Question 1D)

Using the Matlab code provided, along with some additional calculations, here are the roots of the Wilkinson Polynomial, along with the relative error for each root:

Root	True root	Relative Error
20.8469 + 0.0000i	20	0.7157
19.5024 + 1.9403i	19	1.8016
19.5024 - 1.9403i	18	2.2058
16.7307 + 2.8126i	17	2.9338
16.7307 - 2.8126i	16	3.0174
13.9924 + 2.5188i	15	3.3614
13.9924 - 2.5188i	14	3.1210
11.7939 + 1.6525i	13	3.0262
11.7939 - 1.6525i	12	2.4633
10.0955 + 0.6449i	11	1.9346
10.0955 - 0.6449i	10	1.1353
8.9158 + 0.0000i	9	0.1663
8.0078 + 0.0000i	8	0.0171
6.9996 + 0.0000i	7	0.0010
6.0000 + 0.0000i	6	0.0001
5.0000 + 0.0000i	5	0.0000
4.0000 + 0.0000i	4	0.0000
3.0000 + 0.0000i	3	0.0000
2.0000 + 0.0000i	2	0.0000
1.0000 + 0.0000i	1	0.0000

Question 5 Part 1 Newton's Method:

```
#Code
#Note: To run for each function, replace f with g or h and df with
#dg/dh
import numpy as np

#Function and derivative for part 1
f = lambda x : np.log(x)+x**2 - 3
df = lambda x: 1/x +2*x

#Function and derivative for part 2
g = lambda x: x**5+x-1
dg = lambda x: 5*x**4+1

#Function and derivative for part 3
h = lambda x: np.sin(x)-6*x-5
dh = lambda x: np.cos(x) - 6

#Note that
def newton(x0, epsilon): #Newton's method
    x1 = x0
    x = x1 + 50*epsilon #Just guarantees runs first iteration
    iter = 0
    while (abs(x1-x) > epsilon): #If > error bound, hasn't converged
        x = x1
        x1 = x1 - f(x1)/df(x1) #Updates new point with function f
        print("       ", iter, "|", x1, "|", f(x1)) #Display
        iter += 1
    return(x1)

x = 3 #Starting point
epsilon = .000001 #Error to end algorithm
print("Iterations|", "x |", "f(x)")
print("Final value:", newton(x, epsilon))

#END CODE
```

```
Part 1) Print out result for x^5+x =1
Iteration | x | h(x)
        0 | 2.396551724137931 | 3.6174910890527627
        1 | 1.9117137201111394 | 1.3026494231046835
        2 | 1.5215611067915087 | -0.2651149465482239
        3 | 1.2094340525058087 | -1.3471167476500898
        4 | 0.9703214785442607 | -2.088604069488606
        5 | 0.8174444097911604 | -2.5333570156794933
```

```
        6 | 0.7610052328597109 | -2.6939860804102604
        7 | 0.7549389861646737 | -2.711185473194238
        8 | 0.7548776724115093 | -2.7113592658509007
        9 | 0.7548776662466928 | -2.7113592833249083
Final value: 0.7548776662466928
```
Rounded to 8 decimal places, the root is at x = 0.75487766

```
Part 2) Print out result for sin(x)=6x+5
Iteration | x | h(x)
        0 | -0.27022954646338393 | -3.645575381080108
        1 | -0.994090786221374 | 0.1262811673085089
        2 | -0.9709400422003657 | 0.00022349899407547724
        3 | -0.9708989236326309 | 6.977751709769109e-10
        4 | -0.9708989235042558 | 0.0
Final value: -0.9708989235042558
```
Rounded to 8 decimal places, the root is at x = -0.97089892

```
Part 3) Print out result for ln(x)+x^2=3
Iteration | x | f(x)
        0 | 1.8791664807366144 | 1.1620949795418927
        1 | 1.6083124147681183 | 0.06185426317372
        2 | 1.592197796540965 | 0.0002091470728369238
        3 | 1.5921429376917977 | 2.4159136913226575e-09
        4 | 1.592142937058094 | 4.440892098500626e-16
Final value: 1.592142937058094
```
Rounded to 8 decimal places, the root is at x = 1.59214293

Question 5 Part 2 Secant Method:

#**CODE**

```
f = lambda x: x**3+x-1 #Function
def secant(x0, x1, epsilon):
    dummy = 0
    for i in range(1000):
        print(dummy,"| " ,x1)
        dummy = x1
        if abs(f(x1)-f(x0)) < epsilon: #If in epsilon bound, end
            break
        x1 = x1 - f(x1)*(x1-x0)/(f(x1)-f(x0)) #Updates new point
        x0 = dummy
    return x1

x0 = 0 #Starting value
x1 = 1
epsilon = 0.0000000000001 #Error bound
```

```
print("iteration| x0 | x1")
print("Final value:", secant(x0, x1, epsilon))
```

#END CODE

```
Print out result for f(x)= x^3+x-1:
iteration| x0 | x1
0 |  1
1 |  0.5
0.5 |  0.6363636363636364
0.6363636363636364 |  0.6900523560209424
0.6900523560209424 |  0.6820204196481856
0.6820204196481856 |  0.6823257814098928
0.6823257814098928 |  0.6823278043590257
0.6823278043590257 |  0.6823278038280184
0.6823278038280184 |  0.6823278038280193
Final value: 0.6823278038280193
```

Rounded to 8 decimal places, the root is at x = 0.68232780

Question 5 Part 3 Bisection Method:

#CODE

```
import numpy as np
import math

f = lambda x: (math.e)**x - np.sin(x) #Function

def bisection(a, b, epsilon):
    c1 = 0
    iter = 0
    while f(a)*f(b) < 0:
        c = (a+b)/2 #New point
        if (abs(c-c1) < epsilon): #If below error bound we are done
            break
        c1 = c #Temp var to see how close we are to convergence
        if f(c)*f(a) < 0: #Update value
            b = c
        elif f(c)*f(b) < 0: #Update value
            a = c
        elif f(c) == 0: #If we're at the root
            break
        print(iter, "|", c)
        iter += 1
    return c

a = -4 #Starting point
```

```
b = -1
epsilon = 0.000001 #Error bound

print("iteration | midpoint")
print(bisection(a, b, epsilon))
```
#END CODE

Print out of result for root of e^x = sin(x) closest to 0:

```
iteration | midpoint
0 | -2.5
1 | -3.25
2 | -2.875
3 | -3.0625
4 | -3.15625
5 | -3.203125
6 | -3.1796875
7 | -3.19140625
8 | -3.185546875
9 | -3.1826171875
10 | -3.18408203125
11 | -3.183349609375
12 | -3.1829833984375
13 | -3.18316650390625
14 | -3.183074951171875
15 | -3.1830291748046875
16 | -3.1830520629882812
17 | -3.183063507080078
18 | -3.1830577850341797
19 | -3.183060646057129
20 | -3.1830620765686035
```
Rounded to 8 decimal places, the root is -3.18306207