Math 4610 Tasksheet 2

Jacob Fitzgerald (A02261889)

Docs

https://jfitzusu.github.io/math4610/

Task 1

Code for Newton's Method:

```
'''
Approximates the Root of a Function Using the Newton Method
f: The Function in Question. MUST BE A SYMPY EXPRESSION
x0: Initial Guess
tol: Maximum Permissible Error
maxIter: Maximum Iterations to Test Before Giving Up
-v: Verbose Mode, Tracks/Returns Intermittent Results
Returns: The Approximate Root, Convergence Status, Intermittent Results if
Toggled

NOTES: f'(x0) Cannot Equal 0
NOTES: FUNCTION MUST BE A SYMPY EXPRESSION
'''
def newton(f, x0, tol, maxIter=1000, v=False):
    x1 = 0
    x = sympy.symbols('x')
    derF = sympy.diff(f, x)
    error = 10 * tol

    if v:
        resultsTable = []

    for i in range(maxIter):
        derivative = derF.subs(x, x0)

        if derivative == 0:
            raise Exception("ERROR: Encountered Invalid Derivative {0}")

        x1 = x0 - f.subs(x, x0) / derivative
        error = abs(x1 - x0)
        if v:
            resultsTable.append([i, x1, error])

        if error <= tol:
            break

        x0 = x1

    if v:
        return x1, error <= tol, resultsTable
```

```
    return x1, error <= tol
```

Code for Testing:

```
def testNewton():
    print("TESTING NEWTON METHOD")
    print("---------------------")
    x = sympy.symbols('x')
    expr = x * math.e ** -x

    for i in range(-3, 4):
        sol = newton(expr, i + 0.2, 0.001)
        print(f"The Approximate Root of x * e ^ -x Using {i + 0.2} as an Initial
Guess is {sol[0]}")
        print(f"CONVERGED: {sol[1]}.")
    print()
```

Testing Output:

```
TESTING NEWTON METHOD
---------------------
The Approximate Root of x * e ^ -x Using -2.8 as an Initial Guess is
-4.84985051763171E-9
CONVERGED: True.
The Approximate Root of x * e ^ -x Using -1.8 as an Initial Guess is
-1.57405083662271E-11
CONVERGED: True.
The Approximate Root of x * e ^ -x Using -0.8 as an Initial Guess is
-3.94254168339394E-9
CONVERGED: True.
The Approximate Root of x * e ^ -x Using 0.2 as an Initial Guess is
-3.19841468958201E-11
CONVERGED: True.
The Approximate Root of x * e ^ -x Using 1.2 as an Initial Guess is
1011.22139775427
CONVERGED: False.
The Approximate Root of x * e ^ -x Using 2.2 as an Initial Guess is
1008.70733923114
CONVERGED: False.
The Approximate Root of x * e ^ -x Using 3.2 as an Initial Guess is
1009.16089902031
CONVERGED: False.
```

Task 2

Code for Secant Method:

```
'''
Approximates the Root of a Function Using the Secant Method
f: The Function in Question, a Lambda
x0: Initial Guess
x1: Second Initial Guess
```

```
tol: Maximum Permissible Error
maxIter: Maximum Iterations to Test Before Giving Up
-v: Verbose Mode, Tracks/Returns Intermittent Results
Returns: The Approximate Root, Convergence Status, Intermittent Results if
Toggled
'''
def secant(f, x0, x1, tol, maxIter=1000, v=False):
    f0 = f(x0)
    f1 = f(x1)
    x2 = 0
    error = tol * 10
    resultsTable = []

    for i in range(maxIter):
        x2 = x1 - f1 * (x1 - x0) / (f1 - f0)
        error = abs(x2 - x1)

        if v:
            resultsTable.append([i, x2, error])

        if error <= tol:
            break

        x0 = x1
        x1 = x2
        f0 = f1
        f1 = f(x1)

    if v:
        return x2, error <= tol, resultsTable

    return x2, error <= tol
```

Code for Testing:

```
def testSecant():
    print("TESTING SECANT METHOD")
    print("---------------------")
    f = lambda x: x * math.e ** -x

    for i in range(-3, 4):
        sol = secant(f, i - 0.2, i, 0.001)
        print(f"The Approximate Root of x * e ^ - x Using {i - 0.2} and {i} as
Initial Values is {sol[0]}")
        print(f"CONVERGED: {sol[1]}")
    print()
```

Testing Output:

```
TESTING SECANT METHOD
---------------------
The Approximate Root of x * e ^ - x Using -3.2 and -3 as Initial Values is
-1.2606547390883835e-06
CONVERGED: True
```

```
The Approximate Root of x * e ^ - x Using -2.2 and -2 as Initial Values is
-1.9046048655496834e-06
CONVERGED: True
The Approximate Root of x * e ^ - x Using -1.2 and -1 as Initial Values is
-4.413357882363867e-07
CONVERGED: True
The Approximate Root of x * e ^ - x Using -0.2 and 0 as Initial Values is 0.0
CONVERGED: True
The Approximate Root of x * e ^ - x Using 0.8 and 1 as Initial Values is
0.9996393159369001
CONVERGED: True
The Approximate Root of x * e ^ - x Using 1.8 and 2 as Initial Values is
701.6745834655374
CONVERGED: False
The Approximate Root of x * e ^ - x Using 2.8 and 3 as Initial Values is
702.0129256425017
CONVERGED: False
```

Task 3

Visible in the Code for Task 1 and Task 2.

Task 4

Code for Hybrid Newton Method:

```
'''
Approximates the Root of a Function Using the Newton Method, Falling Back to
Bisection if It Doesn't Converge
f: The Function in Question. MUST BE A SYMPY EXPRESSION
a: Lower Boundary
b: Upper Boundary
tol: Maximum Permissible Error
maxIter: Maximum Iterations to Test Before Giving Up
maxTries: Maximum Times to Reduce the Problem With Bisection Before Giving Up
strictInterval: Boolean, Dictates Whether Convergence Must Happen Within the
Interval

Returns: The Approximate Root, Convergence Status

NOTES: f'(b) Cannot Equal 0
NOTES: FUNCTION MUST BE A SYMPY EXPRESSION
'''


def hybridNewton(f, a, b, tol, maxiter=1000, maxTries=10, strictInterval=False):
    error = 10 * tol
    x = sympy.symbols('x')
    derF = sympy.diff(f, x)

    for i in range(maxTries):

        # Setup for Newton Method
        x0 = b
        x1 = 0
```

```
        for j in range(maxiter):
            derivative = derF.subs(x, x0)

            if derivative == 0:
                raise Exception("ERROR: Encountered Invalid Derivative {0}")

            x1 = x0 - f.subs(x, x0) / derivative

            error = abs(x1 - x0)

            if error <= tol:
                break

            x0 = x1

        # Returns Root if Convergence Happened
        if strictInterval:
            if error < tol and a < x1 < b:
                return x1, True
        elif error < tol:
            return x1, True

        # Reduces Interval With Bisection if Convergence Failed
        fa = f.subs(x, a)
        fb = f.subs(x, b)

        for j in range(4):
            c = 0.5 * (a + b)
            fc = f.subs(x, c)

            if fa * fc < 0:
                fb = fc
                b = c
            elif fb * fc < 0:
                fa = fc
                a = c
            elif fc == 0:
                return c, True
            else:
                return c, False

        if abs(b - a) < tol:
            return b, True

    return b, False
```

Code for Testing:

```
def testHybridNewton():
    print("TESTING HYBRID NEWTON METHOD")
    print("----------------------------")
    x = sympy.symbols('x')
    expr = 10.14 * math.e ** (x * x) * sympy.cos(math.pi / x)
    sol = hybridNewton(expr, -3, 7, 0.001)
```

```
    print(f"The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range
[-3, 7] is {sol[0]}")
    print(f"CONVERGED: {sol[1]}.")
    print()

def allRootsHybridNewton(strict):
    print("TESTING ALL ROOTS FOR THE HYBRID NEWTON METHOD")
    print("-----------------------------------------------")
    x = sympy.symbols('x')
    expr = 10.14 * math.e ** (x * x) * sympy.cos(math.pi / x)
    intervals = [(-3, -1), (-1, -0.5), (-0.5, -0.3), (-0.3, 0.2), (0.2, 0.25),
(0.25, 0.35), (0.35, 0.5), (0.5, 0.7), (0.7, 7)]
    for interval in intervals:
        sol = hybridNewton(expr, interval[0], interval[1], 0.001,
strictInterval=strict)
        print(f"The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within
Range [{interval[0]}, {interval[1]}] is {sol[0]}")
        print(f"CONVERGED: {sol[1]}.")
    print()
```

Testing Output:

```
TESTING HYBRID NEWTON METHOD
----------------------------
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [-3, 7] is
2.00000000664768
CONVERGED: True.

TESTING ALL ROOTS FOR THE HYBRID NEWTON METHOD
-----------------------------------------------
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [-3, -1] is
0.285714201641803
CONVERGED: True.
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [-1, -0.5]
is 0.666666666665263
CONVERGED: True.
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [-0.5, -0.3]
is -0.285714285513292
CONVERGED: True.
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [-0.3, 0.2]
is -2.00000090705241
CONVERGED: True.
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [0.2, 0.25]
is -2.00000063850729
CONVERGED: True.
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [0.25, 0.35]
is 0.399999996895097
CONVERGED: True.
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [0.35, 0.5]
is -0.666666666665263
CONVERGED: True.
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [0.5, 0.7]
is 0.666666613719453
CONVERGED: True.
```

```
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [0.7, 7] is
2.00000000664768
CONVERGED: True.
```

Notes

The function techincally has infinite roots, so defining an interval for each is impossible.
You might notice that the function returns roots outside some of the intervals. If you want to prevent this, you simply make a call to the testing function with strict set to True.

Strict Output:

```
TESTING ALL ROOTS FOR THE HYBRID NEWTON METHOD
----------------------------------------------
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [-3, -1] is
-1.99951171875
CONVERGED: True.
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [-1, -0.5]
is -0.666666666664977
CONVERGED: True.
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [-0.5, -0.3]
is -0.399999999959820
CONVERGED: True.
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [-0.3, 0.2]
is -0.285714284192987
CONVERGED: True.
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [0.2, 0.25]
is 0.222222207910603
CONVERGED: True.
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [0.25, 0.35]
is 0.285714285286432
CONVERGED: True.
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [0.35, 0.5]
is 0.399999984493125
CONVERGED: True.
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [0.5, 0.7]
is 0.666666613719453
CONVERGED: True.
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [0.7, 7] is
2.00000000664768
CONVERGED: True.
```

Task 5

Code for Hybrid Secant Method:

```
'''
Approximates the Root of a Function Using the Secant Method, Falling Back to
Bisection if It Doesn't Converge
f: The Function in Question, a Lambda
a: Lower Boundary
b: Upper Boundary
tol: Maximum Permissible Error
maxIter: Maximum Iterations to Test Before Giving Up
```

```
maxTries: Maximum Times to Reduce the Problem With Bisection Before Giving Up
strictInterval: Boolean, Dictates Whether Convergence Must Happen Within the
Interval

Returns: The Approximate Root, Convergence Status
'''



def hybridSecant(f, a, b, tol, maxiter=1000, maxTries=10, strictInterval=False):
    error = 10 * tol

    for i in range(maxTries):

        # Setup for Secant Method
        x0 = b
        x1 = 10 * tol + x0
        x2 = 0

        f0 = f(x0)
        f1 = f(x1)
        for j in range(maxiter):

            x2 = x1 - f1 * (x1 - x0) / (f1 - f0)

            error = abs(x2 - x1)

            if error <= tol:
                break

            x0 = x1
            x1 = x2
            f0 = f1
            f1 = f(x1)

        # Returns Result if Convergence Happened
        if strictInterval:
            if error < tol and a < x1 < b:
                return x1, True
        elif error < tol:
            return x1, True

        # Reduces Interval Using Bisection if Failed to Converge
        fa = f(a)
        fb = f(b)

        for j in range(4):
            c = 0.5 * (a + b)
            fc = f(c)

            if fa * fc < 0:
                fb = fc
                b = c
            elif fb * fc < 0:
                fa = fc
                a = c
```

```
            elif fc == 0:
                return c, True
            else:
                return c, False

        # Returns Result if Bisection Managed to Converge, Otherwise Loops Back
to Secant Method
        if abs(b - a) < tol:
            return b, True

    return b, False
```

Code for Testing:

```
def testHybridSecant():
    print("TESTING HYBRID SECANT METHOD")
    print("----------------------------")
    expr = lambda x: 10.14 * math.e ** (x * x) * math.cos(math.pi / x)
    sol = hybridSecant(expr, -3, 7, 0.001)
    print(f"The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range
[-3, 7] is {sol[0]}")
    print(f"CONVERGED: {sol[1]}.")
    print()

def allRootsHybridSecant(strict):
    print("TESTING ALL ROOTS FOR HYBRID SECANT METHOD")
    print("-----------------------------------------")
    expr = lambda x: 10.14 * math.e ** (x * x) * math.cos(math.pi / x)
    intervals = [(-3, -1), (-1, -0.5), (-0.5, -0.3), (-0.3, 0.2), (0.2, 0.25),
(0.25, 0.35), (0.35, 0.5), (0.5, 0.7), (0.7, 7)]
    for interval in intervals:
        sol = hybridSecant(expr, interval[0], interval[1], 0.001,
strictInterval=strict)
        print(f"The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within
Range [{interval[0]}, {interval[1]}] is {sol[0]}")
        print(f"CONVERGED: {sol[1]}.")
    print()
```

Testing Output:

```
TESTING HYBRID SECANT METHOD
----------------------------
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [-3, 7] is
2.0010210919301907
CONVERGED: True.

TESTING ALL ROOTS FOR THE HYBRID NEWTON METHOD
-----------------------------------------------
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [-3, -1] is
-1.99951171875
CONVERGED: True.
```

```
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [-1, -0.5]
is -0.666666666664977
CONVERGED: True.
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [-0.5, -0.3]
is -0.399999999959820
CONVERGED: True.
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [-0.3, 0.2]
is -0.285714284192987
CONVERGED: True.
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [0.2, 0.25]
is 0.222222207910603
CONVERGED: True.
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [0.25, 0.35]
is 0.285714285286432
CONVERGED: True.
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [0.35, 0.5]
is 0.399999984493125
CONVERGED: True.
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [0.5, 0.7]
is 0.666666613719453
CONVERGED: True.
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [0.7, 7] is
2.00000000664768
CONVERGED: True.
```

Notes

The function techincally has infinite roots, so defining an interval for each is impossible.

You might notice that the function returns roots outside some of the intervals. If you want to prevent this, you simply make a call to the testing function with strict set to True.

Strict Output:

```
TESTING ALL ROOTS FOR HYBRID SECANT METHOD
------------------------------------------
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [-3, -1] is
-1.9998876271473107
CONVERGED: True.
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [-1, -0.5]
is -0.6666758963111825
CONVERGED: True.
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [-0.5, -0.3]
is -0.39921875
CONVERGED: True.
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [-0.3, 0.2]
is -0.28506357128807963
CONVERGED: True.
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [0.2, 0.25]
is 0.222265625
CONVERGED: True.
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [0.25, 0.35]
is 0.2855840834490296
CONVERGED: True.
```

```
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [0.35, 0.5] is 0.39969147979083564
CONVERGED: True.
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [0.5, 0.7] is 0.6658546859864347
CONVERGED: True.
The Approximate Root of 10.14 * e ^ (x ^ 2) * cos(PI/x) Within Range [0.7, 7] is 2.0010210919301907
CONVERGED: True.


Process finished with exit code 0
```