Q4

I rewrote the code from Matlab to Python (cause I don't know Matlab). And note that the system warnings for the original code refer to iterations $n = 16$ and $n = 32$. There are no system warning for my section of the code.

```
import numpy as np
import scipy.interpolate as sc
from math import pi, log
import pandas as pd

a = 0
b = 3.1
xe = np.linspace(a, b, 1000)
ye = np.cos(pi*xe)

print(['interval (' + str(a) + ', ' + str(b)+ ')'])
print('n    err poly   err lin_spl')
data1 = []
for nn in range(1, 7):
    n = 2**nn
    xi = np.linspace(a, b, n+1)
    yi = np.cos(pi*xi)
    yp = np.polyval(np.polyfit(xi, yi, n), xe)
    yl = sc.interp1d(xi, yi, 'linear')(xe)
    ep = max(abs(ye-yp))
    el = max(abs(ye-yl))
    data1.append([n, ep, el])

df1 = pd.DataFrame(data1, columns=['n', 'err poly', 'err lin_spl'])
print(df1.to_string(index=False))

print('\n')
print(['interval (' + str(a) + ', ' + str(b)+ ')'])
data = []
for nn in range(4, 11):
    n = 2**nn
    xi = np.linspace(a, b, n+1)
    yi = np.cos(pi*xi)
    q = lambda xi: ((-5*pi*np.sin(11/10*pi))/31)*xi**2
    new_fi = np.cos(pi*xi) - q(xi)
    yl = sc.interp1d(xi, yi, 'linear')(xe) # linear
    ycc = sc.CubicSpline(xi, new_fi)(xe)
    ecc = max(abs(ye-ycc-q(xe)))
    el = max(abs(ye-yl))
    data_list = [n, el, ecc]
    data.append(data_list)


count = 1
while count < len(data):
    data[count].extend([log(data[count-1][1]/data[count][1])/log(2),
                        log(data[count-1][2]/data[count][2])/log(2)])
    count += 1
```

```
df = pd.DataFrame(data, columns=['n', 'linear s. error', 'clamped cubic s. error',
                                 'linear s. order of conv', ' clamped cubic s. order of conv'])
print(df.to_string(index=False))
```

```
['interval (0, 3.1)']
n    err poly   err lin_spl
sys:1: RankWarning: Polyfit may be poorly conditioned
sys:1: RankWarning: Polyfit may be poorly conditioned
 n     err poly  err lin_spl
 2 1.502234e+00     1.470800
 4 8.748848e-01     0.642336
 8 4.638694e-02     0.178750
16 9.962069e-07     0.045951
32 1.772720e-09     0.011544
64 3.430988e-09     0.002892
```

```
['interval (0, 3.1)']
   n  linear s. error  clamped cubic s. error  linear s. order of conv   clamped cubic s. order of conv
  16         0.045951           3.665451e-03                      NaN                             NaN
  32         0.011544           2.418708e-04                 1.992896                        3.921682
  64         0.002892           1.492963e-05                 1.996858                        4.017987
 128         0.000723           9.420607e-07                 2.000366                        3.986214
 256         0.000181           5.501978e-08                 1.999798                        4.097798
 512         0.000045           3.328209e-09                 1.999768                        4.047132
1024         0.000011           2.115619e-11                 2.009295                        7.297523
```

Note that due to the specifications of the

```
scipy.interpolate.CubicSpline(xi, yi, bc_type='clamped')
```

function in python, more specifically that it assumes that derivative of the function at the endpoints is 0 (which it doesn't have to be and isn't in this case), I had to resort to changing the actual function itself by subtracting a polynomial q to make the endpoints of this new function have derivative 0 and accordingly correct the calculation of the error. To clear up confusion, please refer to prof Christina as she was the one who came up with this method of fixing the issue. There is no other function in python to compute specifically the clamped cubic interpolation.

Here we see that the error of the linear spline is decreasing by a factor of 4 between two consecutive values of n as n increases in accordance with the expectation from the theory. Similarly, the error of the clamped cubic spline decreases by a factor of 16 as predicted by the theory. The linear spline order of convergence is at about 2 regardless of the iteration that we consider while the order of convergence of the clamped cubic hovers at about 4 (which again is in accordance with the theory) up until $n = 1024$ and indeed, the factor by which the error of the cubic spline decreases from $n = 512$ to $n = 1024$ jumps to about 160.

According to the table above, ratio between the error of the linear spline and the clamped cubic starts off at about 13 and as the value of n's doubles every iteration, this ratio increases by about a factor of 4 which is expected as the mentioned above, the factor by which the error for the linear one decreases is 4c while the clamped one, by 16, hence we get $16/4 = 4$. So as the number n doubles from iteration to iteration, the maximum error decreases while from both the theory and the experimental data in the table we see that the clamped cubic spline interpolates the data significantly better and would be more preferential however more costly as it requires significantly more operations to compute the result when interpolating a given data.

In the above result, we also see that the polynomial interpolation is significantly more accurate on the given interval. The rate at which the error of the polynomial interpolation decreases from one n to $2n$ increases exponentially going from about 1.7, to 19, to 46564 to later drop down as fast is it rose to 562 and lastly to 0.5 as the error reaches the higher end (in absolute values) of the negative single digit exponent of 10 while the error for the linear decreases by a factor of 4 as mentioned above.