HW3_AdamSanchez

September 18, 2020

1 Adam Sanchez

1.1 HW 3 Math 4650

```
In [2]: import matplotlib
        matplotlib.rcParams['text.usetex'] = True
        import matplotlib.pyplot as plt
        %matplotlib inline
        import numpy as np
        import sympy as sym
        from sympy import init_printing
        init_printing()
        from numpy.polynomial import polynomial as P
        import math
```

1.2 1)

a)

$$g(x) = -16 + 6x - \frac{12}{x}$$

$$g'(x) = 6 - \frac{12}{x^2}$$

This is not a contraction near $p = 2$ so there is no guarantee that we converge to p.

b)

$$g(x) = \frac{2}{3}x + \frac{1}{x^2}$$

$$g'(x) = \frac{2}{3} - \frac{2}{x^3}$$

Lets see if this is a contraction on $[1, 2]$:

$$\left|\frac{2}{3} - \frac{2}{x^3}\right| < \frac{2}{3}$$

for all $x \in [1, 2]$ So we have a contraction with linear convergence with rate $\frac{2}{3}$. Since $p \in [1, 2]$ we know we will converge to it.

c)

$$g(x) = \frac{12}{1+x}$$
$$g'(x) = -\frac{12}{1+x^2}$$

This is not a contraction near $p = 3$ so there is no guarantee that we converge to p

```
In [3]: def Newton(f,fprime,x0,maxIter = 100, fTol = 1e-8, relTol = 1e-8,Verbose=False):
          history_x  = np.zeros(maxIter)
          history_fx = np.zeros(maxIter)
          x    = np.asarray(x0,dtype=np.double).copy()
          fx   = f(x)
          history_x[0]   = x
          history_fx[0] = fx
          for n in range(1,maxIter):
            try:
              x -= fx / fprime(x)
            except ZeroDivisionError:
              return x, history_x, history_fx
            #print(x,fprime(x)) # for debugging
            if Verbose:
              print("Iteration {:4d}, x is {:+14.8e}, f(x) is {:+14.8e},  f'(x) is {:+14.8e}".f
            fx = f(x)
            history_x[n] = x
            history_fx[n] = fx
          return x, history_x, history_fx
```

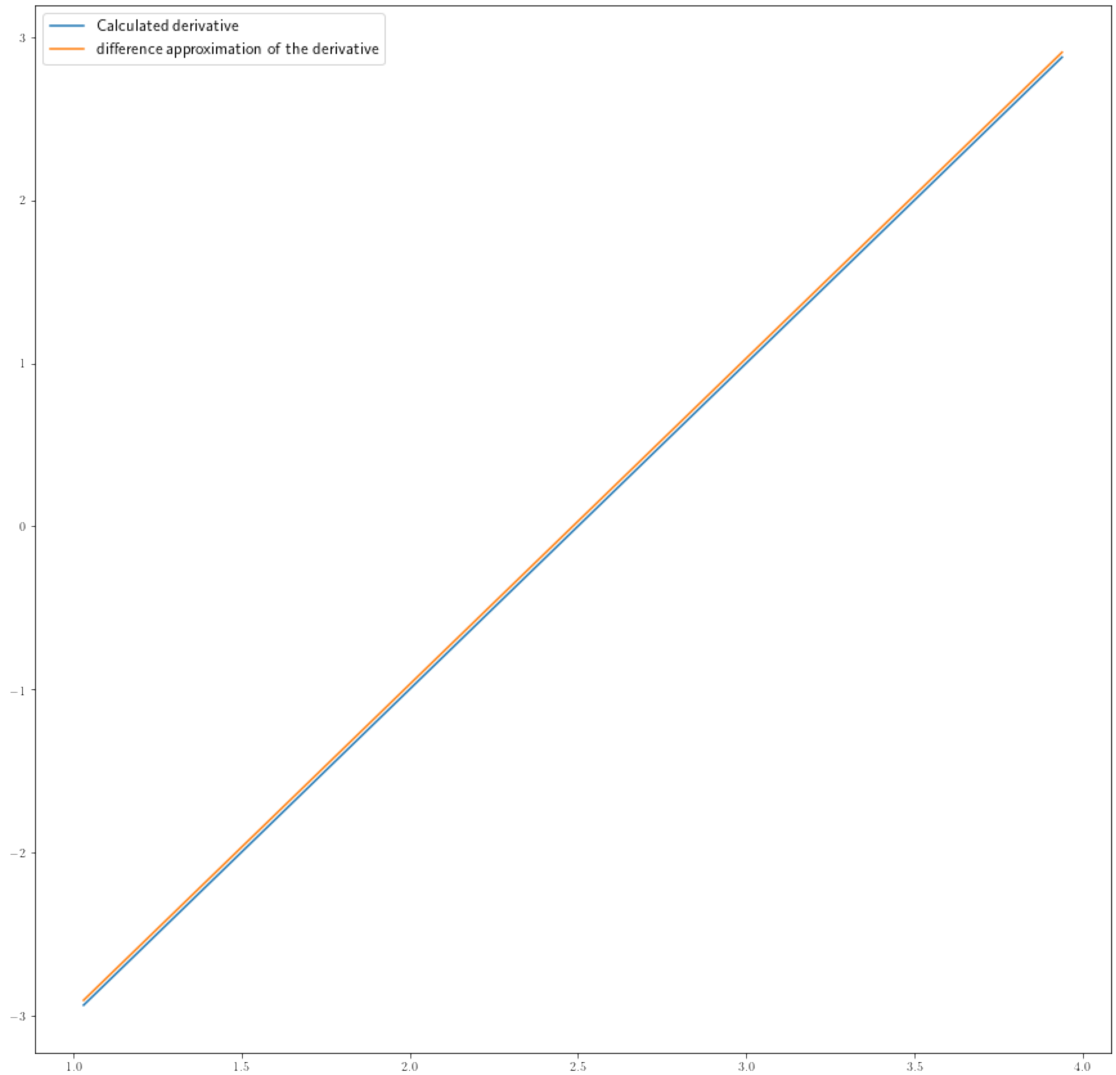
```
In [3]: x_vals = x_vals = np.linspace(1,4,100)
        f = lambda y: (y-3)*(y-2)
        f_prime = lambda x: 2*x - 5
        fprime = f_prime(x_vals)
        delty = np.diff(f(x_vals))
        deltx = np.diff(x_vals)
        approx_f_prime = delty/deltx
```

1.3 2)

a)

```
In [4]: plt.figure(figsize = (15,15))
        plt.plot(x_vals[1:98], fprime[1:98], x_vals[1:98], approx_f_prime[1:98])
        plt.legend(['Calculated derivative', 'difference approximation of the derivative'], fon
```

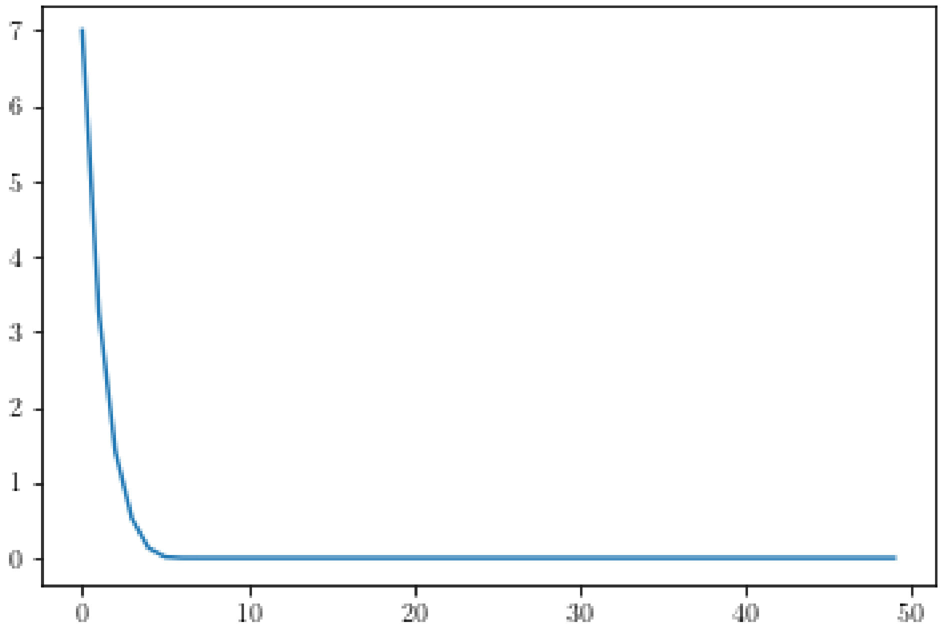
```
Out[4]: <matplotlib.legend.Legend at 0x10ea737f0>
```




```
In [44]: x0  = 10
         p, history_x, history_fx = Newton(f,f_prime,x0,maxIter=50)
         er = abs(history_x -3)
```

b) Yes the error does to converge to 0. We would expect it to decay at a quadric rate because 3 is a simple root of $f(x)$, which is what the plot below looks like.

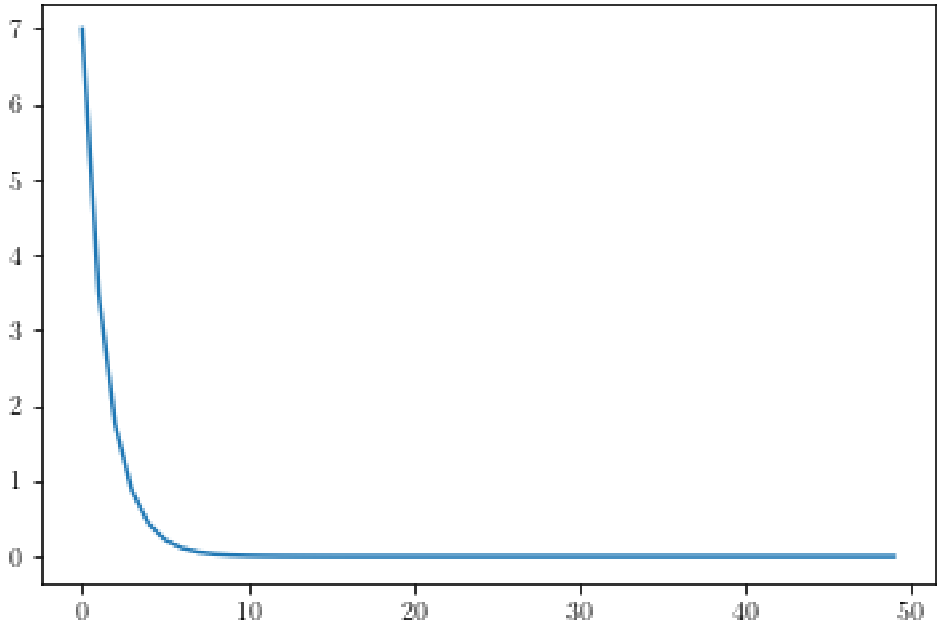
```
In [45]: plt.plot(er)

Out[45]: [<matplotlib.lines.Line2D at 0x10f9792b0>]
```




```
In [13]: fnew = lambda x: (3-x)**2
         fnew_prime = lambda x: 2*(x-3)
         p, history_x, history_fx = Newton(fnew,fnew_prime,x0,maxIter=50)
         er = abs(history_x -3)
```

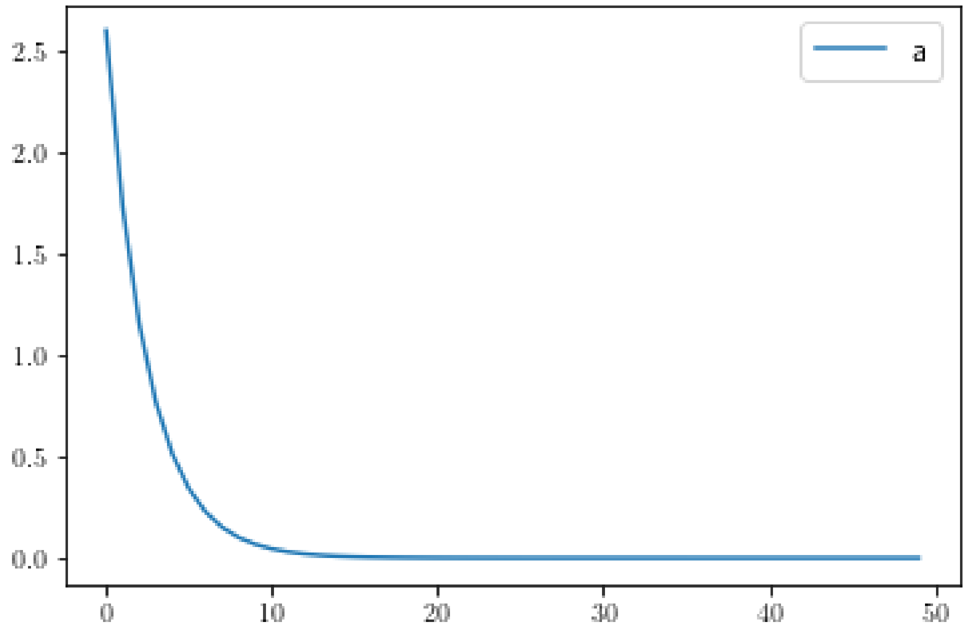
c) It does look like the error decays to 0 at a fairly fast rate. We woudn't expect quadric decay because 3 is not a simple root for our $f(x)$, but our plot does look like it is pretty fast.

```
In [47]: plt.plot(er)

Out[47]: [<matplotlib.lines.Line2D at 0x10f8d8a90>]
```



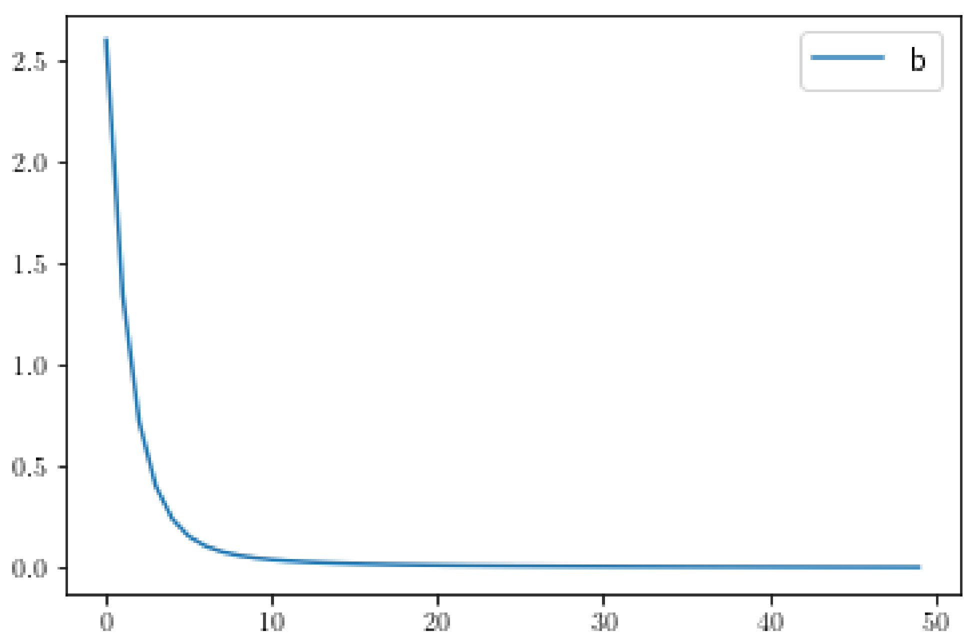
d) From the plots bellow it looks like both errors do decay to 0, but *i* looks like it decays a little faster

```
In [32]: fnew_primei = lambda x: 3*(x-3)
         p, history_x, history_fx = Newton(fnew,fnew_primei,x0,maxIter=50,fTol = 1e-15,relTol =
         er = abs(history_x -3)
         plt.plot(er)
         plt.legend('a', fontsize = 13)

Out[32]: <matplotlib.legend.Legend at 0x11a3e93c8>
```




```
In [31]: fnew_primeii = lambda x: 2*(x-3.1)
         p, history_x, history_fx = Newton(fnew,fnew_primeii,x0,maxIter=50)
         err = abs(history_x -3)
         plt.plot(err)
         plt.legend('b', fontsize = 13)
```

```
Out[31]: <matplotlib.legend.Legend at 0x11a2d9518>
```



```
In [34]: g = lambda x: (x-3)**2 + 1
         g_prime = lambda x: 2(x-3)
         p, history_x, history_fx = Newton(g,g_prime,x0,maxIter=50)
         p
```

```
        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-34-571c968760b1> in <module>()
          1 g = lambda x: (x-3)**2 + 1
          2 g_prime = lambda x: 2(x-3)
    ----> 3 p, history_x, history_fx = Newton(g,g_prime,x0,maxIter=50)
          4 p


        <ipython-input-3-e1378764275e> in Newton(f, fprime, x0, maxIter, fTol, relTol, Verbose)
          8   for n in range(1,maxIter):
          9     try:
    ---> 10       x -= fx / fprime(x)
         11     except ZeroDivisionError:
         12       return x, history_x, history_fx


        <ipython-input-34-571c968760b1> in <lambda>(x)
          1 g = lambda x: (x-3)**2 + 1
    ----> 2 g_prime = lambda x: 2(x-3)
          3 p, history_x, history_fx = Newton(g,g_prime,x0,maxIter=50)
          4 p


        TypeError: 'int' object is not callable
```

e) As expected, when we run Newton Method for this function we have an error because we dont have any real roots. Newtons Method is trying to find the root but as we progress the number never stabalizes

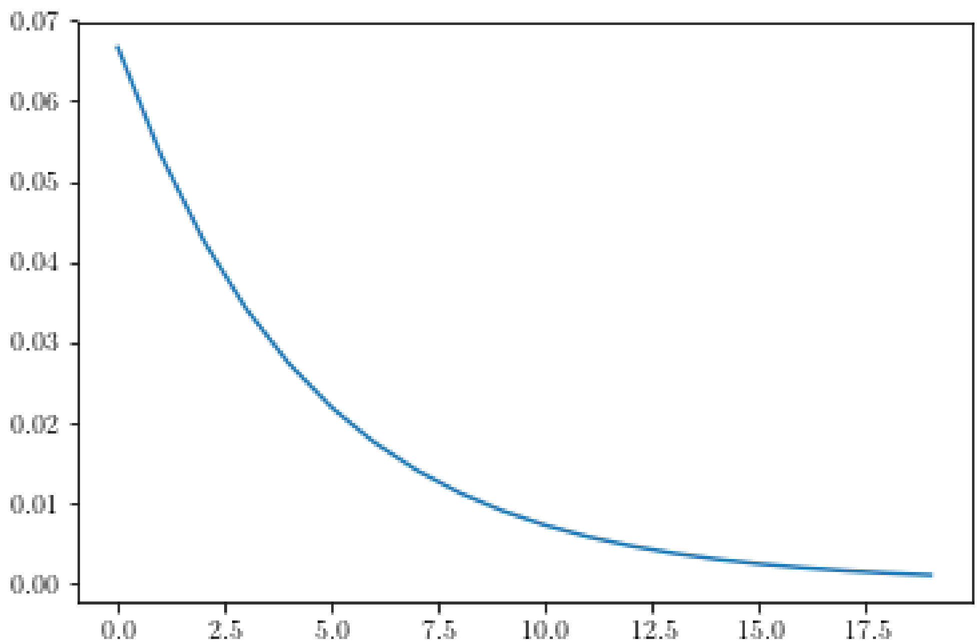
1.4 3

a) It does appear that the error decays to zero, but at a rate slower than quadric, which makes sense because $\frac{1}{3}$ is not a simple root

```
In [36]: ####3
         #a
```

```
f3 = lambda x: (x-(1/3))**5
f3_prime = lambda x: 5*(x-(1/3))**4
x0 = .4
p, history_x, history_fx = Newton(f3,f3_prime,x0,maxIter=20)
er = abs(history_x -(1/3))
plt.plot(er)
```

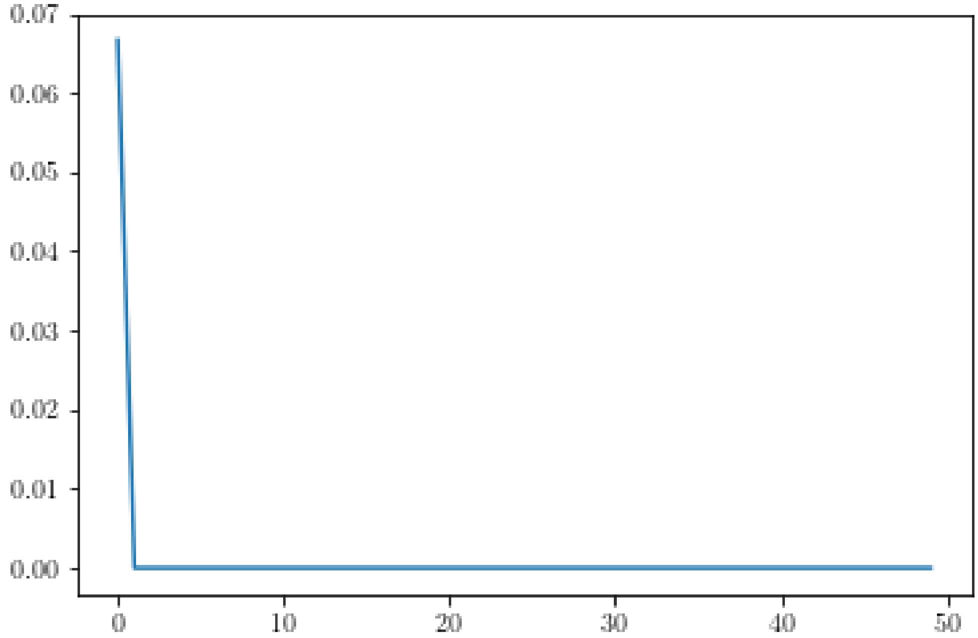
Out[36]: [<matplotlib.lines.Line2D at 0x115644898>]



b) It looks like the error decays to 0 increadbly fast (quadric) which is expected because we now know $\frac{1}{3}$ is a simple root of μ

In [38]:
```
#b
f3 = lambda x: (x-(1/3))**5
f3_prime = lambda x: 5*(x-(1/3))**4
mux = lambda x: (1/5)*(x-(1/3))
mux_prime = lambda x: (1/5)+(0*x)
x0 = .4
p, history_x, history_fx = Newton(mux,mux_prime,x0,maxIter=50)
er = abs(history_x -(1/3))
plt.plot(er)
```

Out[38]: [<matplotlib.lines.Line2D at 0x11a4f5a90>]



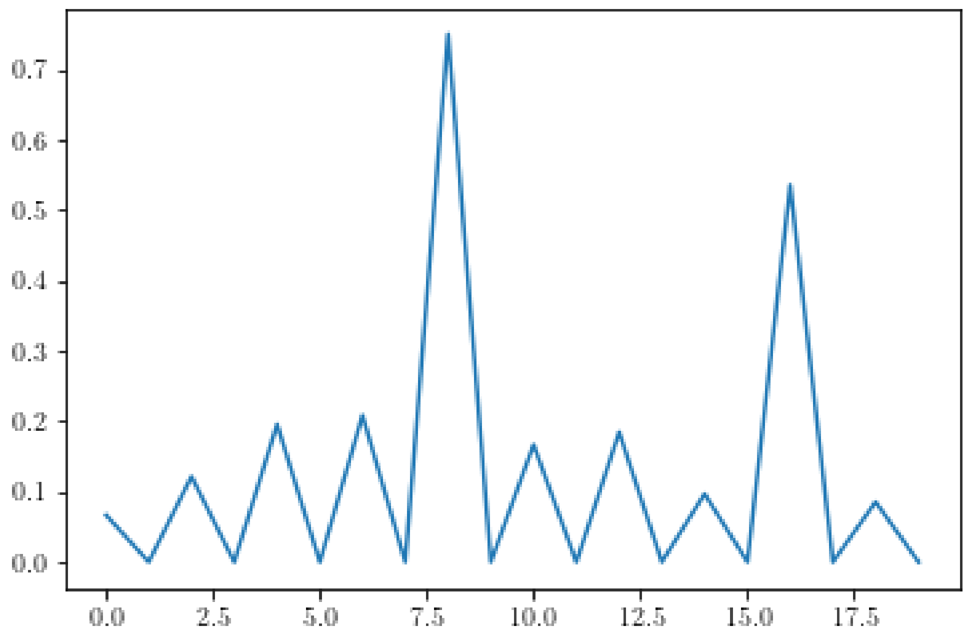
c)It looks like the error is bouncing around 0. I think this is because we first changed the function to a polynmial so we may be running into so numerical issues during Newtons Method.

```
In [8]: #c
        def NewModifiedNewton(f,fprime,fdoubleprime,x0,maxIter = 100,fTol = 1e-8, relTol = 1e-8
            history_x  = np.zeros(maxIter)
            history_fx = np.zeros(maxIter)
            x   = np.asarray(x0,dtype=np.double).copy()
            fx  = f(x)
            history_x[0]  = x
            history_fx[0] = fx
            for n in range(1,maxIter):
                try:
                    x -= (fx*fprime(x)) / ((fprime(x)**2)-(fx*fdoubleprime(x)))
                except ZeroDivisionError:
                    return x, history_x, history_fx
                if Verbose:
                    print("Iteration {:4d}, x is {:+14.8e}, f(x) is {:+14.8e},  f'(x) is {:+14
                fx = f(x)
                history_x[n] = x
                history_fx[n] = fx
            return x, history_x, history_fx

In [9]: roots = [(1/3), (1/3), (1/3), (1/3), (1/3)]
        x0=.4
        f = np.poly1d(np.poly(roots))
```

```
fprime = np.poly1d(np.polyder(np.poly(roots)))
fdoubleprime = np.poly1d(np.polyder(np.polyder(np.poly(roots))))
```

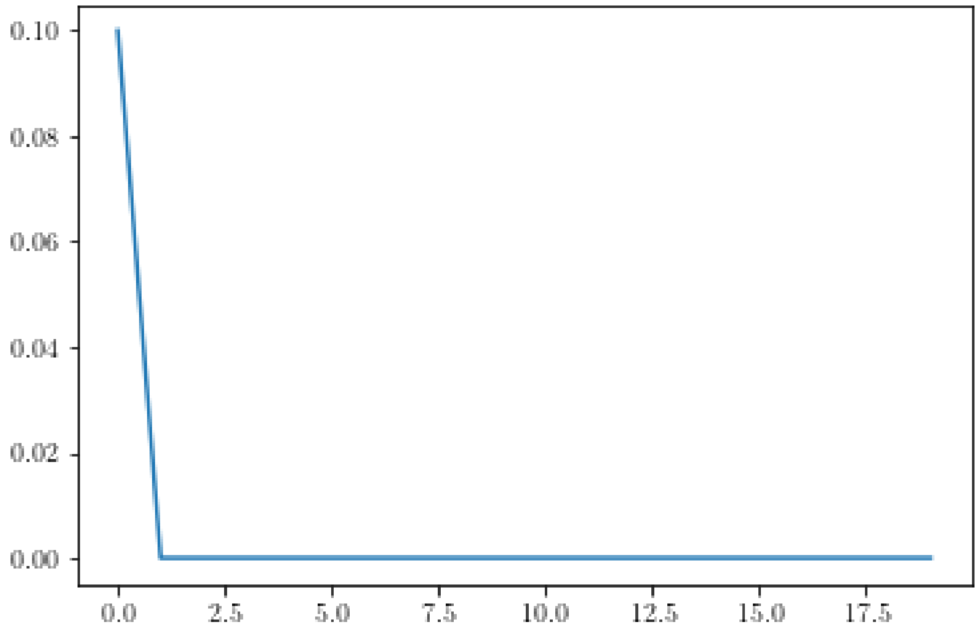
In [11]:
```
p,hist_x,hist_fx = NewModifiedNewton(f,fprime,fdoubleprime,x0)
er = abs(hist_x -(1/3))
plt.plot(er[:20])
```

Out[11]: [<matplotlib.lines.Line2D at 0x10bde1160>]



In [42]:
```
#d
#doing part b
f4 = lambda x: (x-(1/2))**5
f3_prime = lambda x: 5*(x-(1/2))**4
mux = lambda x: (1/5)*(x-(1/2))
mux_prime = lambda x: (1/5)+(0*x)
x0 = .4
p, history_x, history_fx = Newton(mux,mux_prime,x0,maxIter=50)
er = abs(history_x -(1/2))
plt.plot(er[:20])
```

Out[42]: [<matplotlib.lines.Line2D at 0x11a76d240>]




```
In [40]: #doing part c
         roots = [(1/2), (1/2), (1/2), (1/2), (1/2)]
         x0=.4
         f = np.poly1d(np.poly(roots))
         fprime = np.poly1d(np.polyder(np.poly(roots)))
         fdoubleprime = np.poly1d(np.polyder(np.polyder(np.poly(roots))))

         p,hist_x,hist_fx = NewModifiedNewton(f,fprime,fdoubleprime,x0, maxIter=20)
         er = abs(hist_x -(1/2))
         er
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:11: RuntimeWarning: invalid value
  # This is added back by InteractiveShellApp.init_path()
```

```
Out[40]: array([1.00000000e-01, 9.43134459e-14,            nan,            nan,
                           nan,            nan,            nan,            nan,
                           nan,            nan,            nan,            nan,
                           nan,            nan,            nan,            nan,
                           nan,            nan,            nan,            nan])
```

1.5 4

Note $F(d)$ is maximized when $F'(d) = 0 = f(d)$. So we know have a root finding problem! Using $f(d)$ as our main function and $f'(d) = F''(d)$ as our derivative in Newtons Method we get: $p = 2.15329236$ dogs are the optimal number of dogs to maximize happiness

```
In [63]: f = lambda d: 2*d - .5*math.exp(d)
         fprime = lambda d: 2-.5*math.exp(d)

         x0 = 2
         p, history_x, history_fx = Newton(f,fprime,x0,maxIter=50, Verbose = True)
```

```
Iteration    1, x is +2.18026963e+00, f(x) is +3.05471951e-01,  f'(x) is -2.42434592e+00
Iteration    2, x is +2.15395051e+00, f(x) is -6.38066536e-02,  f'(x) is -2.30942003e+00
Iteration    3, x is +2.15329277e+00, f(x) is -1.51900815e-03,  f'(x) is -2.30658647e+00
Iteration    4, x is +2.15329236e+00, f(x) is -9.31982949e-07,  f'(x) is -2.30658473e+00
Iteration    5, x is +2.15329236e+00, f(x) is -3.51718654e-13,  f'(x) is -2.30658473e+00
Iteration    6, x is +2.15329236e+00, f(x) is -8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration    7, x is +2.15329236e+00, f(x) is +8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration    8, x is +2.15329236e+00, f(x) is -8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration    9, x is +2.15329236e+00, f(x) is +8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   10, x is +2.15329236e+00, f(x) is -8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   11, x is +2.15329236e+00, f(x) is +8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   12, x is +2.15329236e+00, f(x) is -8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   13, x is +2.15329236e+00, f(x) is +8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   14, x is +2.15329236e+00, f(x) is -8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   15, x is +2.15329236e+00, f(x) is +8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   16, x is +2.15329236e+00, f(x) is -8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   17, x is +2.15329236e+00, f(x) is +8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   18, x is +2.15329236e+00, f(x) is -8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   19, x is +2.15329236e+00, f(x) is +8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   20, x is +2.15329236e+00, f(x) is -8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   21, x is +2.15329236e+00, f(x) is +8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   22, x is +2.15329236e+00, f(x) is -8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   23, x is +2.15329236e+00, f(x) is +8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   24, x is +2.15329236e+00, f(x) is -8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   25, x is +2.15329236e+00, f(x) is +8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   26, x is +2.15329236e+00, f(x) is -8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   27, x is +2.15329236e+00, f(x) is +8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   28, x is +2.15329236e+00, f(x) is -8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   29, x is +2.15329236e+00, f(x) is +8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   30, x is +2.15329236e+00, f(x) is -8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   31, x is +2.15329236e+00, f(x) is +8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   32, x is +2.15329236e+00, f(x) is -8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   33, x is +2.15329236e+00, f(x) is +8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   34, x is +2.15329236e+00, f(x) is -8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   35, x is +2.15329236e+00, f(x) is +8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   36, x is +2.15329236e+00, f(x) is -8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   37, x is +2.15329236e+00, f(x) is +8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   38, x is +2.15329236e+00, f(x) is -8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   39, x is +2.15329236e+00, f(x) is +8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   40, x is +2.15329236e+00, f(x) is -8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   41, x is +2.15329236e+00, f(x) is +8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   42, x is +2.15329236e+00, f(x) is -8.88178420e-16,  f'(x) is -2.30658473e+00
```

```
Iteration   43, x is +2.15329236e+00, f(x) is +8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   44, x is +2.15329236e+00, f(x) is -8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   45, x is +2.15329236e+00, f(x) is +8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   46, x is +2.15329236e+00, f(x) is -8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   47, x is +2.15329236e+00, f(x) is +8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   48, x is +2.15329236e+00, f(x) is -8.88178420e-16,  f'(x) is -2.30658473e+00
Iteration   49, x is +2.15329236e+00, f(x) is +8.88178420e-16,  f'(x) is -2.30658473e+00
```