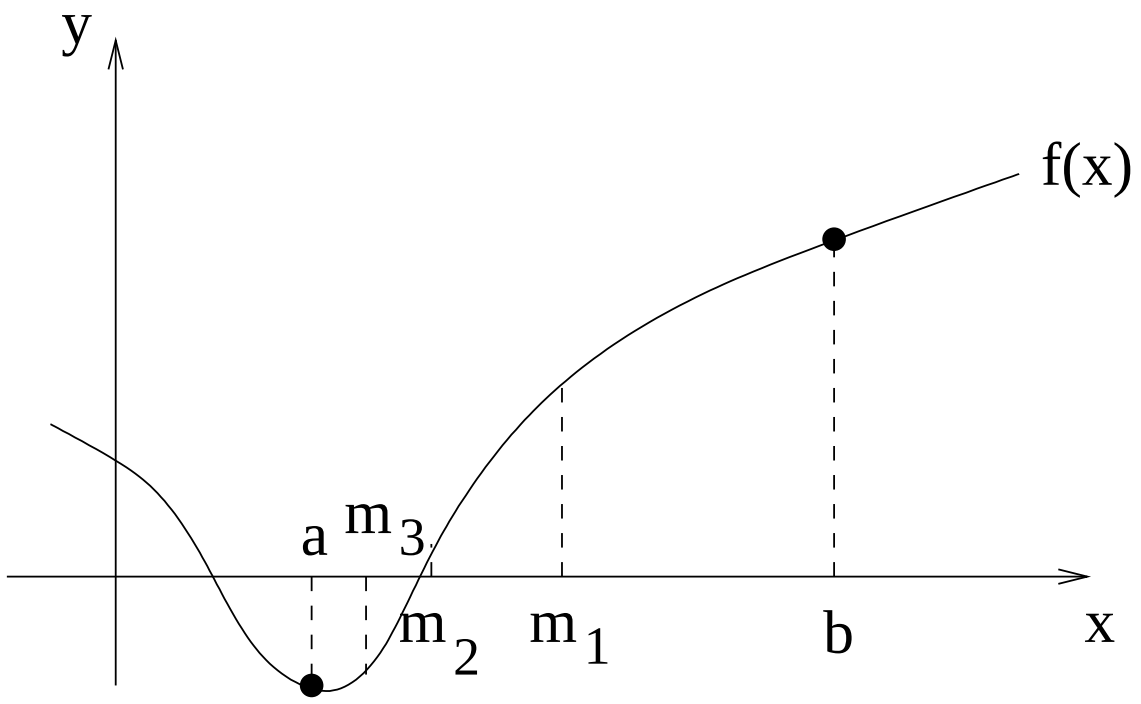
- We have seen that comparing two floating point numbers `X` and `Y` for equality is best done by checking if `ABS(X-Y) < tol`.

- The stopping criteria used for this algorithm requires a similar consideration.

- Due to round-off errors in calculations, it is unlikely that the value of the function will ever be exactly zero. Thus, rather than testing $f(m) = 0$ it would be wise to instead introduce some *tolerance* (`tol`) and test that `ABS`$(f(m))$ `< tol`.

- For example, a tolerance of 1.0×10^{-5} would ensure that $f(m)$ is very close to zero before concluding that we have reached a root.

Illustration of the bisection algorithm



Example: Bisection Method

```
PROGRAM Bisection
!-----------------------------------------------------------------------
! The following program uses the bisection method to approximate
! the root of an equation f(x) (ie. a value of x where f(x)=0).
!  INPUT:
!   left,right - endpoints of an interval where f(left)*f(right)<0
!   nmax - maximum number of iterations
!   tol - tolerance to be used to determine convergence to the root
!  OUTPUT:
!   mid - the value of the root at convergence (if achieved)
!   n - the number of iterations required for convergence (if achieved)
!-----------------------------------------------------------------------
   IMPLICIT NONE
   INTEGER::n,nmax
   REAL::left,right,mid  ! Left, right, and mid points
   REAL::fl,fr,fmid  ! Function values at left, right, and midpoint
   REAL::tol

   INTERFACE
      FUNCTION fval(x)
         REAL::fval
         REAL,INTENT(IN)::x
      END FUNCTION fval
   END INTERFACE

   WRITE(*,*) 'Please enter the left endpoint:'
   READ(*,*) left
   WRITE(*,*) 'Please enter the right endpoint:'
   READ(*,*) right
   WRITE(*,*) 'Enter the maximum number of iterations:'
   READ(*,*) nmax
   WRITE(*,*) 'Enter a value for the tolerance'
   READ(*,*) tol

   fl=fval(left)
   fr=fval(right)
```

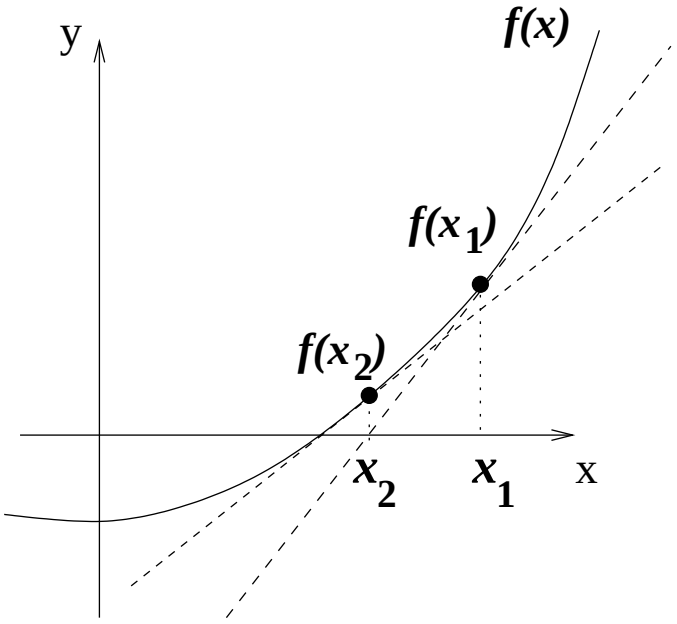
```
   IF (fl*fr>=0) THEN
      WRITE(*,*) 'The interval does not bracket a root'
      STOP
   ENDIF

   n=1
   mid=(left+right)/2.0
   fmid=fval(mid)

   DO WHILE (ABS(fmid)>tol)
      IF (n>nmax) THEN
         WRITE(*,*) 'Maximum number of iterations exceeded'
         STOP
      ENDIF
      IF (fl*fmid<0) THEN  ! Root is between left and mid
         right=mid
      ELSE  ! Root is between mid and right
         left=mid
      ENDIF
      mid=(left+right)/2.0
      fl=fval(left)
      fr=fval(right)
      fmid=fval(mid)
      n=n+1
   END DO
   WRITE(*,*) 'The root is approximately x = ',mid
   WRITE(*,*) 'The value of the function there is f(x) = ',fmid
   WRITE(*,*) 'There were ',n,' iterations required'
END PROGRAM Bisection

FUNCTION fval(x)
!-----------------------------------------------------------------------
! The following function computes the value of f(x) at a given x value.
!-----------------------------------------------------------------------
   REAL::fval
   REAL,INTENT(IN)::x
   fval = x**2-2
END FUNCTION fval
```

Newton's Method



- Starting with an initial approximation x_1, compute the tangent line to the graph at $(x_1, f(x_1))$.

- The point x_2 where the tangent line crosses the x-axis is taken as the second approximation.

- Similarly, compute the tangent line to the graph at $(x_2, f(x_2))$, and take the x-intercept of the tangent line as the third approximation.

- Continue this process until we are sufficiently close to the root.

Newton's Method

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

- Newton's method requires only one initial guess, instead of an interval like that required for the bisection method.

- The guess must be "sufficiently" close to the root.

- The following program implements Newton's method for the function $f(x) = x^3 + x - 5$, terminating when `ABS(f(x)) < tol` or some maximum number of iterations has been exceeded.

Example: Newton's Method

```
PROGRAM Newtons_Method
!-----------------------------------------------------------------------
! Program to find an approximate root of a function f(x) using Newton's
! method.  Variables used are:
!   fval,fpval    : the function and its derivative (internal functions)
!   oldapprox     : previous approximation (initially the first one)
!   fp_old        : value of the derivative of f at oldapprox
!   newapprox     : the new approximation
!   fapprox       : value of f at an approximation
!   ftol          : repetition stops when ABS(fapprox) < ftol
!   fptol         : repetition stops when ABS(fp_old) < fptol
!   maxnum        : limit on number of iterations
!   n             : number of iterations
!
! Input:  ftol,fptol,maxnum,oldapprox
! Output: Iteration number n, the nth approximation, and the value of
!         f at that approximation, or an error message indicating
!         that the method fails
!-----------------------------------------------------------------------
   IMPLICIT NONE
   INTEGER::maxnum,n
   REAL::oldapprox,fp_old,newapprox,ftol,fptol,fapprox

   ! Get termination values (ftol and fptol), maximum number of
   ! iterations, and initial approximation
   WRITE(*,*) 'Enter tolerance for f(x) and derivative of f(x)'
   READ(*,*) ftol, fptol
   WRITE(*,*) 'Enter max # of iterations,and the initial approximation:'
   READ(*,*) maxnum,oldapprox

   ! Initialize function value and iteration counter
   fapprox=fval(oldapprox)
   n=0
   WRITE(*,*) '  N      X(N)       F(X(N))'
   WRITE(*,*) '============================'
   WRITE(*,10) 0, oldapprox, fapprox
```

```
   10 FORMAT(1X, I3, F11.5, E14.5)

   ! Iterate using Newton's method while ABS(fapprox) is greater
   ! than or equal to tol and n has not reached maxnum
   DO
      IF ((ABS(fapprox) < ftol) .OR. (n > maxnum)) EXIT
      ! If a termination condition met, stop generating approximations
      ! Otherwise continue with the following
      n = n + 1
      fp_old = fpval(oldapprox)

      ! Terminate if the derivative is 0 at some approximation
      IF (ABS(fp_old) < fptol) THEN
         WRITE(*,*) 'Newton''s method fails -- derivative = 0'
         EXIT
      END IF

      ! Generate a new approximation
      newapprox = oldapprox - (fapprox/fp_old)
      fapprox = fval(newapprox)
      WRITE(*,10) n,newapprox,fapprox
      oldapprox = newapprox
   END DO

CONTAINS

   !-fval(x)-----------------------------------
   ! Function for which a root is being found
   !-------------------------------------------
   FUNCTION fval(x)
     REAL::fval
     REAL,INTENT(IN)::x
     fval = x**3 + x - 5.0
   END FUNCTION fval

   !-fpval(x)----------------------------------
   ! The derivative of the function f
   !-------------------------------------------
```

```
   FUNCTION fpval(x)
     REAL::fpval
     REAL,INTENT(IN)::x
     fpval = 3.0*x**2 + 1.0
   END FUNCTION fpval
END PROGRAM Newtons_Method
```