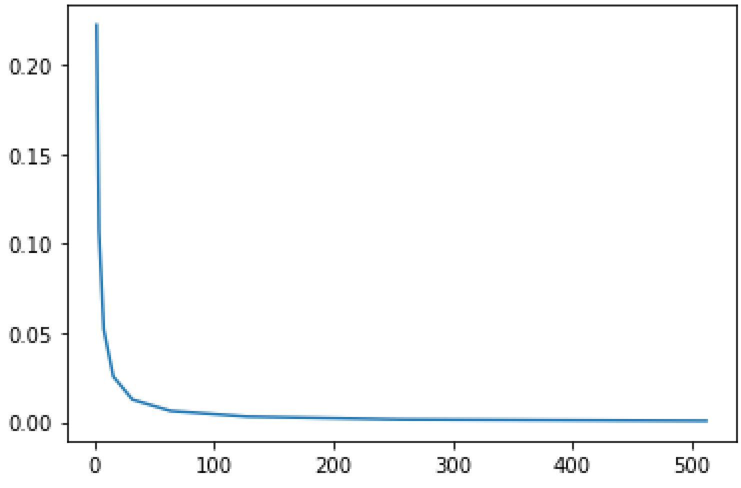
In [35]:

```
import numpy as np
import matplotlib.pyplot as plt

#使う関数の定義.
def derivative(f, x, h):
  return (f(x+h) - f(x)) / h
def func(x):
  return np.sin(x)

#一回微分の値を数値的に求めよ.
h = 0.000000001
x = 0.3*np.pi
t = derivative(func, x, h)
print(t)

#y=h,z=誤差でグラフを作成.
y = []
z = []

for num in range(1,10):
    num = 2**num
    y.append(num)
    num2 = 1/num
    t = derivative(func, x, num2)
    u = 0.587785252292473 - t
    z.append(u)

print(y)
print(z)
plt.plot(y,z)
plt.show()
```

```
0.5877851538826917
[2, 4, 8, 16, 32, 64, 128, 256, 512]
[0.22226230564168425, 0.10670517445836725, 0.05202725381174089, 0.025656150355766294, 0.012735525353675947, 0.006344233431004324, 0.003166185803134547, 0.0015816041219910426, 0.0007904291111447 97]
```



In [40]:

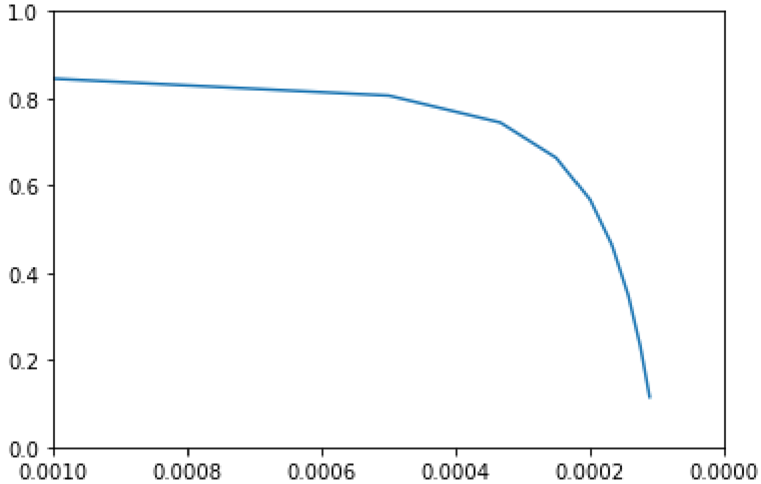
```
from sympy import *

#値がπになるかプログラムで確認.
x = symbols('x')
f = 4/(1+x**2)
integrate(f,(x,0,1))
```

Out[40]: π

In [25]:

```python
import numpy as np
import matplotlib.pyplot as plt

#使う関数の定義
def f(x):
    return 4/(1+x**2)

def sigma(func, frm, to):
    result = 0;
    for i in range(frm, to+1):
        result += func(i/10000)
    return result

#y=h,z=誤差でグラフ作成
y = []
z = []

for num in range(1,10000,1000):
    h = 1/num
    y.append(h)
    t = h*sigma(f,1,num)
    u = t-np.pi
    z.append(u)

print(y)
print(z)
plt.plot(y,z)
plt.xlim([0.001,0])
plt.ylim([0,1])
plt.show()
```

```
[1.0, 0.000999000999000999, 0.0004997501249375312, 0.0003332222592469177, 0.00024993
751562109475, 0.0001999600079984003, 0.00016663889351774705, 0.00014283673760891302,
0.00012498437695288088, 0.00011109876680368848]
[0.8584073064102076, 0.845107255210646, 0.806229167737933, 0.7443707410824203, 0.663
3129301754144, 0.5674063631456274, 0.4610054957834291, 0.3480609884543018, 0.2318976
5609060276, 0.11514756977529172]
```



In [7]:

```python
import numpy as np
import matplotlib.pyplot as plt

#解きたい方程式
def func_f(x):
    return x**2-4*x

kon = (4+np.sqrt(4**2-4*1*0))/2*1
kon2 = (4-np.sqrt(4**2-4*1*0))/2*1
```

```python
print("kon:{:3f}".format(kon))
print("kon2:{:3f}".format(kon2))

#二分法
def bisection(func_f, x_min, x_max, error=1e-10, max_loop=100):

    num_calc = 0
    print("{:3d}:  {:.15f} <= x <= {:.15f}".format(num_calc, x_min, x_max))

    while(True):
        x_mid = (x_max +x_min)/2.0

        if (0.0 < func_f(x_mid)*func_f(x_max)):
            x_max = x_mid
        else:
            x_min = x_mid

        num_calc += 1
        print("{:3d}:  {:.15f} <= x <= {:.15f}".format(num_calc, x_min, x_max))

        if((x_max-x_min <= error) or max_loop <= num_calc):
            break

    print("x = {:.15f}".format(x_mid))

    return x_mid


#可視化
def visualization(func_f, x_min, x_max, x_solved):

    exact_x = np.arange(x_min,x_max, (x_max-x_min)/500.0)
    exact_y = func_f(exact_x)

    plt.plot(exact_x,exact_y)
    plt.scatter(x_solved,0.0)
    plt.axhline(0, color='#000000')
    plt.text(x_solved,0.0, "$x$ = {:.9f}".format(x_solved), va='bottom', color='#000
    plt.show()

#メイン実行部
if (__name__ == '__main__'):
    solution = bisection(func_f, -1.0, 100)

    visualization(func_f, solution-1.0, solution+1.0, solution)
```

```
kon:4.000000
kon2:0.000000
  0:   -1.000000000000000 <= x <= 100.000000000000000
  1:   -1.000000000000000 <= x <= 49.500000000000000
  2:   -1.000000000000000 <= x <= 24.250000000000000
  3:   -1.000000000000000 <= x <= 11.625000000000000
  4:   -1.000000000000000 <= x <= 5.312500000000000
  5:   2.156250000000000 <= x <= 5.312500000000000
  6:   3.734375000000000 <= x <= 5.312500000000000
  7:   3.734375000000000 <= x <= 4.523437500000000
  8:   3.734375000000000 <= x <= 4.128906250000000
  9:   3.931640625000000 <= x <= 4.128906250000000
 10:   3.931640625000000 <= x <= 4.030273437500000
 11:   3.980957031250000 <= x <= 4.030273437500000
 12:   3.980957031250000 <= x <= 4.005615234375000
 13:   3.993286132812500 <= x <= 4.005615234375000
 14:   3.999450683593750 <= x <= 4.005615234375000
 15:   3.999450683593750 <= x <= 4.002532958984375
 16:   3.999450683593750 <= x <= 4.000991821289062
 17:   3.999450683593750 <= x <= 4.000221252441406
 18:   3.999835968017578 <= x <= 4.000221252441406
 19:   3.999835968017578 <= x <= 4.000028610229492
 20:   3.999932289123535 <= x <= 4.000028610229492
 21:   3.999980449676514 <= x <= 4.000028610229492
 22:   3.999980449676514 <= x <= 4.000004529953003
 23:   3.999992489814758 <= x <= 4.000004529953003
 24:   3.999998509883881 <= x <= 4.000004529953003
 25:   3.999998509883881 <= x <= 4.000001519918442
 26:   3.999998509883881 <= x <= 4.000000014901161
 27:   3.999999262392521 <= x <= 4.000000014901161
 28:   3.999999638646841 <= x <= 4.000000014901161
 29:   3.999999826774001 <= x <= 4.000000014901161
 30:   3.999999920837581 <= x <= 4.000000014901161
 31:   3.999999967869371 <= x <= 4.000000014901161
 32:   3.999999991385266 <= x <= 4.000000014901161
 33:   3.999999991385266 <= x <= 4.000000003143214
 34:   3.999999997264240 <= x <= 4.000000003143214
 35:   3.999999997264240 <= x <= 4.000000000203727
 36:   3.999999998733983 <= x <= 4.000000000203727
 37:   3.999999999468855 <= x <= 4.000000000203727
 38:   3.999999999836291 <= x <= 4.000000000203727
 39:   3.999999999836291 <= x <= 4.000000000020009
 40:   3.999999999928150 <= x <= 4.000000000020009
x = 3.999999999928150
```