CH2120

Class 15

# mainProgram.f08

**program** mainProgram

**implicit** **none**

**call** mainBracketingMethods()

**end** **program** mainProgram

# mainBracketingMethods.f08

**subroutine** mainBracketingMethods

**implicit** **none**

**real**, **external** :: rootFindingFunction

**logical**, **external** :: haveOppositeSigns

**real** xLow, xHigh

**real** tolerance

**real** root, error

**integer** iterations

**character**(*len*=25) methodName

**real** fLow, fHigh

**write**(\*,\*) "Enter the lower limit of the initial bracket:"

**read**(\*,\*) xLow

**write**(\*,\*) "Enter the lower limit of the initial bracket:"

**read**(\*,\*) xHigh

**write**(\*,\*) "Enter the tolerance:"

**read**(\*,\*) tolerance

fLow = rootFindingFunction(xLow)

fHigh = rootFindingFunction(xHigh)

**write**(\*,10) "|", "Method", "|", "Root", "|", "Error", "|", "Iterations", "|"

**if**(fLow == 0) **then**

root = xLow

error = 0.0

iterations = 0

methodName = "None"

**call** displayRoot(methodName, root, error, iterations)

**return**

**end** **if**

**if**(fHigh == 0) **then**

root = xHigh

error = 0.0

iterations = 0

methodName = "None"

**call** displayRoot(methodName, root, error, iterations)

**return**

**end** **if**

**if**(haveOppositeSigns(fLow, fHigh) .**eqv**. .**false**.) **then**

**stop** "Error: Limits don't bracket the root."

**end** **if**

methodName = "Bisection"

**call** bisectionRoot(xLow, xHigh, tolerance, root, error, iterations)

**call** displayRoot(methodName, root, error, iterations)

methodName = "False-Position"

**call** falsePositionRoot(xLow, xHigh, tolerance, root, error, iterations)

**call** displayRoot(methodName, root, error, iterations)

methodName = "Modified False-Position"

**call** modifiedFalsePositionRoot(xLow, xHigh, tolerance, root, error, iterations)

**call** displayRoot(methodName, root, error, iterations)

10 **format**(a3, a25, a3, a10, a3, a10, a3, a12, a3)

**end** **subroutine** mainBracketingMethods

# bisectionRoot.f08

**subroutine** bisectionRoot(xLowInput, xHighInput, tolerance, root, error, iterations)

**implicit** **none**

**real**, **external** :: rootFindingFunction

**logical**, **external** :: haveOppositeSigns

**real**, **intent**(in) :: xLowInput, xHighInput, tolerance

**real**, **intent**(out) :: root, error

**integer**, **intent**(out) :: iterations

**real** xLow, xMid, xHigh

**real** fLow, fMid, fHigh

xLow = xLowInput

xHigh = xHighInput

xMid = (xLow + xHigh) / 2

fLow = rootFindingFunction(xLow)

fHigh = rootFindingFunction(xHigh)

root = xMid

error = *abs*((xHigh - xLow) / 2)

iterations = 0

**do** **while**(error > tolerance)

iterations = iterations + 1

xMid = (xLow + xHigh) / 2

fMid = rootFindingFunction(xMid)

error = *abs*((xHigh - xLow) / 2)

**if**(haveOppositeSigns(fLow, fMid)) **then**

xHigh = xMid

fHigh = fMid

**else** **if**(haveOppositeSigns(fHigh, fMid)) **then**

xLow = xMid

fLow = fMid

**else**

**exit**

**end** **if**

root = xMid

**end** **do**

**end** **subroutine** bisectionRoot

# falsePositionRoot.f08

**subroutine** falsePositionRoot(xLowInput, xHighInput, tolerance, root, error, iterations)

**implicit** **none**

**real**, **external** :: rootFindingFunction

**logical**, **external** :: haveOppositeSigns

**real**, **intent**(in) :: xLowInput, xHighInput, tolerance

**real**, **intent**(out) :: root, error

**integer**, **intent**(out) :: iterations

**real** xLow, xIntercept, xHigh

**real** fLow, fIntercept, fHigh

**real** xInterceptPrevious

**real** slope

xLow = xLowInput

xHigh = xHighInput

fLow = rootFindingFunction(xLow)

fHigh = rootFindingFunction(xHigh)

slope = (fHigh - fLow) / (xHigh - xLow)

xIntercept = xHigh - (fHigh / slope)

iterations = 0

**do** **while**((error > tolerance) .**or**. (iterations <= 2))

iterations = iterations + 1

slope = (fHigh - fLow) / (xHigh - xLow)

xIntercept = xHigh - (fHigh / slope)

fIntercept = rootFindingFunction(xIntercept)

**if**(haveOppositeSigns(fLow, fIntercept)) **then**

xHigh = xIntercept

fHigh = fIntercept

**else** **if**(haveOppositeSigns(fHigh, fIntercept)) **then**

xLow = xIntercept

fLow = fIntercept

**else**

**exit**

**end** **if**

error = *abs*(xIntercept - xInterceptPrevious)

xInterceptPrevious = xIntercept

**end** **do**

root = xIntercept

**end** **subroutine** falsePositionRoot

# modifiedFalsePositionRoot.f08

**subroutine** modifiedFalsePositionRoot(xLowInput, xHighInput, tolerance, root, error, iterations)

**implicit** **none**

**real**, **external** :: rootFindingFunction

**logical**, **external** :: haveOppositeSigns

**real**, **intent**(in) :: xLowInput, xHighInput, tolerance

**real**, **intent**(out) :: root, error

**integer**, **intent**(out) :: iterations

**real** xLow, xIntercept, xHigh

**real** fLow, fIntercept, fHigh

**real** xInterceptPrevious

**real** slope

**real** xLowMoves, xHighMoves

xLow = xLowInput

xHigh = xHighInput

fLow = rootFindingFunction(xLow)

fHigh = rootFindingFunction(xHigh)

slope = (fHigh - fLow) / (xHigh - xLow)

xIntercept = xHigh - (fHigh / slope)

xLowMoves = 0

xHighMoves = 0

iterations = 0

**do** **while**((error > tolerance) .**or**. (iterations <= 2))

iterations = iterations + 1

slope = (fHigh - fLow) / (xHigh - xLow)

xIntercept = xHigh - (fHigh / slope)

fIntercept = rootFindingFunction(xIntercept)

**if**(haveOppositeSigns(fLow, fIntercept)) **then**

xHigh = xIntercept

fHigh = fIntercept

xHighMoves = xHighMoves + 1

xLowMoves = 0

**else** **if**(haveOppositeSigns(fHigh, fIntercept)) **then**

xLow = xIntercept

fLow = fIntercept

xLowMoves = xLowMoves + 1

xHighMoves = 0

**else**

**exit**

**end** **if**

error = *abs*(xIntercept - xInterceptPrevious)

xInterceptPrevious = xIntercept

**if**(xLowMoves == 2) **then**

fHigh = fHigh / 2

xLowMoves = 0

**end** **if**

**if**(xHighMoves == 2) **then**

fLow = fLow / 2

xHighMoves = 0

**end** **if**

**end** **do**

root = xIntercept

**end** **subroutine** modifiedFalsePositionRoot

# rootFindingFunction.f08

**real** **function** rootFindingFunction(x)

**implicit** **none**

**real**, **intent**(in) :: x

rootFindingFunction = x\*\*11 - 1

! rootFindingFunction = (3 \* x) + sin(x) - exp(x)

**end** **function** rootFindingFunction

# haveOppositeSigns.f08

**logical** **function** haveOppositeSigns(number1, number2)

**implicit** **none**

**real**, **intent**(in) :: number1, number2

haveOppositeSigns = .**false**.

**if**(number1 \* number2 < 0) **then**

haveOppositeSigns = .**true**.

**end** **if**

**end** **function** haveOppositeSigns

# displayRoot.f08

**subroutine** displayRoot(methodName, root, error, iterations)

**implicit** **none**

**character**(*len*=25), **intent**(in) :: methodName

**real**, **intent**(in) :: root, error

**integer**, **intent**(in) :: iterations

**write**(\*,10) "|", methodName, "|", root, "|", error, "|", iterations, "|"

10 **format**(a3, a25, a3, f10.4, a3, f10.4, a3, i12, a3)

**end** **subroutine** displayRoot

# Output

## Output 1 [(3 \* x) + *sin*(x) - *exp*(x)]

Enter the lower limit of the initial bracket:

0

Enter the lower limit of the initial bracket:

1

Enter the tolerance:

1e-6

| Method | Root | Error | Iterations |

|Bisection | 0.3604 | 0.0000 | 20 |

|False-Position | 0.3604 | 0.0000 | 8 |

|Modified False-Position | 0.3604 | 0.0000 | 6 |

## Output 2: f(x) is highly nonlinear [x\*\*11 - 1]

Enter the lower limit of the initial bracket:

0

Enter the lower limit of the initial bracket:

1.5

Enter the tolerance:

1e-6

| Method | Root | Error | Iterations |

|Bisection | 1.0000 | 0.0000 | 21 |

|False-Position | 1.0000 | 0.0000 | 221 |

|Modified False-Position | 1.0000 | 0.0007 | 17 |

Note on the output of the Modified False-Position method: For this set of inputs (0, 1.5, 10-6), the x-intercept after the 17th iteration becomes equal to 1.0 (the exact root). Consequently, the third condition in the [if, else-if, else, end if] block of statements is evaluated as true, and the compiler exits the do-while loop. The error displayed on the console is the “speed” with which the false-position method reached the root in the last iteration. This value is greater than the tolerance; iterations stop nonetheless because fIntercept doesn’t have opposite signs compared to the function values at either limit.

To verify this, try adding the following statements in the else condition (before exiting the loop)—

**else**

**write**(\*,\*) "xIntercept = ", xIntercept

**write**(\*,\*) "fIntercept = ", fIntercept

**write**(\*,\*) "xLow = ", xLow

**write**(\*,\*) "fLow = ", fLow

**write**(\*,\*) "xHigh = ", xHigh

**write**(\*,\*) "fHigh = ", fHigh

**exit**

**end** **if**