Assignment 01

Problem 1

Finding the Roots of f(x) = x * e ** -x Using Fixed Point Iteration

Fixed Point Iteration Code:

```
def fixedPointIteration(g, x0, tol, maxIterations=100):
    x1 = x0
    error = 10 * tol

    try:
        for i in range(maxIterations):
            x1 = g(x0)
            error = math.fabs(x1 - x0)

            if error <= tol:
                break
            x0 = x1

    except OverflowError:
        return math.inf, False

    return x1, error <= tol
```

This function approximates the root of a (mathematical) function using the fixed point iteration method. It excepts serveral parameters:

* g: The Modified Fixed Point Version of the Function
* x0: Initial Guess for the Root
* tol: Maximum Permissible Error
* maxIterations: Maximum Steps to Try Before Giving Up

The code returns the approximation of the root, as well as True/False depending on whether convergence was reached within the maximum number of iterations.

Problem Code:

```
import math

def problem1():
    # Original Function
    f = lambda x: x * math.e ** -x

    # Fixed Point Alterations
    g1 = lambda x: x - f(x)
    g2 = lambda x: x + f(x)

    roots1 = []
```

```
    roots2 = []


    # Tests Initial Points from -5 to 5 for the First Fixed Point Alteration
    print("Approximating the Roots of f(x) = x * e ** x Using g(x) = x - f(x):")
    print("-----------------------------------------------------------------------")
    for i in range(-5, 6):
        print(f"    Using Initial Guess x0 = {i}")
        approximation, converges = fixedPointIteration(g1, i, 0.00000001)
        if converges:
            print(f"        SUCCESS. Result: {approximation}")
            roots1.append(approximation)
        else:
            print(f"        FAILED. Result: {approximation}")

    print("Roots Found:")
    print("------------")
    for root in roots1:
        print(f"x={root}")
    print("\n\n\n\n")


    # Tests Initial Points from -5 to 5 for the Second Fixed Point Alteration
    print("Approximating the Roots of f(x) = x * e ** x Using g(x) = x + f(x):")
    print("-----------------------------------------------------------------------")

    for i in range(-5, 6):
        print(f"    Using Initial Guess x0 = {i}")
        approximation, converges = fixedPointIteration(g2, i, 0.000000001)
        if converges:
            print(f"        SUCCESS. Result: {approximation}")
            roots2.append(approximation)
        else:
            print(f"        FAILED. Result: {approximation}")

    print("Roots Found:")
    print("------------")
    for root in roots2:
        print(f"x={root}")
    print("\n\n\n\n")
```

This function sets up the original (mathematical) function as a Python lambda. This allows us to pass it into our root finding method, or into other lambdas to define our initial g(x).

For our case, we use two separate g(x) functions. The first adding the original function to x, and the second subtracting. This should give us more opportunities for convergence. After that, it attempts to find convergence at each integer step on an interval from [-5, 6] for both g(x) functions, keeping track of any roots found, while reporting both intermittent and final results.

This code can be run by navigating to the directory in this project containing the *problems.py* file, and running *python problems.py* from the command line. It should be noted that this will also run the test code for problem 3.

Output

```
Approximating the Roots of f(x) = x * e ** x Using g(x) = x - f(x):
----------------------------------------------------------------------
    Using Initial Guess x0 = -5
        SUCCESS. Result: 737.0657955128829
    Using Initial Guess x0 = -4
        SUCCESS. Result: 214.39260013257692
    Using Initial Guess x0 = -3
        SUCCESS. Result: 57.256610769563
    Using Initial Guess x0 = -2
        FAILED. Result: 12.774536617781685
    Using Initial Guess x0 = -1
        SUCCESS. Result: 2.465190328815662e-32
    Using Initial Guess x0 = 0
        SUCCESS. Result: 0.0
    Using Initial Guess x0 = 1
        SUCCESS. Result: 8.909872570538368e-19
    Using Initial Guess x0 = 2
        SUCCESS. Result: 9.860761315262648e-32
    Using Initial Guess x0 = 3
        SUCCESS. Result: 1.8755956882534777e-26
    Using Initial Guess x0 = 4
        SUCCESS. Result: 1.1768542528449143e-25
    Using Initial Guess x0 = 5
        SUCCESS. Result: 9.50419618610275e-27
Roots Found:
------------
x=737.0657955128829
x=214.39260013257692
x=57.256610769563
x=2.465190328815662e-32
x=0.0
x=8.909872570538368e-19
x=9.860761315262648e-32
x=1.8755956882534777e-26
x=1.1768542528449143e-25
x=9.50419618610275e-27




Approximating the Roots of f(x) = x * e ** x Using g(x) = x + f(x):
----------------------------------------------------------------------
    Using Initial Guess x0 = -5
        FAILED. Result: inf
    Using Initial Guess x0 = -4
        FAILED. Result: inf
    Using Initial Guess x0 = -3
        FAILED. Result: inf
    Using Initial Guess x0 = -2
        FAILED. Result: inf
    Using Initial Guess x0 = -1
        FAILED. Result: inf
```

```
    Using Initial Guess x0 = 0
        SUCCESS. Result: 0.0
    Using Initial Guess x0 = 1
        FAILED. Result: 6.237041784975049
    Using Initial Guess x0 = 2
        FAILED. Result: 6.272247336842037
    Using Initial Guess x0 = 3
        FAILED. Result: 6.32659000325185
    Using Initial Guess x0 = 4
        FAILED. Result: 6.428172965212631
    Using Initial Guess x0 = 5
        FAILED. Result: 6.6216446133731575
Roots Found:
------------
x=0.0
```

ACTUAL FUNCTION ROOTS: x=0.

After we run run it, you can see a lot more results than we'd expect, as the function has only one root at zero. However, all the roots it finds are either arbitrarily close to zero, in which case it makes sense as they are within error, or way off in the positive axis.

This makes sense as well, because if we graph the function, we can see that it approaches the x axis as you go left. This means that our algorithm will find "roots" there, because at a certain precision there's no difference between touching and almost touching the x-axis.

Problem 2

Adding a Verbose Mode to the Fixed Point Iteration Code

The code is basically the same as in problem 1. All that's needed is a single extra parameter, as you can see below, which toggles a debug print statement.

Fixed Point Iteration Code:

```
def fixedPointIteration(g, x0, tol, maxIterations=100, v=False):
    x1 = x0
    error = 10 * tol

    try:
        for i in range(maxIterations):
            x1 = g(x0)
            error = math.fabs(x1 - x0)

            if v:
                print(f"iteration: {i}. xApprox: {x1:.4E}. error: {error:.4E}.")

            if error <= tol:
                break
            x0 = x1

    except(OverflowError):
        return math.inf, False
```

```

```

Parameters Added:
  * v: Toggles Verbose Mode, which Prints Debug Information

Function Usage:

The function usage remains identical to before, except it's now possible to optionally specify verbose mode. In this case, we're testing for convergence of f(x) = x ** 2 - 1, using g(x) = x - f(x), and an initial guess of 3.

```
fixedPointIteration(lambda x: x - (x ** 2 - 1), 3, 0.0001, v=True)
```

Output

```
iteration: 0. xApprox: -5.0000E+00. error: 8.0000E+00.
iteration: 1. xApprox: -2.9000E+01. error: 2.4000E+01.
iteration: 2. xApprox: -8.6900E+02. error: 8.4000E+02.
iteration: 3. xApprox: -7.5603E+05. error: 7.5516E+05.
iteration: 4. xApprox: -5.7158E+11. error: 5.7158E+11.
iteration: 5. xApprox: -3.2670E+23. error: 3.2670E+23.
iteration: 6. xApprox: -1.0674E+47. error: 1.0674E+47.
iteration: 7. xApprox: -1.1393E+94. error: 1.1393E+94.
iteration: 8. xApprox: -1.2979E+188. error: 1.2979E+188.
(inf, False)
```

As we can see, at each iteration it prints out the current approximation of x. In this case, however, since our function not only doesn't converge, but also reaches extremely large values extremely fast, the function cuts off before reaching the maximum number of iterations, as it knows convergence isn't possible.

Problem 3

Finding the Roots of f(x) = 10.14 * e ** (x ** 2) * cos(PI / x) Using Fixed Point Iteration

Fixed Point Iteration Code:

```
Same as in Problem 2
```

Problem Code:

```
import math

def problem3():
    # Original Function
    f = lambda x: 10.14 * math.e ** (x * x) * math.cos(math.pi / x)

    # Fixed Point Alteration
```

```
    g = lambda x: x - f(x)

    roots = []

    # Tests Initial Points from -3 to 7 for the Fixed Point Alteration
    print("Approximating the Roots of f(x) = 10.14 * e ** (x * x) * math.cos(pi /
x) Using g(x) = x - f(x):")
    print("------------------------------------------------------------------------
-------------------------")
    for i in range(-3, 8):
        if i == 0:
            i += 0.000001
        print(f"    Using Initial Guess x0 = {i}")
        approximation, converges = fixedPointIteration(g, i, 0.00000001)
        if converges:
            print(f"        SUCCESS. Result: {approximation}")
            roots.append(approximation)
        else:
            print(f"        FAILED. Result: {approximation}")

    print("Roots Found:")
    print("------------")
    for root in roots:
        print(f"x={root}")
    print("\n\n\n\n")
```

This code is extremely similar to the code to run problem 1. We again use a lambda to set up our function, but in this case, we only use one g(x), namely the x - f(x) version. We then iterate over the range [-2, 7], checking every integer for convergence using fixed point iteration. In thise case, since x=0 is undefined, we slightly offset our x when checking that value. It again tracks keeps track of roots found, and reports success both intermittently and at the end.

This code can be run by navigating to the directory in this project containing the *problems.py* file, and running *python problems.py* from the command line. It should be noted that this will also run the test code for problem 1.

Output

```
Approximating the Roots of f(x) = 10.14 * e ** (x * x) * math.cos(pi / x) Using
g(x) = x - f(x):
------------------------------------------------------------------------------
---------------
    Using Initial Guess x0 = -3
        FAILED. Result: inf
    Using Initial Guess x0 = -2
        SUCCESS. Result: -2.0000000000000338
    Using Initial Guess x0 = -1
        FAILED. Result: -inf
    Using Initial Guess x0 = 1e-06
        FAILED. Result: inf
    Using Initial Guess x0 = 1
        FAILED. Result: inf
    Using Initial Guess x0 = 2
```

```
        SUCCESS. Result: 1.999999999999966
    Using Initial Guess x0 = 3
        FAILED. Result: inf
    Using Initial Guess x0 = 4
        FAILED. Result: inf
    Using Initial Guess x0 = 5
        FAILED. Result: inf
    Using Initial Guess x0 = 6
        FAILED. Result: inf
    Using Initial Guess x0 = 7
        FAILED. Result: inf
Roots Found:
------------
x=-2.0000000000000338
x=1.999999999999966
```

ACTUAL FUNCTION ROOTS: x=-2, x=2

In this case, we didn't get any "weird" results, which is very nice. As we can see, however, this function is pretty hard to work with. It only actually converges when we get very close to the roots, so another g(x) might give better results, or we could give bisection a try.

Problem 4

Implementing the Bisection Method

Bisection Code:

```
import math

def bisect(f, a, b, tol, v=False):
    if f(a) * f(b) >= 0:
        raise Exception("ERROR: f(a) and f(b) Must be On Opposite Sides of the x
Axis")

    iterations = math.ceil(math.log2((b - a) / tol))
    fa = f(a)
    fb = f(b)
    c = (a + b) / 2

    for i in range(iterations):
        c = (a + b) / 2
        fc = f(c)

        if v:
            print(f"iteration: {i}. xApprox: {c:.4E}. error: {math.fabs(b -
c):.4E}")

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

```
            a = c
        elif fc == 0:
            break
        else:
            raise Exception("ERROR: f(a) and f(b) Must be On Opposite Sides of
the x Axis")

    return c
```

The function approximates the root of a (mathematical) function using the bisection method. It accepts the following parameters:

* f: The Function in Question
  * a: Start of the Interval Containing at Least One Root
  * b: End of the Interval Containing at Least One Root
  * tol: Maximum Permissible Error
  * -v: Verbose Mode, Prints Debug Info

It returns the approximate root, which is guaranteed to be within the tolerance value due to it's implementation. It should be noted that f(a) and f(b) must be on opposite sides of the x-axis, otherwise, an error will be raised.

Function Example:

The function can be called by specifying the necessary information of f, a, b, and tol. In this case, we're using f(x) = x ** 2 - 1, and an interval from [0, 5] with a tolerance of 0.00001. Verbose mode is also toggled on to show debug info.

```
print(bisect(lambda x: x **2 - 1, 0, 5, 0.00001, True))
print(bisect(lambda x: x **2 - 1, -5, 5, 0.00001, True))
```

Output

```
iteration: 0. xApprox: 2.5000E+00. error: 2.5000E+00
iteration: 1. xApprox: 1.2500E+00. error: 1.2500E+00
iteration: 2. xApprox: 6.2500E-01. error: 6.2500E-01
iteration: 3. xApprox: 9.3750E-01. error: 3.1250E-01
iteration: 4. xApprox: 1.0938E+00. error: 1.5625E-01
iteration: 5. xApprox: 1.0156E+00. error: 7.8125E-02
iteration: 6. xApprox: 9.7656E-01. error: 3.9062E-02
iteration: 7. xApprox: 9.9609E-01. error: 1.9531E-02
iteration: 8. xApprox: 1.0059E+00. error: 9.7656E-03
iteration: 9. xApprox: 1.0010E+00. error: 4.8828E-03
iteration: 10. xApprox: 9.9854E-01. error: 2.4414E-03
iteration: 11. xApprox: 9.9976E-01. error: 1.2207E-03
iteration: 12. xApprox: 1.0004E+00. error: 6.1035E-04
iteration: 13. xApprox: 1.0001E+00. error: 3.0518E-04
iteration: 14. xApprox: 9.9991E-01. error: 1.5259E-04
iteration: 15. xApprox: 9.9998E-01. error: 7.6294E-05
iteration: 16. xApprox: 1.0000E+00. error: 3.8147E-05
iteration: 17. xApprox: 1.0000E+00. error: 1.9073E-05
iteration: 18. xApprox: 9.9999E-01. error: 9.5367E-06
0.9999942779541016
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "G:\School\Fall 2022\MATH 4610\math4610\Assignment01\rootfinding.py", line
15, in bisect
    raise Exception("ERROR: f(a) and f(b) Must be On Opposite Sides of the x
Axis")
Exception: ERROR: f(a) and f(b) Must be On Opposite Sides of the x Axis
```

As expected, with proper setup this function correctly finds (one of) the roots at x=1. When we call it with values that result in f(a) and f(b) being on opposite sides of the x-axis, however, it results in an error as specified.

Problem 5

My GitHub link is:

```
https://github.com/jfitzusu/math4610
```

I sent you an invite.