CH2120

Class 21

# mainProgram.f08

**program** mainProgram

**implicit** **none**

**call** mainBirgeVieta()

**end** **program** mainProgram

# mainBirgeVieta.f08

**subroutine** mainBirgeVieta

! Goal: Obtain n roots of the polynomial equation Pn(x) = 0

**implicit** **none**

! Variables

!! n: degree of Pn(x)

!! a: array to store (n+1) coefficients of P

!! b: array to store (n) coefficients of Q [Pn(x) = (x - xi)Qn-1(x) + R]

!! roots: array to store roots of Pn(x) = 0

**integer** n

**real**, **dimension**(:), **allocatable** :: a, b

**real**, **dimension**(:), **allocatable** :: roots

**real** xGuess, error, tolerance

**integer** iterations, counter

! Illustration

!! P(x) = x^5 - 7x^4 - 3x^3 + 79x^2 - 46x - 120 = 0

!! Roots: -3, -1, 2, 4, 5

! Initialize n, a

n = 5

**allocate**(a(n+1))

**allocate**(b(n+1))

**allocate**(roots(n))

a(1) = 1

a(2) = -7

a(3) = -3

a(4) = 79

a(5) = -46

a(6) = -120

! Initialize the roots array

roots = 0

! Initialize xGuess, tolerance

xGuess = 1000.00

tolerance = 1.0e-6

! Find roots of Pn(x) = 0 one by one [from root #1 to root # (n-1)].

**do** counter = 1, (n - 1)

**call** birgeVietaOneRoot(a, (n + 1 - counter), xGuess, tolerance, b, roots(counter), error, iterations)

! Explanation for the (n + 1 - counter) term--

!! counter=1, n+1-counter = n, obtain the first root for the original Pn

!! counter=2, n+1-counter = n-1, obtain the second root for the reduced polynomial of degree (n-1)

!! counter=3, n+1-counter = n-2, obtain the third root for the reduced polynomial of degree (n-2)

!! ...and so on

!! During each subsequent execution of the loop, the degree reduces by 1.

!! counter=(n-1), n+1-counter = 2, obtain the (n-1)th root.

!! P has reduced to a quadratic term.

!! After extracting (x-xi), Q reduces to a linear term; Q = b1x + b2.

! Display the current root

**write**(\*,10) "Root #", counter, " = ", roots(counter)

!! Qn-1 obtained from one call to birgeVietaOneRoot becomes the Pn for the next call.

a = b

**end** **do**

! The last (nth) root is obtained directly since Q is now linear.

! b1x + b2 = 0

roots(n) = -b(2) / b(1)

**write**(\*,10) "Root #", n, " = ", roots(n)

**deallocate**(a)

**deallocate**(b)

10 **format**(a6, i1, a3, f5.2)

**end** **subroutine** mainBirgeVieta

# birgeVietaOneRoot.f08

**subroutine** birgeVietaOneRoot(a, n, xGuess, tolerance, b, root, error, iterations)

! Goal: Return one root of Pn(x) = 0 using the Birge-Vieta method

**implicit** **none**

! External function

**real**, **external** :: newtonRaphsonOneStep

! Variables: Input Arguments

!! a: array that stores all (n+1) coefficients of Pn(x)

!! n: degree of polynomial Pn(x)

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

**real**, **dimension**(n+1), **intent**(in) :: a

**real**, **intent**(in) :: xGuess, tolerance

! Variables: Output Arguments

!! b: array that stores all (n) coefficients of Qn-1(x)

!! and the remainder R in the last element of b

!! Why is b an output argument?

**real**, **dimension**(n+1), **intent**(out) :: b

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

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

! Variables: Local

!! c: array that stores (n) values of d(bk)/d(xi)

**real**, **dimension**(n) :: c

**real** x, xPrevious

**integer** counter

! Initialize Output Arguments

b = 0

iterations = 0

! Initialize Local Variables

c = 0

x = xGuess

! Newton-Raphson Iterations

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

iterations = iterations + 1

! Use the recursive expression to find all b's from a's and previous b's

! b\_k = a\_k + (xi)(b\_k-1)

! b(1) = b\_0 from PowerPoint

! b(n+1) = bn = R from PowerPoint.

!! This is the numerator of the Newton-Raphson expression.

b(1) = 1

**do** counter = 2, (n + 1)

b(counter) = a(counter) + (x \* b(counter - 1))

**end** **do**

! Use the recursive expression to find all c's from b's and previous c's

! c\_k = b\_k + (xi)(c\_k-1)

! c(1) = c\_0 = d(b\_1)/d(xi) from PowerPoint

! c(n) = c\_n-1 = d(b\_n) / d(xi) = dR/d(xi) from PowerPoint

c(1) = 1

**do** counter = 2, (n)

c(counter) = b(counter) + (x \* c(counter - 1))

**end** **do**

! Update x using the Newton-Raphson expression

!! Numerator = R(xi) = b(n+1)

!! Denominator = R'(xi) = c(n)

x = newtonRaphsonOneStep(x, b(n+1), c(n))

error = *abs*(x - xPrevious)

xPrevious = x

**end** **do**

root = x

**end** **subroutine** birgeVietaOneRoot

# newtonRaphsonOneStep.f08

**real** **function** newtonRaphsonOneStep(xGuess, fOfx, dfBydx)

! Perform one iteration of the Newton-Raphson method and return the new x

**implicit** **none**

**real**, **intent**(in) :: xGuess, fOfx, dfBydx

newtonRaphsonOneStep = xGuess - (fOfx / dfBydx)

**end** **function** newtonRaphsonOneStep

# Output

Root #1 = 5.00

Root #2 = 4.00

Root #3 = 2.00

Root #4 = -1.00

Root #5 = -3.00