Suites récurrentes, listes

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

# Discretisation pour l'affichage de graphes de fonctions
m = 300
```

Suites récurrentes

In [2]:
```
def u(f, alpha, n):
    ans = alpha
    for k in range(n):
        ans = f(ans)
    return ans
```

In [3]:
```
def f0(x):
    return np.sqrt(1 + x)
```

In [4]:
```
u(f0, 0, 2)
```

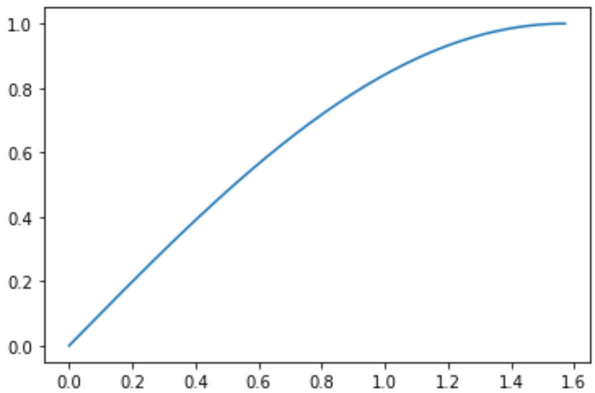
Out[4]: 1.4142135623730951

In [5]:
```
def listes_graphe(f, x_min, x_max, m):
    x_graphe = [x_min + (k / m) * (x_max - x_min) for k in range(m + 1)]
    y_graphe = [f(x) for x in x_graphe]
    return x_graphe, y_graphe
```

In [6]:
```
x, y = listes_graphe(np.sin, 0, np.pi / 2, m)
```

In [7]:
```
plt.plot(x, y)
```

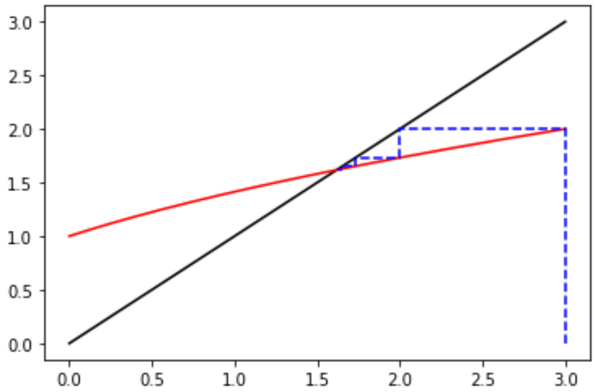
Out[7]: [<matplotlib.lines.Line2D at 0x7f78991bcb90>]



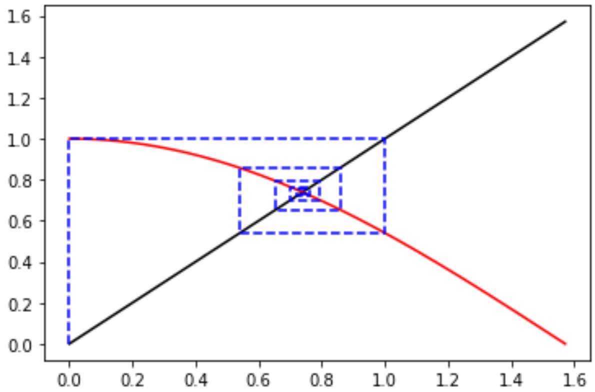
In [8]:
```
def liste_escalier(f, alpha, n):
    u = alpha
    x_escalier = [u]
    y_escalier = [0.0]
    for k in range(n):
        x_escalier.append(u)
        u = f(u)
        x_escalier.append(u)
        y_escalier.append(u)
        y_escalier.append(u)
    return x_escalier, y_escalier
```

In [9]:
```
def escalier(f, alpha, n, x_min, x_max, m):
    x_bissectrice = [x_min, x_max]
    y_bissectrice = [x_min, x_max]
    plt.plot(x_bissectrice, y_bissectrice, color='black')
    x_f, y_f = listes_graphe(f, x_min, x_max, m)
    plt.plot(x_f, y_f, color='red')
    x_escalier, y_escalier = liste_escalier(f, alpha, n)
    plt.plot(x_escalier, y_escalier, '--', color='blue')
```

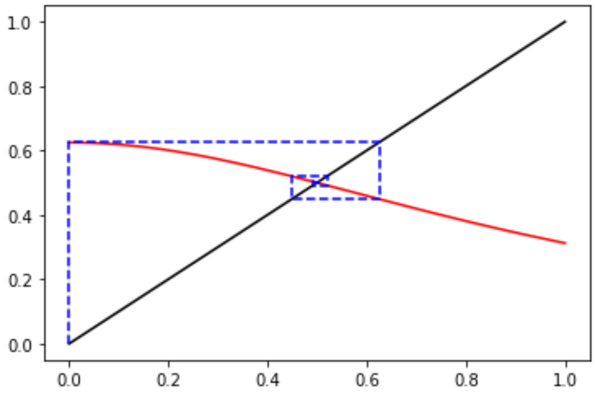
In [10]:
```
def f0(x):
    return np.sqrt(1 + x)

escalier(f0, 3.0, 10, 0.0, 3.0, m)
```



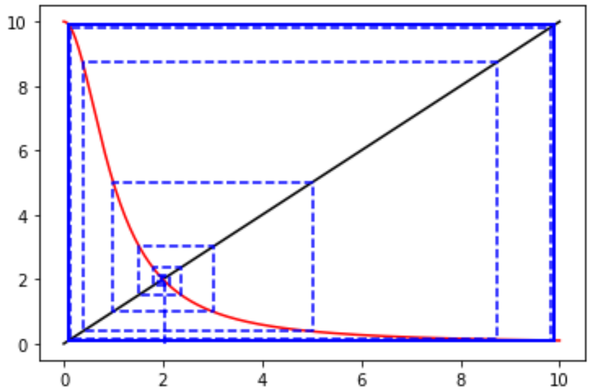
In [11]:
```
escalier(np.cos, 0.0, 10, 0.0, np.pi / 2, m)
```



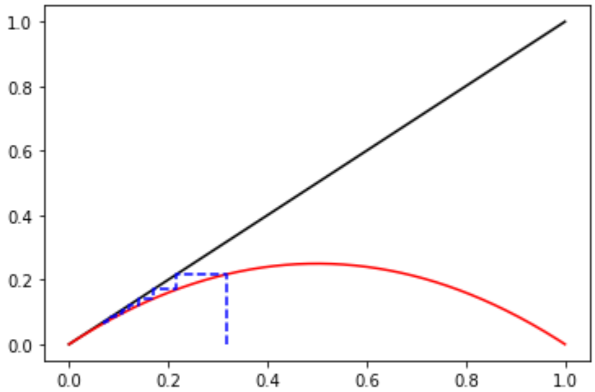
In [12]:
```
a = 1 / 2

def f(x):
    return a * (1 + a**2) / (1 + x**2)

escalier(f, 0.0, 10, 0.0, 1, m)
```



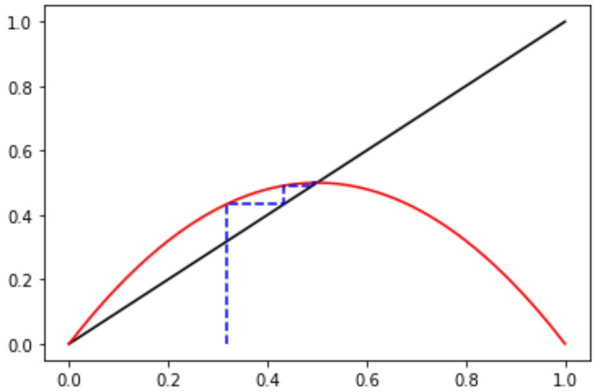
In [13]:
```
a = 2
escalier(lambda x: a * (1 + a**2) / (1 + x**2), 1.01 * a, 100, 0.0, 10, m)
```



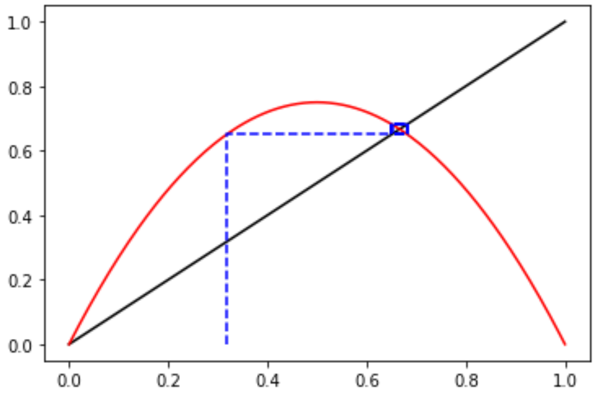
In [14]:
```
a = 1
escalier(lambda x: a * x * (1 - x), 1 / np.pi, 10, 0.0, 1.0, m)
```



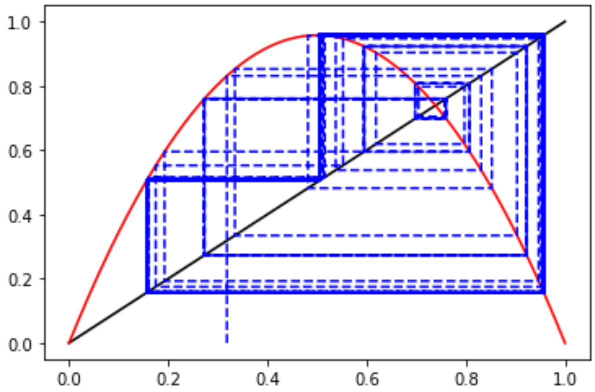
In [15]:
```
a = 2
escalier(lambda x: a * x * (1 - x), 1 / np.pi, 10, 0.0, 1.0, m)
```



In [16]:
```
a = 3
escalier(lambda x: a * x * (1 - x), 1 / np.pi, 10, 0.0, 1.0, m)
```

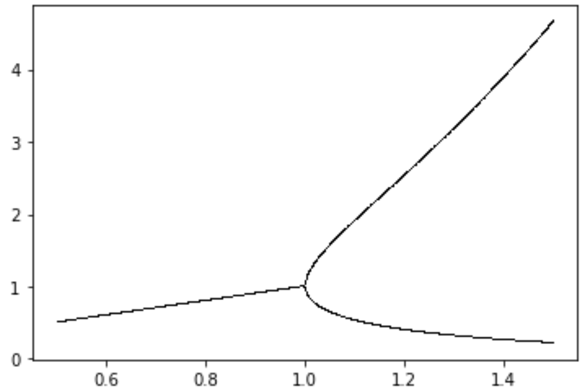


In [17]:
```
def f(x):
    return 3.83 * x * (1 - x)

escalier(f, 1 / np.pi, 1000, 0.0, 1.0, m)
```

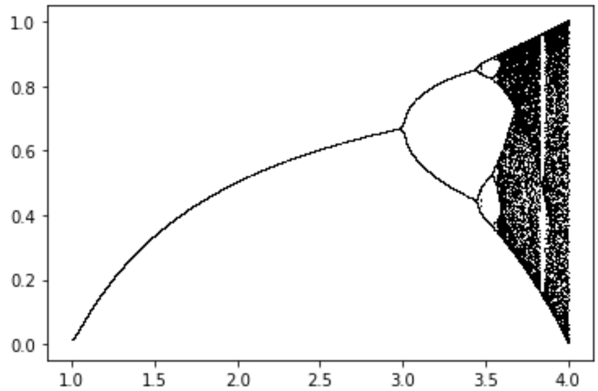


In [18]:
```
def f(a, x):
    return a * (1 + a**2) / (1 + x**2)

def bifurcation(f, alpha, a_min, a_max, na, n_min, n_max):
    a_graphe = [a_min + (k / na) * (a_max - a_min) for k in range(na + 1)]
    u_graphe = [alpha for k in range(na + 1)]
    for i in range(n_min):
        for k in range(na + 1):
            u_graphe[k] = f(a_graphe[k], u_graphe[k])
    for i in range(n_min, n_max):
        plt.plot(a_graphe, u_graphe, ',', color='black')
        for k in range(na + 1):
            u_graphe[k] = f(a_graphe[k], u_graphe[k])
```

In [19]:
```
bifurcation(f, 1 / np.pi, 1/2, 1.5, 1000, 1000, 1010)
```



In [20]:
```
def f(a, x):
    return a * x * (1 - x)

bifurcation(f, 1 / np.pi, 1, 4, 10000, 100, 110)
```



Bloc de somme maximale

In [21]:
```
def somme_max(t):
    n = len(t)
    ans = 0
    for i in range(n + 1):
        for j in range(i, n + 1):
            sum = 0
            for k in range(i, j):
                sum = sum + t[k]
            if sum > ans:
                ans = sum
    return ans
```

On montre que la complexite temporelle de cette fonction est en O(n^3).

In [22]:
```
somme_max([5, 3, -3, 2])
```
Out[22]: 8

In [23]:
```
somme_max([-2, -3])
```
Out[23]: 0

In [24]:
```
somme_max([1, -2, 5, -1, 7, -1])
```
Out[24]: 11

In [25]:
```
somme_max([4, -1, 2, 3, -1, 2])
```
Out[25]: 9

Il y avait une erreur d'enonce. Il est corrige dans le nouveau document disponible sur le site.

In [26]:
```
def somme_max_lin(t):
    n = len(t)
    v = [0 for k in range(n + 1)]
    for j in range(1, n + 1):
        value = v[j - 1] + t[j - 1]
        if value >= 0:
            v[j] = value
    ans = 0
    for k in range(n + 1):
        if v[k] > ans:
            ans = v[k]
    return ans
```

In [27]:
```
somme_max_lin([5, 3, -3, 2])
```
Out[27]: 8

In [28]:
```
somme_max_lin([-2, -3])
```
Out[28]: 0

In [29]:
```
somme_max_lin([1, -2, 5, -1, 7, -1])
```
Out[29]: 11

In [30]:
```
somme_max_lin([4, -1, 2, 3, -1, 2])
```
Out[30]: 9

In [31]:
```
def somme_max_lin(t):
    n = len(t)
    v = 0
    ans = 0
    for j in range(n):
        new_v = v + t[j]
        if new_v >= 0:
            v = new_v
        else:
            v = 0
        if v > ans:
            ans = v
    return ans
```

In [32]:
```
somme_max_lin([5, 3, -3, 2])
```

Out[32]: 8

In [33]:
```
somme_max_lin([-2, -3])
```

Out[33]: 0

In [34]:
```
somme_max_lin([1, -2, 5, -1, 7, -1])
```

Out[34]: 11

In [35]:
```
somme_max_lin([4, -1, 2, 3, -1, 2])
```

Out[35]: 9