1. The equation:

can be solved for by using Newton’s method with

and

To evaluate at the approximation , we need a quadrature formula to approximate

1. Find a solution to accurate to within using Newton’s method with and the Composite Simpson’s rule.

! Mikayla Webber

! 4670 Numerical Analysis

! Homework Four Due 11/6/17

module secret ! secret module

integer :: fcounter ! counter for loop

end module secret

!-------------------------------------------------------------------------------------------------------------------------

program WebberHomework4Question1 ! main program

use secret ! uses secret module

implicit none

real :: a, b, f, h ! declares real variables

real :: actual ! declares real variable

real :: simpson, s ! declares real variable

integer :: i, n ! declares integer variables

a = 1.0 ! sets a to 1

b = 2.0 ! sets b to 2

n = 2 ! sets n to 2

actual = log(2.0) ! sets actual to the log of 2

print\*, "Newton and Composite Simpson Method" ! displays message to output

do i = 1, 25

h = (b - a) / float(n) ! sets h to (b - a) / n

s = (simpson(n, a, b, f)) ! sets s to simpson function

!print\*, n, s, abs(actual - s), abs(actual - s) / h\*\*4.0

write(\*,\* ) n, s, abs(actual - s), abs(actual - s) / h\*\*4.0

! writes message to standard output

n = (n \* 2) ! sets n to n \* 2

fcounter = (fcounter + 1) ! increments counter

end do

stop

end program WebberHomework4Question1 ! ends main program

!-------------------------------------------------------------------------------------------------------------------------

real function fprime(x) ! function for fprime(x)

implicit none

real :: f, pi, x ! declares real variables

f = exp(1.0)

pi = (4 \* ATAN(1.0)) ! most accurate for pi

fprime = (1.0 / SQRT(2.0 \* pi)) \* (f\*\*(-x\*\*2.0) / 2.0)

! sets fprime to the equation from problem

return

end

!-------------------------------------------------------------------------------------------------------------------------

real function simpson(n, a, b, fprime) ! function for composite simpson

implicit none

integer :: i, n ! declares integer variables

real :: a, b, fv, h, fprime ! declares real variables

real :: se, so ! declares simpson even and odd real variables

h = (b - a) / float(n) ! sets h to (b - a) / n

simpson = 0.0 ! initial value of zero

se = 0.0 ! initial value of zero

so = 0.0 ! initial value of zero

do i = 1, n - 1 ! loop that decrements

fv = fprime(a + float(i) \* h) ! calls fprime function for fv value

if (mod(i, 2) == 0) then ! if even then set se to

se = se + fv ! simpson even + fv

else

so = so + fv ! else set so to simpson odd + fv

end if

end do

simpson = (h / 3.0 \* (fprime(a) + fprime(b) + 2.0 \* se + 4.0 \* so))

! sets simpson to composite simpson equation

return

end

1. Repeat (a) using the Composite Trapezoidal rule in place of the Composite Simpson’s rule.

! Mikayla Webber

! 4670 Numerical Analysis

! Homework Four Due 11/6/17

module secret ! secret module

integer :: fcounter ! counter for loop

end module secret

!-------------------------------------------------------------------------------------------------------------------------

program WebberHomework4Question1 ! main program

use secret ! uses secret module

implicit none

real :: a, b, f, h ! declares real variables

real :: actual, t, trapezoid ! declares real variable

integer :: i, n ! declares integer variables

a = 1.0 ! sets a to 1

b = 2.0 ! sets b to 2

n = 2 ! sets n to 2

actual = log(2.0) ! sets actual to the log of 2

print\*, "Newton and Composite Trapezoid Method" ! displays message to output

do i = 1, 25

h = (b - a) / float(n) ! sets h to (b - a) / n

! print\*, h ! for debugging

t = (trapezoid(n, a, b, f)) ! sets t to trapezoid function

! print\*, t ! for debugging

print\*, n, t, abs(actual - t), abs(actual - t) / h\*\*4.0 ! writes message to standard output

n = (n \* 2) ! sets n to n \* 2

! print\*, n ! for debugging

fcounter = (fcounter + 1) ! increments counter

! print\*, fcounter ! for debugging

end do

stop

end program WebberHomework4Question1 ! ends main program

!-------------------------------------------------------------------------------------------------------------------------

real function fprime(x) ! function for fprime(x)

implicit none

real :: f, pi, x ! declares real variables

f = exp(1.0)

pi = (4 \* ATAN(1.0)) ! most accurate for pi

fprime = (1.0 / SQRT(2.0 \* pi)) \* (f\*\*(-x\*\*2.0) / 2.0)

! sets fprime to the equation from problem

return

end ! ends function

!-------------------------------------------------------------------------------------------------------------------------

real function trapezoid(n, a, b, fprime) ! function for trapezoid

implicit none

integer :: i, n ! declares integer variables

real :: a, b, fprime, h, xuse ! declares real variables

h = (b - a) / float(n) ! sets h to (b - a) / n

trapezoid = 0.0 ! initial value of zero

do i = 1, n - 1

xuse = a + float(i) \* h ! sets xuse to a + i \* h

trapezoid = fprime + fprime(xuse) ! sets trapezoid to fprime + fprime(xuse)

end do

trapezoid = h / 2.0 \* (2 \* trapezoid + fprime(a) + fprime(b))

! sets trapezoid to equation in problem

return

end ! ends function

1. In all our approximation methods, we make the tacit assumption that the function we are integrating is fairly nice in the sense of having continuous derivatives, etc. What if this is not so? Use the composite trapezoid rule, composite Simpson’s rule and Romberg quadrature to estimate:

where if and if . This function is continuous but not differentiable at . Try to get as much accuracy as you can, be sure to mention the number of subdivisions and how much accuracy was obtained. Report any interesting behaviors.

1. Fully describe the development of your two-dimensional Simpson’s rule approximation and implement it in code. Make up some simple example as we did in class and compute the exact value of the double integral. Use this as a sanity check that your code is working correctly.
2. Let *R* be the region described by and . A mass density function is given by:

The center of mass is calculated in the following way:

Estimate the center of mass, using your two-dimensional Simpson’s rule code to do the three double integrals needed. First use 12 subdivisions in the variable and 14 in the variable. Then return the problem using one hundred subdivisions for each variable.