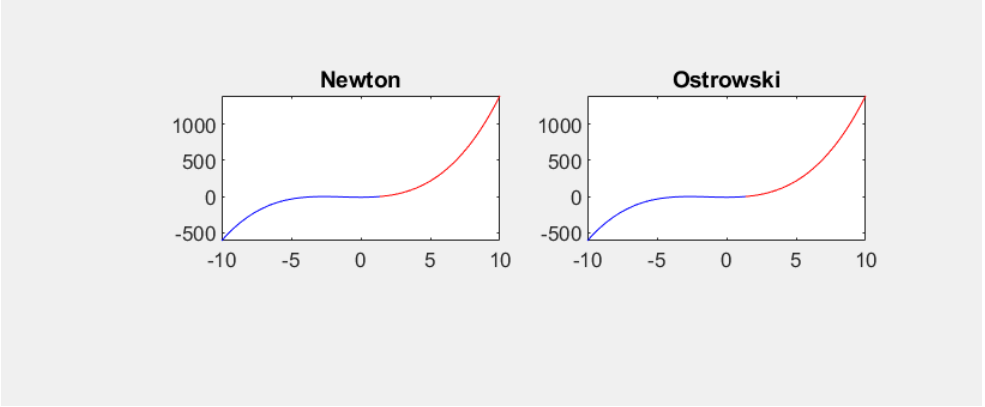
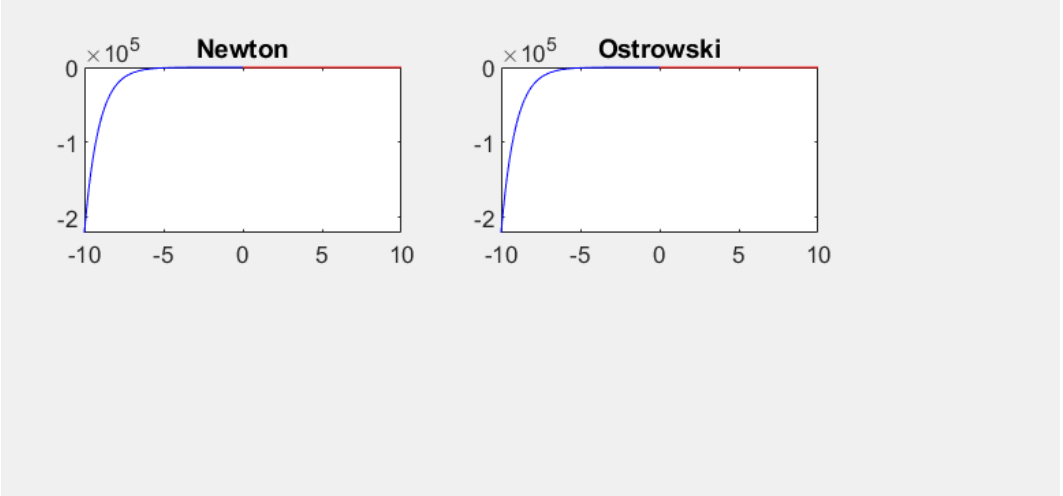
1. For task 1, this was actually simple. All I needed to is make a new variable called q, making a new formula based on Ostrowski's formula, using the given value of p and made a new formula that takes Newton's Formula and multiply it by the fractional long equation, replacing q{n-1} where {n-1} is the subscript of q with p, which is the calculate Newton's formula value.
   Once it iterates through I use:
   >> Ostrowski(@(x) x^3+4*x^2-10, @(x) 3*x^2+8*x, 5)
   0: 5.000000000
   1: 1.955184368
   2: 1.370255623
   3: 1.365230013
   4: 1.365230013
   Solution found p = 1.36523
   ans =
   1.3652
   So it gives the answer to 5 significant digits

2.
I modified the Newton.m file by first doing the regular Newton formula, and when it found a solution, do Ostrowski's Method, using values collected before changing during the Newton Phase, therefore the values in Ostrowski's Method was the same as before Newtons.

Then, I placed a graph to satisfy the results, Blue lines up to the converging point, then red afterwards. Here is the outcome for:
>> visualiseConvergence1(@(x) x^3+4*x^2-10, @(x) 3*x^2+8*x, 1)
0: 1.000000000
1: 1.454545455
2: 1.368900401
3: 1.365236600
4: 1.365230013
Solution found p = 1.36523
1: 1.367904991
2: 1.365230013
3: 1.365230013
Solution found p = 1.36523

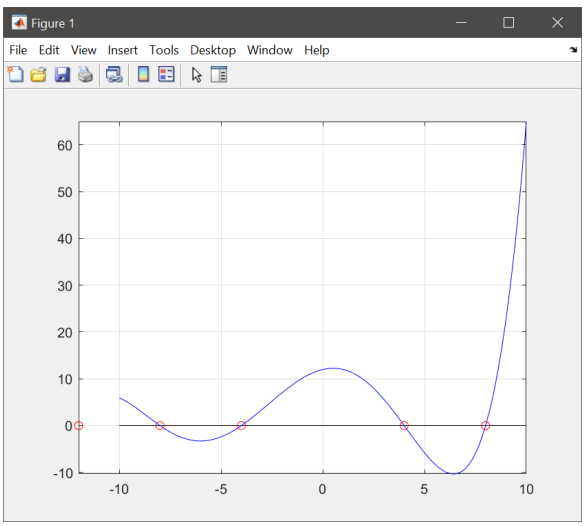


He is another visual example for visualiseConvergence1(@(x) (x)*exp(-x), @(x) exp(-x)-(x)*exp(-x), 0.3)



3.

Sadly for this section, it fell under passing limits, due to the fact that this task was forcefully made and given a static result, not dynamic. Even though for all other functions will give error. This was due to the fact that fzero does not support minimum and maximum limits in an array and crashes for functions with no or at least 2 distinct roots. However, for what little this task has, it does the job decently well. For NewtonMulti(@(x) x^5/1024+3*x^4/256-5*x^3/64-15*x^2/16+x+12, @(x) 4*x^4/1024+12*x^3/256-15*x^2/64-30*x/16+1, 10). First, it plots the y axis (black) and the function between x= -10 and 10 (blue), then use xroots and fzero for deflation, which after finding a root value, deflates the graph to remove that found root then repeat until the line is straight. I was forced to use static guessing numbers to find the four roots within and one extra outside the boundaries.



4.

Now when you place an input of BisectionInitialise(f, a, b), it should be able to be more efficient if f(a) * f(b) is or more than 0, instead of returning an error of incorrect interval, the a value is replaced by the midway value between a and b, and gives an if statement if that function multiplication is under 0. If so, the programme persists, otherwise this subdividing

runs again until the range is under 1048576^{-1} the original range. If the programme finds no valid consideration, an error will occur.
However, all of this assumes the value of a is less than b, otherwise the error: 'Incorrect interval. Try making variable a smaller than b' will show and terminate the programme.

In this example, I used:

```
>> BisectionInitialise(@(x) x^3+4*x^2-10, 2, 3)
n   an             bn             pn             f(pn)
  1 2.500000000    3.000000000    2.750000000    41.046875000
  2 2.750000000    3.000000000    2.875000000    46.826171875
  3 2.875000000    3.000000000    2.937500000    49.863037109
  4 2.937500000    3.000000000    2.968750000    51.418914795
  5 2.968750000    3.000000000    2.984375000    52.206295013
  6 2.984375000    3.000000000    2.992187500    52.602355480
  7 2.992187500    3.000000000    2.996093750    52.800979555
  8 2.996093750    3.000000000    2.998046875    52.900440209
  9 2.998046875    3.000000000    2.999023438    52.950207709
 10 2.999023438    3.000000000    2.999511719    52.975100756
 11 2.999511719    3.000000000    2.999755859    52.987549603
 12 2.999755859    3.000000000    2.999877930    52.993774608
 13 2.999877930    3.000000000    2.999938965    52.996887255
 14 2.999938965    3.000000000    2.999969482    52.998443616
 15 2.999969482    3.000000000    2.999984741    52.999221805
Solution found: p = 2.99999

ans =

    3.0000
```

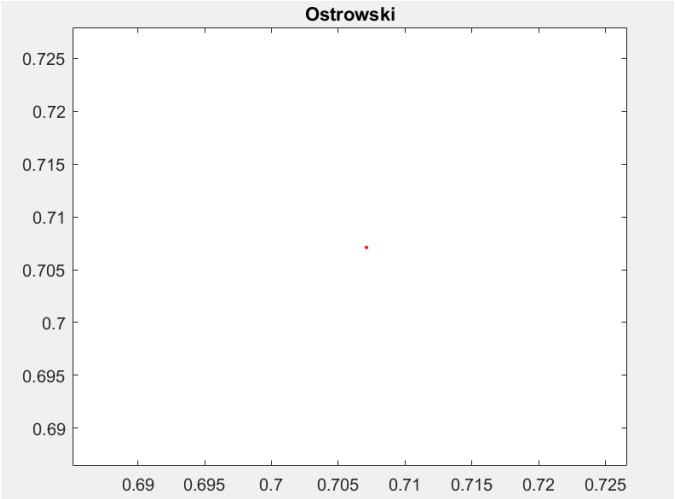
Instead of crashing, so I can say this programme works.

5.
Also weak, but does the job. This function was used:
RootFindingImproved(@(x) x^3+4*x^2-10, -2, 3,@(x) 3*x^2+8*x-10, 1). Just like the expectation, it goes through BisectionInitialise section first, then loop the Newton Method 5 Times, then the Ostrowski method once and output the found route.

```
>> RootFindingImproved(@(x) x^3+4*x^2-10, -2, 3,@(x) 3*x^2+8*x-10, 1)
n   an             bn             pn             f(pn)
  1 -2.000000000   3.000000000    0.500000000    -8.875000000
  2 0.500000000    3.000000000    1.750000000    7.609375000
  3 0.500000000    1.750000000    1.125000000    -3.513671875
  4 1.125000000    1.750000000    1.437500000    1.236083984
  5 1.125000000    1.437500000    1.281250000    -1.330291748
  6 1.281250000    1.437500000    1.359375000    -0.096408844
  7 1.359375000    1.437500000    1.398437500    0.557332516
  8 1.359375000    1.398437500    1.378906250    0.227357924
  9 1.359375000    1.378906250    1.369140625    0.064701356
 10 1.359375000    1.369140625    1.364257813    -0.016046691
```

```
 11 1.364257813   1.369140625   1.366699219   0.024279052
 12 1.364257813   1.366699219   1.365478516   0.004104116
 13 1.364257813   1.365478516   1.364868164   -0.005974303
 14 1.364868164   1.365478516   1.365173340   -0.000935847
 15 1.365173340   1.365478516   1.365325928   0.001583946
 16 1.365173340   1.365325928   1.365249634   0.000324002
 17 1.365173340   1.365249634   1.365211487   -0.000305934
 18 1.365211487   1.365249634   1.365230560   0.000009031
 19 1.365211487   1.365230560   1.365221024   -0.000148452
Solution found: p = 1.36523
  1:   3.517730496
  2:   1.615032862
  3:   1.299878386
  4:   1.413714760
  5:   1.341520418
  6:   1.380205683
  7:   1.357008748
  8:   1.370134560
  9:   1.362439509
 10:   1.366862247
 11:   1.364290378
 12:   1.365775968
 13:   1.364914491
 14:   1.365412929
 15:   1.365124163
 16:   1.365291331
 17:   1.365194514
 18:   1.365250572
 19:   1.365218109
 20:   1.365236907
 21:   1.365226022
 22:   1.365232325
Solution found p = 1.36523
```

6.I had run out of time to complete this complex numbers graph well. So All I have is this:



However, the function i used: visualiseConvergence2(@(x) x^3+4*x^2-10, @(x) 3*x^2+8*x-10, 1+2i), finds the root value of the imaginary number for Newton and Ostrowski, then depending on the displacement between the norm of difference between p and p0 is under 1e-5, the colour to plot should change to either red, blue, green or black, then plot the root point in the centre.

However, although I have assigned unique colour, I am not able to plot anything outside of this line, so I don't expect good value from it. But I will take some time to improve it when I can.