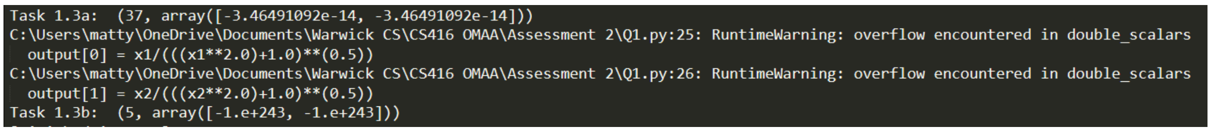
CS416 Assessment 2

Task 1.

The focus of this assessment is to explore how different implementations of Newton's method converge to roots, using a variety of functions and starting points. First, we look at the standard 'Pure' Newton's method, on a given function $g(x,y) = \sqrt{x^2+1} + \sqrt{y^2+1}$. The Hessian of this function is $H = \begin{matrix} (x^2+1)^{-\frac{3}{2}} & 0 \\ 0 & (y^2+1)^{-\frac{3}{2}} \end{matrix}$. This matrix is clearly positive definite, as it is a diagonal matrix, and so the eigenvectors are the values on the diagonals; these values are always positive. The minimum point is at the root of the derivative $D = \frac{x}{\sqrt{x^2+1}} \quad \frac{y}{\sqrt{y^2+1}}$. Setting D = 0 it is clear that the only solution is the point (0,0). One way of checking if a function is m-strongly convex is checking whether the eigenvalues of the Hessian are all greater than m. Here the eigenvalues are the values on the diagonal of H, which approach 0 as x, y increase, and so g is not m-strongly convex.

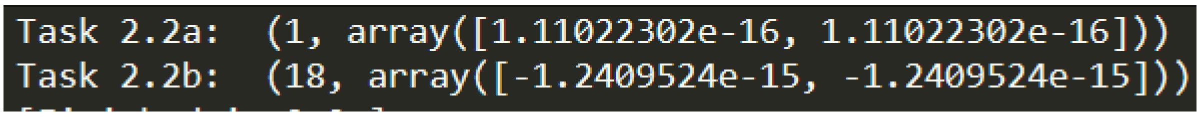


```
Task 1.3a:  (37, array([-3.46491092e-14, -3.46491092e-14]))
C:\Users\matty\OneDrive\Documents\Warwick CS\CS416 OMAA\Assessment 2\Q1.py:25: RuntimeWarning: overflow encountered in double_scalars
  output[0] = x1/(((x1**2.0)+1.0)**(0.5))
C:\Users\matty\OneDrive\Documents\Warwick CS\CS416 OMAA\Assessment 2\Q1.py:26: RuntimeWarning: overflow encountered in double_scalars
  output[1] = x2/(((x2**2.0)+1.0)**(0.5))
Task 1.3b:  (5, array([-1.e+243, -1.e+243]))
```

Task 1.3: (1,1) converges, (10,10) diverges hence the error

Task 2.

Secondly, we implement the Damped Newton's Method, to compare the results with the pure Newton's method.

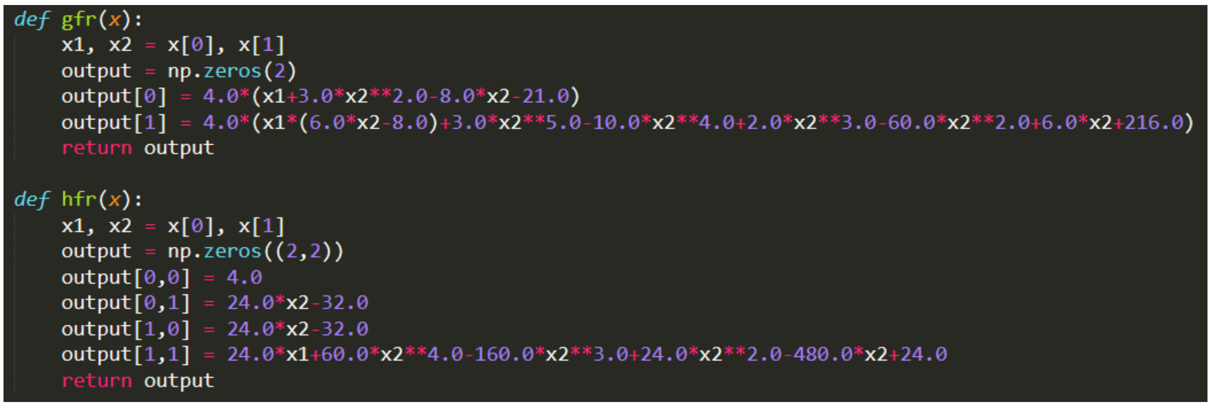


```
Task 2.2a:  (1, array([1.11022302e-16, 1.11022302e-16]))
Task 2.2b:  (18, array([-1.2409524e-15, -1.2409524e-15]))
```

Task 2.2: This time both converge, and (1,1) in fewer iterations

Task 3.

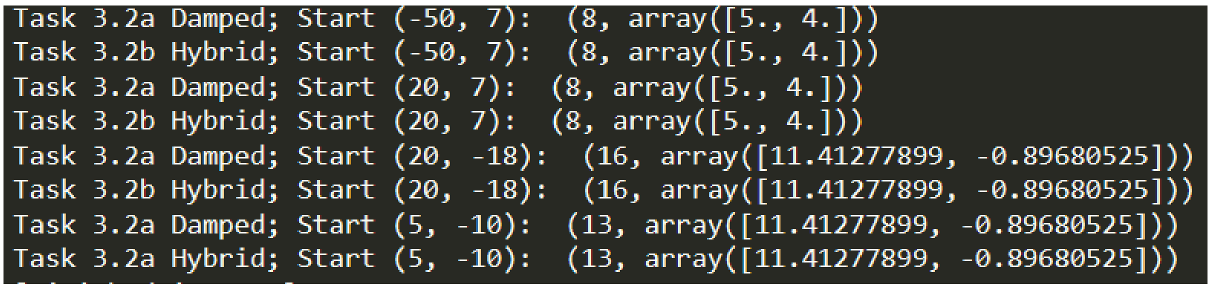
Finally, we address the issue of the Hessian potentially not being positive definite; we do this by running a version of Newton's method that instead runs gradient descent where the Hessian is not PD. Here we test on the Freudenstein and Roth test function, and compare the results with that of the Damped method. On the following page are the computations of Gradient and Hessian of this function, and how it performed on the inputs given.



```python
def gfr(x):
    x1, x2 = x[0], x[1]
    output = np.zeros(2)
    output[0] = 4.0*(x1+3.0*x2**2.0-8.0*x2-21.0)
    output[1] = 4.0*(x1*(6.0*x2-8.0)+3.0*x2**5.0-10.0*x2**4.0+2.0*x2**3.0-60.0*x2**2.0+6.0*x2+216.0)
    return output

def hfr(x):
    x1, x2 = x[0], x[1]
    output = np.zeros((2,2))
    output[0,0] = 4.0
    output[0,1] = 24.0*x2-32.0
    output[1,0] = 24.0*x2-32.0
    output[1,1] = 24.0*x1+60.0*x2**4.0-160.0*x2**3.0+24.0*x2**2.0-480.0*x2+24.0
    return output
```

Computation of the Gradient and Hessian of the FR Test function



```
Task 3.2a Damped; Start (-50, 7):  (8, array([5., 4.]))
Task 3.2b Hybrid; Start (-50, 7):  (8, array([5., 4.]))
Task 3.2a Damped; Start (20, 7):  (8, array([5., 4.]))
Task 3.2b Hybrid; Start (20, 7):  (8, array([5., 4.]))
Task 3.2a Damped; Start (20, -18):  (16, array([11.41277899, -0.89680525]))
Task 3.2b Hybrid; Start (20, -18):  (16, array([11.41277899, -0.89680525]))
Task 3.2a Damped; Start (5, -10):  (13, array([11.41277899, -0.89680525]))
Task 3.2a Hybrid; Start (5, -10):  (13, array([11.41277899, -0.89680525]))
```

Task 3.2: As the Hessian is always PD, the Damped and Hybrid functions operate equivalently