Corrigé

Étude de la suite logistique

Question 1. Les graphes de la figure 1 ont été obtenus à l'aide de la fonction :

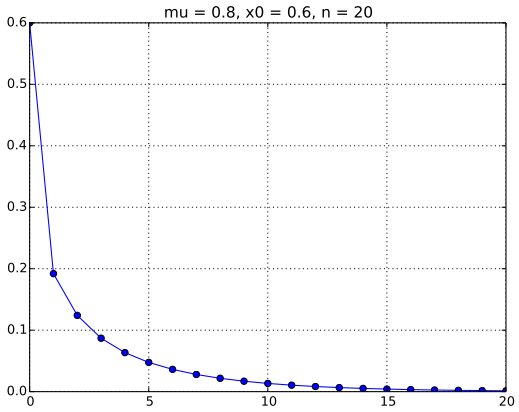
```
def logistique2(mu, x0, n):
    def f(x):
        return mu * x * (1 - x)
    plt.figure()
    plt.title('mu = {}, x0 = {}, n = {}'.format(mu, x0, n))
    Xn = [x0]
    for k in range(n):
        Xn.append(f(Xn[-1]))
    plt.plot(Xn, 'o-')
    plt.grid()
    plt.show()
```

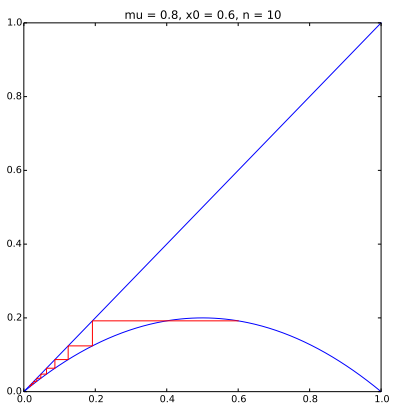
Ceux de la figure 2 à l'aide de la fonction :

```
def logistique1(mu, x0, n):
    def f(x):
        return mu * x * (1 - x)
    plt.figure(figsize=(8, 8))
    plt.title('mu = {}, x0 = {}, n = {}'.format(mu, x0, n))
    X = np.linspace(0, 1, 256)
    Y = [f(x) for x in X]
    plt.plot(X, Y, color='blue')
    plt.plot([0, 1], [0, 1], color='blue')
    x, y = x0, f(x0)
    Xn, Yn = [x], [y]
    for k in range(n-1):
        x, y = y, f(y)
        Xn.append(x)
        Xn.append(x)
        Yn.append(x)
        Yn.append(y)
    plt.plot(Xn, Yn, color='red')
    plt.show()
```

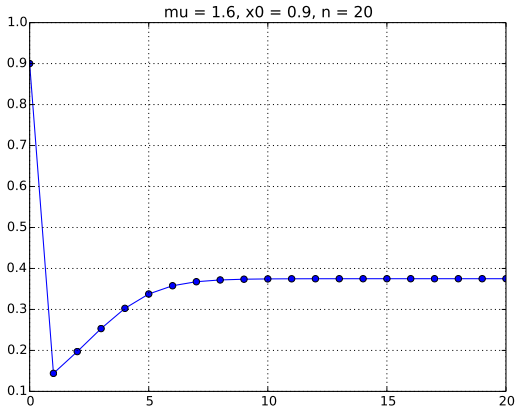
Question 2.

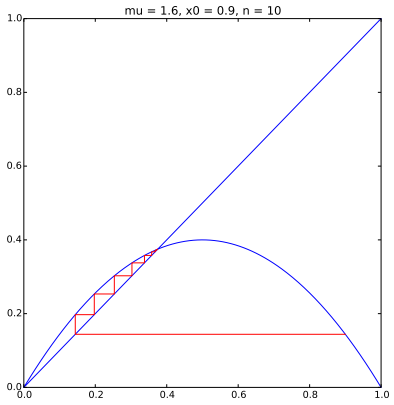
Lorsque $\mu \in [0,1]$ la suite $(x_n)_{n\in\mathbb{N}}$ décroit et converge vers 0 :



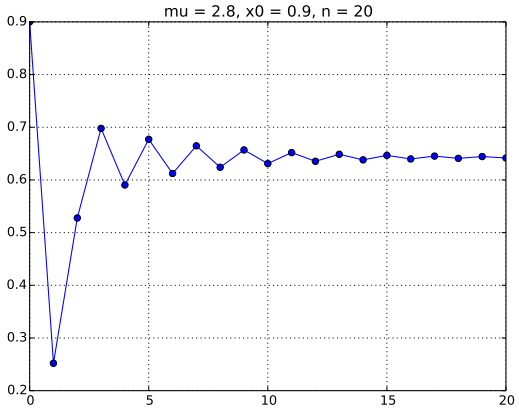


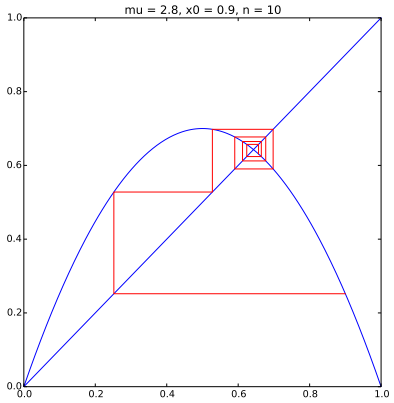
Lorsque $\mu \in [1,2]$ la suite $(x_n)_{n\in\mathbb{N}}$ croit à partir d'un certain rang et converge :



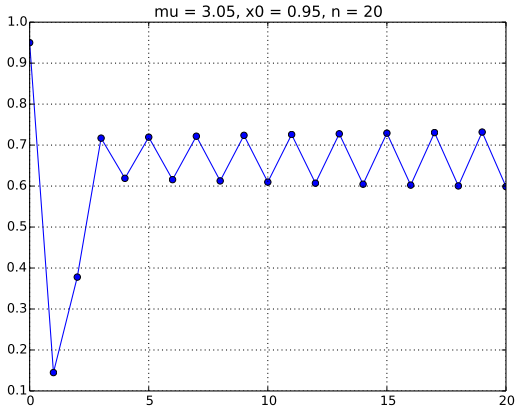


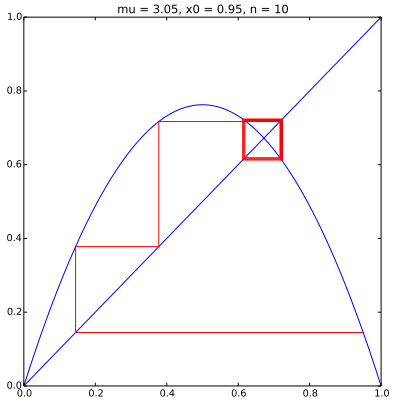
Lorsque $\mu \in [2,3]$ les suites $(x_{2n})_{n\in\mathbb{N}}$ et $(x_{2n+1})_{n\in\mathbb{N}}$ sont adjacentes :



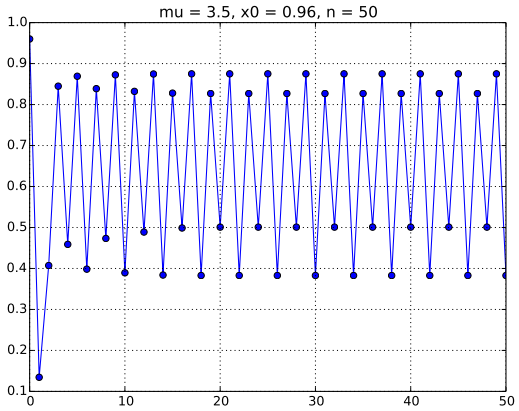


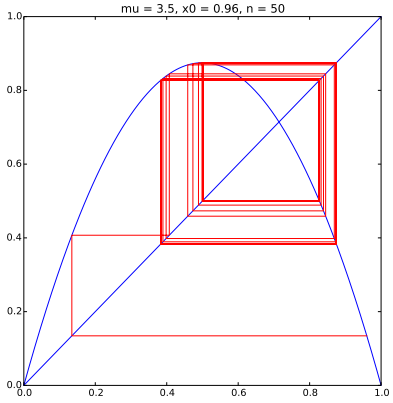
Lorsque $\mu = 3{,}05$ les suites $(x_{2n})_{n\in\mathbb{N}}$ et $(x_{2n+1})_{n\in\mathbb{N}}$ convergent vers deux limites distinctes :



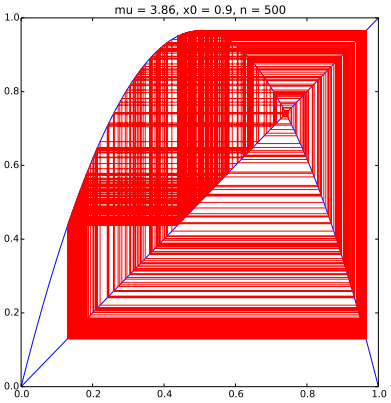


Lorsque $\mu = 3{,}5$ on peut observer une oscillation entre quatre valeurs distinctes :





Lorsque $\mu = 3{,}86$ le comportement est chaotique :



Question 3. Le diagramme des bifurcations a été obtenu à l'aide du script :

```
def cycle(mu):
    def f(x):
        return mu * x * (1 - x)
    x = 0.9
    for n in range(100):
        x = f(x)
    lst = []
    for n in range(200):
        y = round(x, 3)
        if y not in lst:
            lst.append(y)
        x = f(x)
    return lst

def bifurcation(a, b, p=.002):
    plt.figure()
    plt.title("Diagramme des bifurcations")
    X, Y = [], []
    mu = a
    while mu < b:
        lst = cycle(mu)
        for v in lst:
            X.append(mu)
            Y.append(v)
        mu += p
    plt.plot(X, Y, marker=',', linestyle='')
    plt.show()
```

La fonction `cycle` retourne l'ensemble des valeurs distinctes à 10^{-3} près de $x_{101},\dots,x_{200}$ en fonction de μ.

Question 4. On commence par définir :

```
def lyapunov(mu, x0, n):
    def f(x):
        return mu * x * (1 - x)
    def fprime(x):
        return mu * (1 - 2 * x)
    x = x0
    l = fprime(x0)
    for k in range(n-1):
        x = f(x)
        l += np.log(abs(fprime(x)))
    return l / n
```

Le diagramme de Lyapunov s'(obtient ensuite à l'aide du script :

```
plt.figure()
plt.title("Variations de l'exposant de Lyapunov")
plt.xlim(3, 4)
plt.ylim(-2, 1)
plt.plot([3, 4], [0, 0])
X = np.linspace(3, 4, 512)
Y = [lyapunov(mu, .9, 100) for mu in X]
plt.plot(X, Y, 'r')
plt.show()
```

Les valeurs négatives correspondent à des positions stables : de faibles erreurs sur la valeur de x_0 n'entraine pas de modification du comportement asymptotique de la suite. À l'inverse, les valeurs positives correspondent à des situations chaotiques : une faible erreur sur x_0 a des conséquences importantes sur le comportement futur de la suite. On peut constater qu'au delà de 3,6 le comportement chaotique est de plus en plus marqué, à l'exception de quelques îlots de stabilité (par exemple aux alentours de $\mu = 3{,}82$; c'est aussi visible sur le diagramme des bifurcations).

Question 5. Le code suivant permet d'obtenir les deux premières décimales de δ :

```
def cycle(mu):
    def f(x):
        return mu * x * (1 - x)
    x = 0.9
    for n in range(100000):
        x = f(x)
    lst = []
    for n in range(256):
        y = round(x, 4)
        if y not in lst:
            lst.append(y)
        x = f(x)
    return len(lst)

def cherche_periode(p, a, b):
    for k in range(20):
        mu = (a + b) / 2
        c = cycle(mu)
        if c < p:
            a = mu
        else:
            b = mu
    return mu
```

```
>>> mu0 = cherche_periode(8, 3, 4)
>>> mu1 = cherche_periode(16, mu0, 4)
>>> mu2 = cherche_periode(32, mu1, 4)
>>> print((mu1 - mu0) / (mu2 - mu1))
4.666449556604482
```