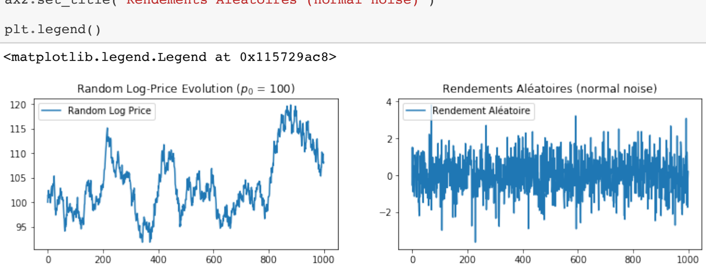
Données et Statistiques en Finance - TP 1

Jeffrey Kaikati & Lorenzo Pugliese

1) Créer un prix aléatoire

[2]:
```python
import numpy as np
import matplotlib.pyplot as plt

NIT = 1000
epsilons = np.random.normal(size=NIT)

p_t = [100]
t = [0]
for i, epsilon in enumerate(epsilons):
    p_t.append(epsilon + p_t[-1])
    t.append(i+1)

figure = plt.figure(figsize=(20,3))

ax1 = figure.add_subplot(131)
ax1.set_title("Random Log-Price Evolution ($p_0$ = 100)")
ax1.plot(t,p_t,label="Random Log Price")

plt.legend()

ax2 = figure.add_subplot(132)
ax2.plot(epsilons, label="Rendement Aléatoire")
ax2.set_title('Rendements Aléatoires (normal noise)')

plt.legend()
```
t[2]:
```
<matplotlib.legend.Legend at 0x115729ac8>
```



[ ]:

2) Mélange d'agents aléatoires et stratégiques

Stratégie conditionnelle ($a_{1,t} = +sgn(r_t)$)

[3]:
```python
phi_1 = 1
r_0 = np.random.normal()

def sign(x):
    return int(x/abs(x))

p_t = [100]
p_t_sans_agent = [100]
r_t = [r_0]
r_t_variance, r_t_esperance = 0,r_0/NIT

time = list(range(NIT))
for t in range(1,NIT):
    w_1_t = phi_1 * sign(r_t[t-1])
    r_t.append(epsilons[t] + w_1_t)
    p_t.append(p_t[-1] + r_t[-1])
    p_t_sans_agent.append(p_t_sans_agent[-1] + epsilons[t])

    r_t_esperance += r_t[-1]/NIT
    r_t_variance += (r_t[-1]**2)/NIT

r_t_variance -= r_t_esperance**2

autocorrelation_sans_agents = []
autocorrelation = []
for l in range(31):
    covariance_l_agents = 0
    covariance_l = 0
    r_t_esperance_sans_agent = 0

    for j in range(len(r_t)):
        r_t_esperance_sans_agent += epsilons[j]/len(r_t)

    for i in range(len(r_t)-l):
        covariance_l_agents += (r_t[i]-r_t_esperance)*(r_t[i+l]-r_t_esperance)/(len(r_t)-l)
        covariance_l += (epsilons[i]-r_t_esperance_sans_agent)*(epsilons[i+l]-r_t_esperance_sans_agent)/(len(r_t
)-l)

    autocorrelation.append(covariance_l_agents)
    autocorrelation_sans_agents.append(covariance_l)

figure = plt.figure(figsize=(20,3))

ax1 = figure.add_subplot(131)
ax1.set_title("Price Evolution (Conditional Strategy)")
ax1.plot(time,p_t,label="Log Price ($\phi_1$ = " + str(phi_1) + ")")
ax1.plot(time,p_t_sans_agent,label="Log Price ($\phi_1$ = 0)")

plt.legend()

ax2 = figure.add_subplot(132)
ax2.plot(time, r_t, label="Rendements")
ax2.set_title('Rendements (Conditional Strategy)')
ax2.plot(time,[r_t_esperance]*len(time),label="Expected value (1 agent)")

ax2.plot(time,[r_t_esperance + np.sqrt(r_t_variance)]*len(time),label="+ 1 $\sigma$",color='r')
ax2.plot(time,[r_t_esperance - np.sqrt(r_t_variance)]*len(time),label="- 1 $\sigma$",color='r')

plt.legend()

ax3 = figure.add_subplot(133)
ax3.set_title("Autocorrelogram (Conditional Strategy)")
ax3.scatter(range(31),autocorrelation, label="Autocorrelation (1 agent)")
ax3.scatter(range(31),autocorrelation_sans_agents, label="Autocorrelation")

plt.legend()
plt.grid()
```

Observations :

- Les prix obtenus dans le cadre de la stratégie conditionnelle présentent les mêmes variations que les prix aléatoires mais avec une plus grande magnitude (il faudrait cependant effectuer de nombreux tirages pour vérifier que ce comportement est significatif) ; il faut également remarquer que les rendements présentent des oscillations qui paraissent assez régulières en fonction du temps (période aux alentours de la centaine d'unités de temps).
- On constate que l'autocorrélation décroît quasiment en x^{-1} avec 1 agent alors qu'elle est presque nulle dès un pas de 1 dans le cas purement aléatoire : on peut affirmer que les rendements sont devenus nettement plus prévisibles (autocorrélation négligeable à partir d'un pas de 15 seulement).

Stratégie MA ($a_{1,t} = +sgn(p_t - MA_{k,t})$)

[4]:
```python
#initialisations
phi_1 = 1
r_0 = np.random.normal()
k = 7

def sign(x):
    return int(x/abs(x))

def suivi_tendance_k(p_t,k):
    # On vérifie qu'on a assez de points pour pouvoir faire la moyenne mobile
    test = len(p_t) - k

    if test >=0: # On a assez de points
        MA_k = 0
        for i in range(len(p_t)-k,len(p_t)):
            MA_k += p_t[i] / k

        return sign(p_t[-1] - MA_k)

    else: # On n'a pas assez de points, mais on fait avec : on calcule une moyenne uniquement avec les points di
sponibles
        moy = 0
        for i in range(0,len(p_t)):
            moy += p_t[i]/len(p_t)
        return moy

#stratégie conditionnelle
p_t = [100]
p_t_sans_agent = [100]
r_t = [r_0]
r_t_variance, r_t_esperance = 0,r_0/NIT

time = list(range(NIT))
for t in range(1,NIT):
    w_1_t = phi_1 * suivi_tendance_k(r_t,k)
    r_t.append(epsilons[t] + w_1_t)
    p_t.append(p_t[-1] + r_t[-1])
    p_t_sans_agent.append(p_t_sans_agent[-1] + epsilons[t])

    r_t_esperance += r_t[-1]/NIT
    r_t_variance += (r_t[-1]**2)/NIT

r_t_variance -= r_t_esperance**2

autocorrelation_sans_agents = []
autocorrelation = []
for l in range(31):
    covariance_l_agents = 0
    covariance_l = 0
    r_t_esperance_sans_agent = 0

    for j in range(len(r_t)):
        r_t_esperance_sans_agent += epsilons[j]/len(r_t)

    for i in range(len(r_t)-l):
        covariance_l_agents += (r_t[i]-r_t_esperance)*(r_t[i+l]-r_t_esperance)/(len(r_t)-l)
        covariance_l += (epsilons[i]-r_t_esperance_sans_agent)*(epsilons[i+l]-r_t_esperance_sans_agent)/(len(r_t
)-l)

    autocorrelation.append(covariance_l_agents)
    autocorrelation_sans_agents.append(covariance_l)

figure = plt.figure(figsize=(20,3))

ax1 = figure.add_subplot(131)
ax1.set_title("Price Evolution (MA Strategy)")
ax1.plot(time,p_t,label="Log Price ($\phi_1$ = " + str(phi_1) + ")")
ax1.plot(time,p_t_sans_agent,label="Log Price ($\phi_1$ = 0)")

plt.legend()

ax2 = figure.add_subplot(132)
ax2.plot(time, r_t, label="Rendements")
ax2.set_title('Rendements (MA Strategy)')
ax2.plot(time,[r_t_esperance]*len(time),label="Expected value (1 agent)")

ax2.plot(time,[r_t_esperance + np.sqrt(r_t_variance)]*len(time),label="+ 1 $\sigma$",color='r')
ax2.plot(time,[r_t_esperance - np.sqrt(r_t_variance)]*len(time),label="- 1 $\sigma$",color='r')

plt.legend()

ax3 = figure.add_subplot(133)
ax3.set_title("Autocorrelogram (MA Strategy)")
ax3.scatter(range(31),autocorrelation, label="Autocorrelation (1 agent)")
ax3.scatter(range(31),autocorrelation_sans_agents, label="Autocorrelation")

plt.legend()
plt.grid()
```

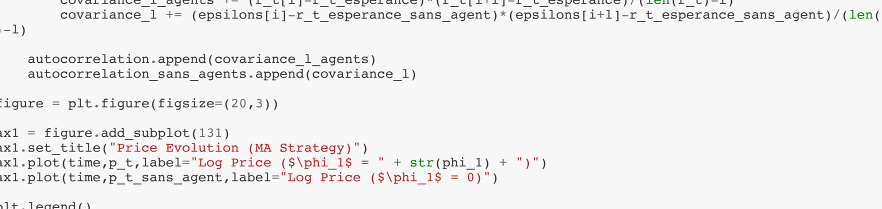
Observations :

- Les prix obtenus dans le cadre de la stratégie MA semblent suivre de manière assez semblable les variations des prix aléatoires (il faudrait cependant effectuer de nombreux tirages pour vérifier que ce comportement est significatif) ; il faut également remarquer que les rendements présentent des oscillations qui paraissent assez régulières en fonction du temps (période très faible, de l'ordre des quelques unités de temps).
- On constate que l'autocorrélation présent un comportement d'oscillateur amorti avec 1 agent alors qu'elle est presque nulle dès 1 pas dans le cas purement aléatoire : on peut affirmer que les rendements sont devenus nettement plus prévisibles (autocorrélation négligeable à partir d'un pas de 30 seulement).

3.4) Variance de r en fonction de ϕ_1

Stratégie conditionnelle ($a_{1,t} = +sgn(r_t)$)

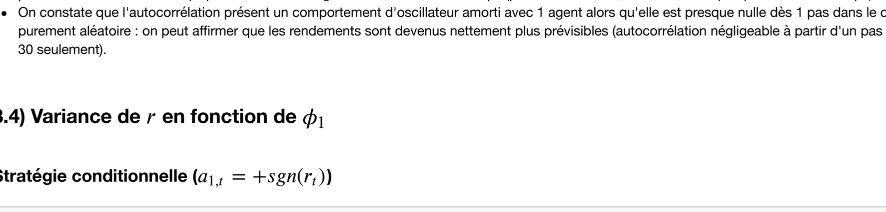
[4]:
```python
phi_1_values = [0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]
r_0 = np.random.normal()

def sign(x):
    return int(x/abs(x))

p_t = [[100],[100],[100],[100],[100],[100],[100],[100],[100],[100],[100]]
p_t_sans_agent = [[100],[100],[100],[100],[100],[100],[100],[100],[100],[100],[100]]
r_t = [[epsilons[0]],[r_0],[r_0],[r_0],[r_0],[r_0],[r_0],[r_0],[r_0],[r_0],[r_0]]
r_t_variance, r_t_esperance = [0]*11,[epsilons[0]/NIT] + [r_0/NIT]*10

for i,phi_1 in enumerate(phi_1_values):

    time = list(range(NIT))
    for t in range(1,NIT):
        w_1_t = phi_1 * sign(r_t[i][t-1])
        r_t[i].append(epsilons[t] + w_1_t)
        p_t[i].append(p_t[i][-1] + r_t[i][-1])

        r_t_esperance[i] += r_t[i][-1]/NIT
        r_t_variance[i] += (r_t[i][-1]**2)/NIT

    r_t_variance[i] -= r_t_esperance[i]**2

figure = plt.figure(figsize=(20,3))

ax1 = figure.add_subplot(111)
ax1.set_title("$\mathbb{V} \ [r](\phi_1)$ (Conditional Strategy)")
ax1.scatter(phi_1_values,r_t_variance,label="$\mathbb{V} \ [r](\phi_1)$")

plt.legend()
```
t[4]:
```
<matplotlib.legend.Legend at 0x11eb3c5f8>
```



On remarque que la variance augmente suivant un comportement en $\propto \phi_1^2$ pour la stratégie conditionnelle : plus l'influence des agents est importante (ϕ_1 est grand en valeur absolue) et plus les rendements présentent des variations disparates (logique car plus il y a de personnes sur un marché et plus on aura des outcomes différents) ; la volatilité augmente (la variance étant une mesure de la volatilité).

Stratégie MA ($a_{1,t} = +sgn(p_t - MA_{k,t})$)

[5]:
```python
phi_1_values = [0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]
r_0 = np.random.normal()

def sign(x):
    return int(x/abs(x))

p_t = [[100],[100],[100],[100],[100],[100],[100],[100],[100],[100],[100]]
p_t_sans_agent = [[100],[100],[100],[100],[100],[100],[100],[100],[100],[100],[100]]
r_t = [[epsilons[0]],[r_0],[r_0],[r_0],[r_0],[r_0],[r_0],[r_0],[r_0],[r_0],[r_0]]
r_t_variance, r_t_esperance = [0]*11,[epsilons[0]/NIT] + [r_0/NIT]*10

for i,phi_1 in enumerate(phi_1_values):

    time = list(range(NIT))
    for t in range(1,NIT):
        w_1_t = phi_1 * suivi_tendance_k(r_t[i],k)
        r_t[i].append(epsilons[t] + w_1_t)
        p_t[i].append(p_t[i][-1] + r_t[i][-1])

        r_t_esperance[i] += r_t[i][-1]/NIT
        r_t_variance[i] += (r_t[i][-1]**2)/NIT

    r_t_variance[i] -= r_t_esperance[i]**2

figure = plt.figure(figsize=(20,3))

ax1 = figure.add_subplot(111)
ax1.set_title("$\mathbb{V} \ [r](\phi_1)$ (MA Strategy)")
ax1.scatter(phi_1_values,r_t_variance,label="$\mathbb{V} \ [r](\phi_1)$")

plt.legend()
```
t[5]:
```
<matplotlib.legend.Legend at 0x11eb2b940>
```

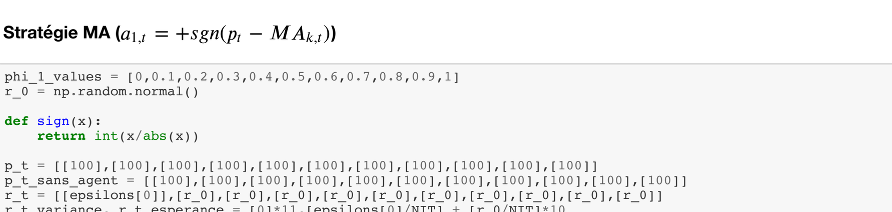


On remarque que la variance augmente suivant un comportement en $\propto \phi_1^2$ pour la stratégie MA : plus l'influence des agents est importante (ϕ_1 est grand en valeur absolue) et plus les rendements présentent des variations disparates (logique car plus il y a de personnes sur un marché et plus on aura des outcomes différents) ; la volatilité augmente (la variance étant une mesure de la volatilité).

3.5) Autocorrélation à 1 pas en fonction de ϕ_1

Stratégie conditionnelle ($a_{1,t} = +sgn(r_t)$)

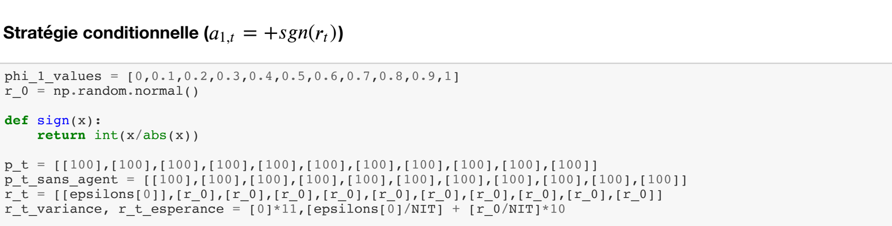
[6]:
```python
phi_1_values = [0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]
r_0 = np.random.normal()

def sign(x):
    return int(x/abs(x))

p_t = [[100],[100],[100],[100],[100],[100],[100],[100],[100],[100],[100]]
p_t_sans_agent = [[100],[100],[100],[100],[100],[100],[100],[100],[100],[100],[100]]
r_t = [[epsilons[0]],[r_0],[r_0],[r_0],[r_0],[r_0],[r_0],[r_0],[r_0],[r_0],[r_0]]
r_t_variance, r_t_esperance = [0]*11,[epsilons[0]/NIT] + [r_0/NIT]*10

for i,phi_1 in enumerate(phi_1_values):

    time = list(range(NIT))
    for t in range(1,NIT):
        w_1_t = phi_1 * sign(r_t[i][t-1])
        r_t[i].append(epsilons[t] + w_1_t)
        p_t[i].append(p_t[i][-1] + r_t[i][-1])

        r_t_esperance[i] += r_t[i][-1]/NIT
        r_t_variance[i] += (r_t[i][-1]**2)/NIT

    r_t_variance[i] -= r_t_esperance[i]**2

autocorrelations = []

for i,phi_1 in enumerate(phi_1_values):
    covariance_1_agents = 0

    for j in range(len(r_t[i])-1):
        covariance_1_agents += (r_t[i][j]-r_t_esperance[i])*(r_t[i][j+1]-r_t_esperance[i])/(len(r_t[i])-1)

    autocorrelations.append(covariance_1_agents)

figure = plt.figure(figsize=(20,3))

ax1 = figure.add_subplot(111)
ax1.set_title("Autocorrelogram as a function of $\phi_1$ (Conditional Strategy)")
ax1.scatter(phi_1_values,autocorrelations,label="Autocorrelation")

plt.legend()
```
t[6]:
```
<matplotlib.legend.Legend at 0x11e483710>
```



On remarque que la variance augmente suivant un comportement en $\propto \phi_1$ pour la stratégie conditionnelle : plus l'influence des agents est importante (ϕ_1 est grand en valeur absolue) et plus l'autocorrélation à 1 pas devient forte (logique car plus il y a de personnes sur un marché qui appliquent la même stratégie et plus les rendements deviennent prévisibles).

Stratégie MA ($a_{1,t} = +sgn(p_t - MA_{k,t})$)

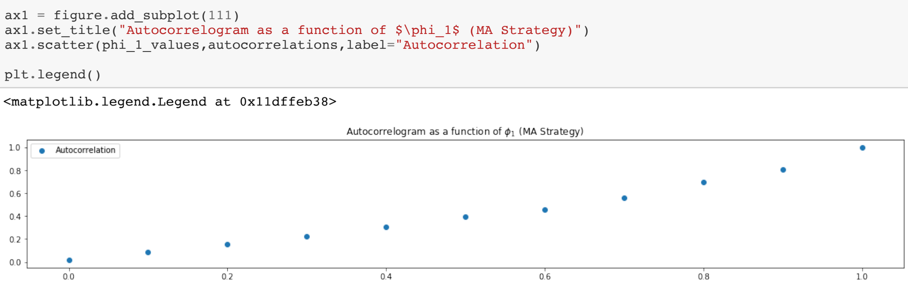
[7]:
```python
phi_1_values = [0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]
r_0 = np.random.normal()

def sign(x):
    return int(x/abs(x))

p_t = [[100],[100],[100],[100],[100],[100],[100],[100],[100],[100],[100]]
p_t_sans_agent = [[100],[100],[100],[100],[100],[100],[100],[100],[100],[100],[100]]
r_t = [[epsilons[0]],[r_0],[r_0],[r_0],[r_0],[r_0],[r_0],[r_0],[r_0],[r_0],[r_0]]
r_t_variance, r_t_esperance = [0]*11,[epsilons[0]/NIT] + [r_0/NIT]*10

for i,phi_1 in enumerate(phi_1_values):

    time = list(range(NIT))
    for t in range(1,NIT):
        w_1_t = phi_1 * suivi_tendance_k(r_t[i],k)
        r_t[i].append(epsilons[t] + w_1_t)
        p_t[i].append(p_t[i][-1] + r_t[i][-1])

        r_t_esperance[i] += r_t[i][-1]/NIT
        r_t_variance[i] += (r_t[i][-1]**2)/NIT

    r_t_variance[i] -= r_t_esperance[i]**2

autocorrelations = []

for i,phi_1 in enumerate(phi_1_values):
    covariance_1_agents = 0

    for j in range(len(r_t[i])-1):
        covariance_1_agents += (r_t[i][j]-r_t_esperance[i])*(r_t[i][j+1]-r_t_esperance[i])/(len(r_t[i])-1)

    autocorrelations.append(covariance_1_agents)

figure = plt.figure(figsize=(20,3))

ax1 = figure.add_subplot(111)
ax1.set_title("Autocorrelogram as a function of $\phi_1$ (MA Strategy)")
ax1.scatter(phi_1_values,autocorrelations,label="Autocorrelation")

plt.legend()
```
t[7]:
```
<matplotlib.legend.Legend at 0x11dffeb38>
```



On remarque que la variance augmente suivant un comportement en $\propto \phi_1$ pour la stratégie MA : plus l'influence des agents est importante (ϕ_1 est grand en valeur absolue) et plus l'autocorrélation à 1 pas devient forte (logique car plus il y a de personnes sur un marché qui appliquent la même stratégie et plus les rendements deviennent prévisibles).

4) Plusieurs stratégies

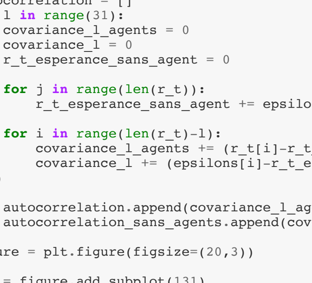
[8]:
```python
#initialisations
phi_1 = 1
phi_2 = 0.5
r_0 = np.random.normal()
k_1 = 7
k_2 = 9

def sign(x):
    return int(x/abs(x))

def suivi_tendance_k(p_t,k):
    # On vérifie qu'on a assez de points pour pouvoir faire la moyenne mobile
    test = len(p_t) - k

    if test >=0: # On a assez de points
        MA_k = 0
        for i in range(len(p_t)-k,len(p_t)):
            MA_k += p_t[i] / k

        return sign(p_t[-1] - MA_k)

    else: # On n'a pas assez de points, mais on fait avec : on calcule une moyenne uniquement avec les points di
sponibles
        moy = 0
        for i in range(0,len(p_t)):
            moy += p_t[i]/len(p_t)
        return moy

#stratégie conditionnelle
p_t = [100]
p_t_sans_agent = [100]
r_t = [r_0]
r_t_variance, r_t_esperance = 0,r_0/NIT

time = list(range(NIT))
for t in range(1,NIT):
    w_1_t = phi_1 * suivi_tendance_k(r_t,k_1) - phi_2 * suivi_tendance_k(r_t,k_2)
    r_t.append(epsilons[t] + w_1_t)
    p_t.append(p_t[-1] + r_t[-1])
    p_t_sans_agent.append(p_t_sans_agent[-1] + epsilons[t])

    r_t_esperance += r_t[-1]/NIT
    r_t_variance += (r_t[-1]**2)/NIT

r_t_variance -= r_t_esperance**2

autocorrelation_sans_agents = []
autocorrelation = []
for l in range(31):
    covariance_l_agents = 0
    covariance_l = 0
    r_t_esperance_sans_agent = 0

    for j in range(len(r_t)):
        r_t_esperance_sans_agent += epsilons[j]/len(r_t)

    for i in range(len(r_t)-l):
        covariance_l_agents += (r_t[i]-r_t_esperance)*(r_t[i+l]-r_t_esperance)/(len(r_t)-l)
        covariance_l += (epsilons[i]-r_t_esperance_sans_agent)*(epsilons[i+l]-r_t_esperance_sans_agent)/(len(r_t
)-l)

    autocorrelation.append(covariance_l_agents)
    autocorrelation_sans_agents.append(covariance_l)

figure = plt.figure(figsize=(20,3))

ax1 = figure.add_subplot(131)
ax1.set_title("Price Evolution (MA Strategy with 2 agents)")
ax1.plot(time,p_t,label="Log Price ($\phi_1$ = " + str(phi_1) + ", " + " $\phi_2$ = " + str(phi_2) +")")
ax1.plot(time,p_t_sans_agent,label="Log Price ($\phi_1 = \phi_2$ = 0)")

plt.legend()

ax2 = figure.add_subplot(132)
ax2.plot(time, r_t, label="Rendements")
ax2.set_title('Rendements (MA Strategy with 2 agents)')
ax2.plot(time,[r_t_esperance]*len(time),label="Expected value (2 agents)")

ax2.plot(time,[r_t_esperance + np.sqrt(r_t_variance)]*len(time),label="+ 1 $\sigma$",color='r')
ax2.plot(time,[r_t_esperance - np.sqrt(r_t_variance)]*len(time),label="- 1 $\sigma$",color='r')

plt.legend()

ax3 = figure.add_subplot(133)
ax3.set_title("Autocorrelogram (MA Strategy with 2 agents)")
ax3.scatter(range(31),autocorrelation, label="Autocorrelation (2 agents)")
ax3.scatter(range(31),autocorrelation_sans_agents, label="Autocorrelation")

plt.legend()
plt.grid()
```



4.1) Variance des rendements bidimensionnelle

Stratégie MA ($a_{1,t} = +sgn(p_t - MA_{k,t})$)

[9]:
```python
from mpl_toolkits import mplot3d

fig = plt.figure()
ax = plt.axes(projection='3d')

phi_1_values = [0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]
phi_2_values = [0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]
r_0 = np.random.normal()

colors = ["black","red","chocolate","darkorange","lawngreen","forestgreen","turquoise","dodgerblue","navy","blue
violet","magenta"]

def sign(x):
    return int(x/abs(x))

p_t = []
p_t_sans_agent = []
r_t = []
r_t_variance = [0] * 11**2
r_t_esperance = ([epsilons[0]/NIT] + [r_0/NIT]*10)*11

for k in range(11**2):
    p_t.append([100])
    p_t_sans_agent.append([100])

    if k%11 == 0:
        r_t.append([epsilons[0]])
    else:
        r_t.append([r_0])

for i,phi_1 in enumerate(phi_1_values):
    for j,phi_2 in enumerate(phi_2_values):

        time = list(range(NIT))
        for t in range(1,NIT):
            w_1_t = phi_1 * suivi_tendance_k(r_t[11*i+j],k) - phi_2 * suivi_tendance_k(r_t[11*i+j],k_2)
            r_t[11*i+j].append(epsilons[t] + w_1_t)
            p_t[11*i+j].append(p_t[11*i+j][-1] + r_t[11*i+j][-1])

            r_t_esperance[11*i+j] += r_t[11*i+j][-1]/NIT
            r_t_variance[11*i+j] += (r_t[11*i+j][-1]**2)/NIT

        r_t_variance[11*i+j] -= r_t_esperance[11*i+j]**2

for i,phi_1 in enumerate(phi_1_values):
    X = []
    Y = []
    Z = []

    for j,phi_2 in enumerate(phi_2_values):
            X.append(phi_1)
            Y.append(phi_2)
            Z.append(r_t_variance[11*i+j])

    ax.scatter3D(X,Y,Z,c=colors[i])

ax.set_title("$\mathbb{V} \ [r](\phi_1, \phi_2)$ (MA Strategy)")
ax.set_xlabel("$\phi_1$")
ax.set_ylabel("$\phi_2$")
```
t[9]:
```
Text(0.5, 0, '$\\phi_2$')
```

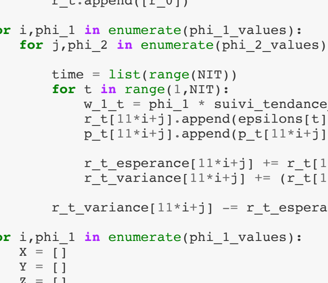


On remarque que la variance augmente suivant un comportement en $\propto \phi_1^2 \times \phi_2^2$ pour la stratégie MA : l'analyse est finalement la même que celle dans le cadre unidimensionnel.

4.2) Autocorrélation à 1 pas en fonction de (ϕ_1, ϕ_2)

Stratégie MA ($a_{1,t} = +sgn(p_t - MA_{k,t})$)

[10]:
```python
phi_1_values = [0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]
phi_2_values = [0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]
r_0 = np.random.normal()

fig = plt.figure()
ax = plt.axes(projection='3d')

def sign(x):
    return int(x/abs(x))

p_t = []
p_t_sans_agent = []
r_t = []
r_t_variance = [0] * 11**2
r_t_esperance = ([epsilons[0]/NIT] + [r_0/NIT]*10)*11

for k in range(11**2):
    p_t.append([100])
    p_t_sans_agent.append([100])

    if k%11 == 0:
        r_t.append([epsilons[0]])
    else:
        r_t.append([r_0])

for i,phi_1 in enumerate(phi_1_values):
    for j,phi_2 in enumerate(phi_2_values):

        time = list(range(NIT))
        for t in range(1,NIT):
            w_1_t = phi_1 * suivi_tendance_k(r_t[11*i+j],k) - phi_2 * suivi_tendance_k(r_t[11*i+j],k_2)
            r_t[11*i+j].append(epsilons[t] + w_1_t)
            p_t[11*i+j].append(p_t[11*i+j][-1] + r_t[11*i+j][-1])

            r_t_esperance[11*i+j] += r_t[11*i+j][-1]/NIT
            r_t_variance[11*i+j] += (r_t[11*i+j][-1]**2)/NIT

        r_t_variance[11*i+j] -= r_t_esperance[11*i+j]**2

autocorrelations = []

for i,phi_1 in enumerate(phi_1_values):
    for a,phi_2 in enumerate(phi_2_values):
        covariance_1_agents = 0

        for j in range(len(r_t[11*i+a])-1):
            covariance_1_agents += (r_t[11*i+a][j]-r_t_esperance[11*i+a])*(r_t[11*i+a][j+1]-r_t_esperance[11*i+a
])/(len(r_t[11*i+a])-1)

        autocorrelations.append(covariance_1_agents)

for i,phi_1 in enumerate(phi_1_values):
    X = []
    Y = []
    Z = []
    for j,phi_2 in enumerate(phi_2_values):
            X.append(phi_1)
            Y.append(phi_2)
            Z.append(autocorrelations[11*i+j])

    ax.scatter3D(X,Y,Z,c=colors[i])

ax.set_title("Autocorrelogram as a function of $\phi_1, \phi_2$ (MA Strategy)")
ax.set_xlabel("$\phi_1$")
ax.set_ylabel("$\phi_2$")
```
[10]:
```
Text(0.5, 0, '$\\phi_2$')
```



On remarque que la variance augmente suivant un comportement en $\propto \phi_1 \times \phi_2$ pour la stratégie MA. L'analyse est finalement proche de celle dans le cadre unidimensionnel à la seule différence qu'ici quand $\phi_1 = \phi_2$ on revient sur des autocorrélations proches de 0 : quand les 2 stratégies sont aussi influentes sur le marché (par exemple quand on a le même nombre d'acteurs les appliquant) alors les effets se compensent contrairement au cas de la variance, ce qui veut dire que les rendements redeviennent imprédictibles.

[ ]:

Données et Statistiques en Finance - TP 2

Jeffrey Kaikati & Lorenzo Pugliese

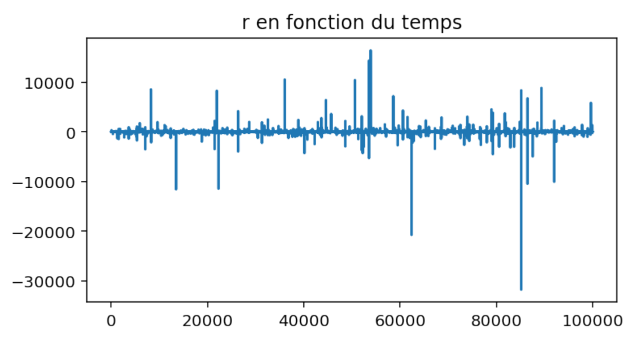
1) Simulation de r_t

In [9]:
```
import matplotlib.pyplot as plt
import numpy as np
import powerlaw
from mpl_toolkits.mplot3d import Axes3D
```

In [2]:
```
(sigma, alpha) = (2 , 1.5)
N = 100000 # essayer 10,000
r = [1]
alpha_hat = [1.1]
epsilons = np.random.normal(size=N+1)*sigma

for i in range(N):
    #j'ajoute alpha_hat_(i+1)
    alpha_hat.append(alpha + epsilons[i+1]/r[-1])
    #j'ajoute r_(i+1)
    r.append((alpha - alpha_hat[i])*r[-1]+epsilons[i+1])
time = list(range(N))

figure = plt.figure(figsize=(20,3))

ax1 = figure.add_subplot(131)
ax1.set_title("r en fonction du temps")
ax1.plot(r)
```

Out[2]:
```
[<matplotlib.lines.Line2D at 0x7f3f79850f60>]
```

Out[2]:



On remarque qu'il existe des pics très élevés qui marquent le caractère "explosif" de la méthode (pics de volatilité). En effet, cela s'explique par le choix :

$$\alpha > 1$$

.

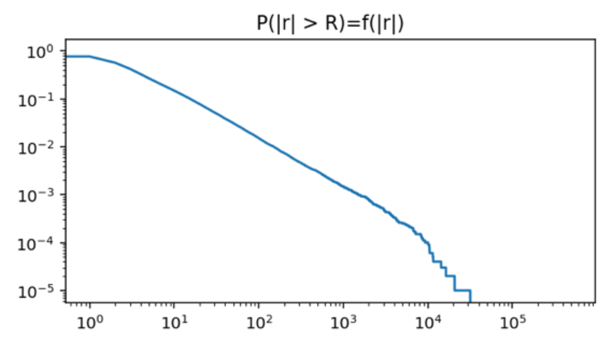
2.1) Tracé de $\mathbb{P}(|r| > R) = f(|r|)$

In [3]:
```
from statsmodels.distributions.empirical_distribution import ECDF
```

In [4]:
```
ecdf = ECDF([abs(x) for x in r])
R = [i for i in range(500000)]
#plt.loglog(R,[1 - x for x in ecdf(R)])


figure = plt.figure(figsize=(20,3))

ax1 = figure.add_subplot(131)
ax1.set_title("P(|r| > R)=f(|r|)")
ax1.loglog(R,[1 - x for x in ecdf(R)])
```

Out[4]:
```
[<matplotlib.lines.Line2D at 0x7f3f65503a90>]
```

Out[4]:



À partir de $|r| = 10^1$, on remarque que l'allure de la courbe log-log est celle d'une fonction affine décroissante. Cela signifie que la grandeur $\mathbb{P}(|r| > R)$ est liée à $|r|$ par une relation de puissance. Le bout de la queue n'est pas significatif puisque pour des valeurs supérieures à $|r| = 10^4$ les probabilités d'occurence sont très faibles (d'où le caractère discret).

2.3) Exposant de la queue

In [8]:
```
mypl = powerlaw.Fit(np.abs(r))
mypl.alpha
```

1.93197410718067

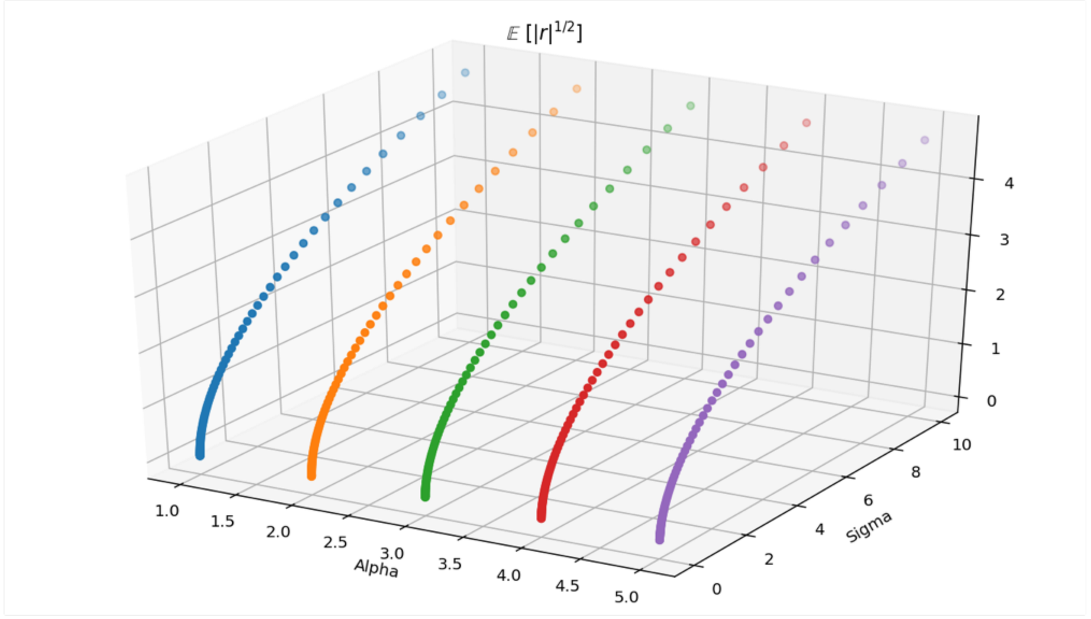
Comme P(r) est *heavy-tailed*, on s'attend à avoir une exposant proche de 2 (comme décrit dans le cours) et on retrouve effectivement un tel comportement sur les résultats expérimentaux.

3.1) Dépendance en α et σ de $\mathbb{E}\ [|r|^2]$

In [6]:
```
alpha_list = [1,2,3,4,5]
sigma_list = np.logspace(-3,1,100)

N = 10000
r = {}
alpha_hat = {}

for alpha in alpha_list:
    for sigma in sigma_list:
        r[(alpha,sigma)] = [1]
        alpha_hat[(alpha,sigma)] = [1.1]

l = np.random.normal(size=N+1)

for alpha in alpha_list:
    for sigma in sigma_list:
        epsilons = l * sigma
        for i in range(N):
            #j'ajoute alpha_hat_(i+1)
            alpha_hat[(alpha,sigma)].append(alpha + epsilons[i+1]/r[(alpha,sigma)][-1])
            #j'ajoute r_(i+1)
            r[(alpha,sigma)].append((alpha - alpha_hat[(alpha,sigma)][i])*r[(alpha,sigma)][-1]+epsilons[i+1])

fig = plt.figure()
ax = plt.axes(projection='3d')

for alpha in alpha_list:
    X = []
    Y = []
    Z = []
    for sigma in sigma_list:
            X.append(alpha)
            Y.append(sigma)
            Z.append(sum([abs(x)**0.5 for x in r[(alpha,sigma)]])/len(r[(alpha,sigma)]))

    ax.scatter3D(X,Y,Z)

ax.set_title("$\mathbb{E} \ [|r|^{1/2}]$")
ax.set_xlabel("Alpha")
ax.set_ylabel("Sigma")
```

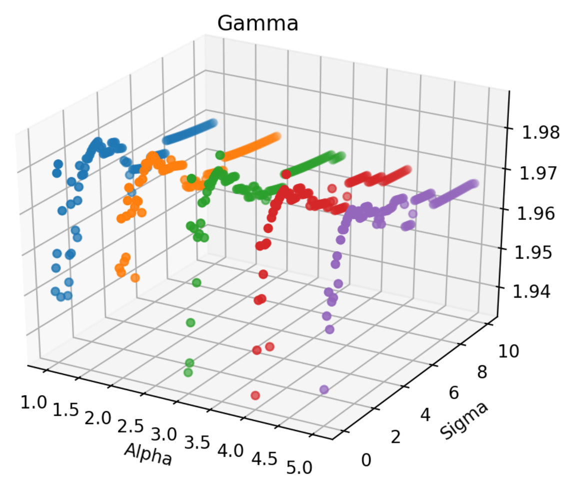
Out[6]:
```
Text(0.5, 0, 'Sigma')
```

Out[6]:



Il semblerait que $\mathbb{E}\ [|r|^2]$ ne dépende pas de α. D'autre part, la dépendance en σ ressemble fortement à celle que l'on pourrait obtenir avec une loi de puissance plus petite que 1 (comme une racine carrée par exemple).

In [ ]:
```
alpha_list = [1,2,3,4,5]
sigma_list = np.logspace(-3,1,100)

N = 10000
r = {}
alpha_hat = {}
gamma_list = {}

for alpha in alpha_list:
    for sigma in sigma_list:
        r[(alpha,sigma)] = [1]
        alpha_hat[(alpha,sigma)] = [1.1]

l = np.random.normal(size=N+1)

for alpha in alpha_list:
    for sigma in sigma_list:
        epsilons = l * sigma
        for i in range(N):
            #j'ajoute alpha_hat_(i+1)
            alpha_hat[(alpha,sigma)].append(alpha + epsilons[i+1]/r[(alpha,sigma)][-1])
            #j'ajoute r_(i+1)
            r[(alpha,sigma)].append((alpha - alpha_hat[(alpha,sigma)][i])*r[(alpha,sigma)][-1]+epsilons[i+1])

        mypl = powerlaw.Fit(np.abs(r[(alpha,sigma)]))
        gamma_list[(alpha,sigma)] = mypl.alpha

fig = plt.figure()
ax = plt.axes(projection='3d')

for alpha in alpha_list:
    X = []
    Y = []
    Z = []
    for sigma in sigma_list:
            X.append(alpha)
            Y.append(sigma)
            Z.append(gamma_list[(alpha,sigma)])

    ax.scatter3D(X,Y,Z)

ax.set_title("Gamma")
ax.set_xlabel("Alpha")
ax.set_ylabel("Sigma")
```



On remarque que pour lorsque σ augmente γ se rapproche de 2 et ce quelle que soit la valeur de α (la dépendance en α semble négligeable à partir de $\sigma = 6$).

In [0]:
```
```

Données et Statistiques en Finance - TP 3

Jeffrey Kaikati & Lorenzo Pugliese

1) Chaos avec trois stratégies triviales

$$(1+r)p_t = \sum_{h=1}^{H} \frac{e^{\beta U_{h,t-1}}}{\sum_{j=1}^{H} e^{\beta U_{j,t-1}}}(p^*_{t+1} + f_{h,t}(p_{t-1} - p^*_{t-1}))$$
$$U_{h,t} = \lambda U_{h,t-1} + (1-\lambda)(p_t - (1+r)p_{t-1})(p^*_t - p^*_{t-1} + f_{h,t-1}(p_{t-2} - p^*_{t-2}, \dots))$$

On modélise p^*_t par une loi normale de paramètres (μ, σ) = (0,1)

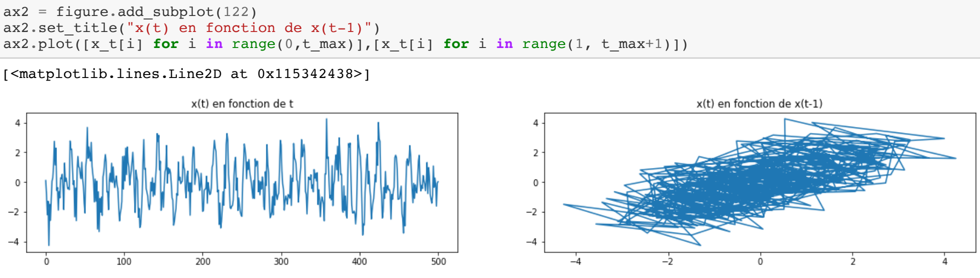
In [3]:
```
import matplotlib.pyplot as plt
import numpy as np
from mpmath import mp
from math import exp

lambd = 0.9
r = 0.01
t_max = 500
```

a) Prenons $\beta = 1$

Tracé de $x(t)$ en fonction de t et de $x(t)$ en fonction de $x(t-1)$

In [4]:
```
H = 3
U = [0.5] * H
beta = 1
g = 3
f = [0,g,-g]

x_t = [0.1]
n = [1] * H
p_etoile = np.random.normal(size = t_max +2)


for t in range(1,t_max + 1):

    resultat_partiel = 0

    for h in range(1,H+1):
        renormalisation = 0
        for j in range(1,H+1):
            renormalisation += mp.exp(beta * U[j-1])

        resultat_partiel += mp.exp(beta * U[h-1]) * (f[h-1] + p_etoile[t+1]) / ((1 + r) * renormalisation)

    x_t.append(resultat_partiel)

    for h in range(1,H+1):
        U[h-1] = lambd * U[h-1] + (1-lambd) * (x_t[-1] - (1+r) * x_t[-2] + p_etoile[t] - (1+r)*p_etoile[t-1]) *
(p_etoile[t] - p_etoile[t-1] + f[h-1])

figure = plt.figure(figsize=(20,3))
ax1 = figure.add_subplot(121)
ax1.set_title("x(t) en fonction de t")
ax1.plot(range(t_max+1),x_t)

ax2 = figure.add_subplot(122)
ax2.set_title("x(t) en fonction de x(t-1)")
ax2.plot([x_t[i] for i in range(0,t_max)],[x_t[i] for i in range(1, t_max+1)])
```

Out[4]: `[<matplotlib.lines.Line2D at 0x115342438>]`

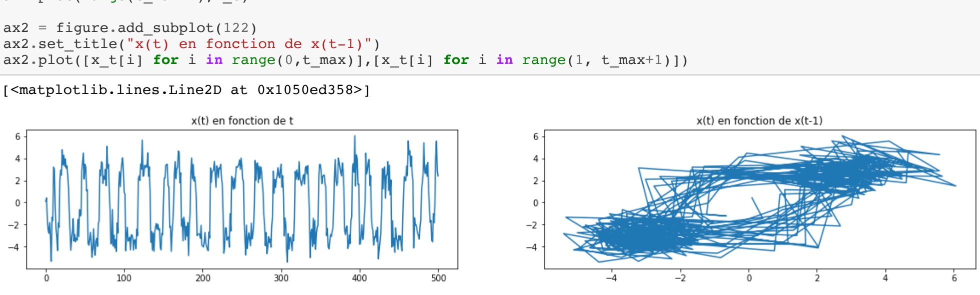


b) Prenons $\beta = 10$

Tracé de $x(t)$ en fonction de t et de $x(t)$ en fonction de x(t-1)

In [5]:
```
H = 3
U = [0.5] * H
beta = 10
g = 3
f = [0,g,-g]

x_t = [0.1]
n = [1] * H
p_etoile = np.random.normal(size = t_max +2)

for t in range(1,t_max + 1):

    resultat_partiel = 0

    for h in range(1,H+1):
        renormalisation = 0
        for j in range(1,H+1):
            renormalisation += mp.exp(beta * U[j-1])

        resultat_partiel += mp.exp(beta * U[h-1]) * (f[h-1] + p_etoile[t+1]) / ((1 + r) * renormalisation)

    x_t.append(resultat_partiel)

    for h in range(1,H+1):
        U[h-1] = lambd * U[h-1] + (1-lambd) * (x_t[-1] - (1+r) * x_t[-2] + p_etoile[t] - (1+r)*p_etoile[t-1]) *
(p_etoile[t] - p_etoile[t-1] + f[h-1])

figure = plt.figure(figsize=(20,3))
ax1 = figure.add_subplot(121)
ax1.set_title("x(t) en fonction de t")
ax1.plot(range(t_max+1),x_t)

ax2 = figure.add_subplot(122)
ax2.set_title("x(t) en fonction de x(t-1)")
ax2.plot([x_t[i] for i in range(0,t_max)],[x_t[i] for i in range(1, t_max+1)])
```

Out[5]: `[<matplotlib.lines.Line2D at 0x1050ed358>]`

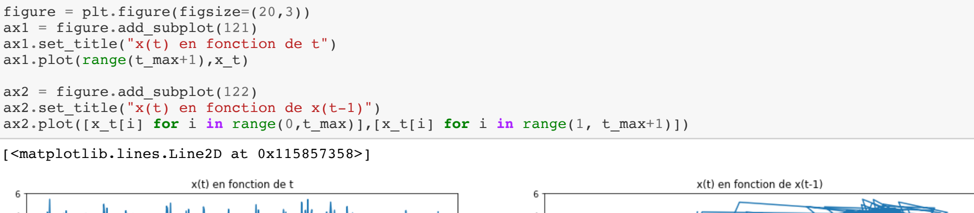


c) Prenons $\beta = 100$

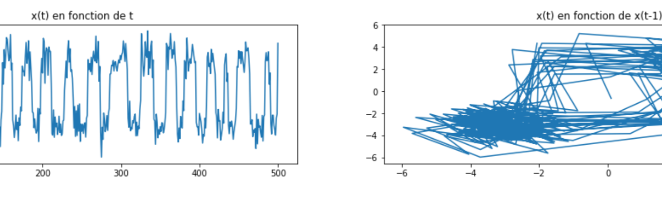
Tracé de $x(t)$ en fonction de t et de $x(t)$ en fonction de $x(t-1)$

In [6]:
```
H = 3
U = [0.5] * H
beta = 100
g = 3
f = [0,g,-g]

x_t = [0.1]
n = [1] * H
p_etoile = np.random.normal(size = t_max +2)

for t in range(1,t_max + 1):

    resultat_partiel = 0

    for h in range(1,H+1):
        renormalisation = 0
        for j in range(1,H+1):
            renormalisation += mp.exp(beta * U[j-1])

        resultat_partiel += mp.exp(beta * U[h-1]) * (f[h-1] + p_etoile[t+1]) / ((1 + r) * renormalisation)

    x_t.append(resultat_partiel)

    for h in range(1,H+1):
        U[h-1] = lambd * U[h-1] + (1-lambd) * (x_t[-1] - (1+r) * x_t[-2] + p_etoile[t] - (1+r)*p_etoile[t-1]) *
(p_etoile[t] - p_etoile[t-1] + f[h-1])

figure = plt.figure(figsize=(20,3))
ax1 = figure.add_subplot(121)
ax1.set_title("x(t) en fonction de t")
ax1.plot(range(t_max+1),x_t)

ax2 = figure.add_subplot(122)
ax2.set_title("x(t) en fonction de x(t-1)")
ax2.plot([x_t[i] for i in range(0,t_max)],[x_t[i] for i in range(1, t_max+1)])
```

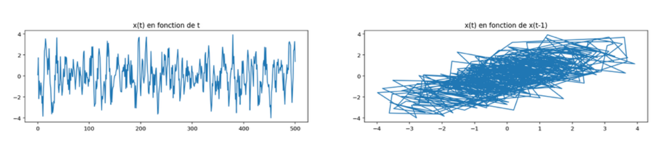
Out[6]: `[<matplotlib.lines.Line2D at 0x115857358>]`



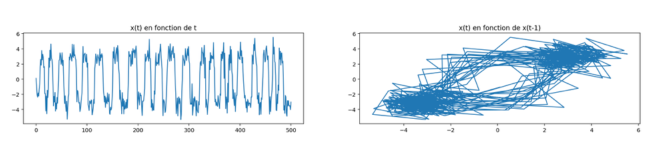
Résultats :
$\beta = 1$



$\beta = 10$



$\beta = 100$



On remarque qu'au fur et à mesure que β augmente, il y a une dilatation de x en fonction du temps et un élargissement de l'ellipse obtenue. En effet, on arrive de plus en plus à discerner l'intérieur de l'ellipse de son contour. Cela se traduit par le fait que plus β est grand, plus la différence entre les différentes stratégies sera exacerbée. Certaines stratégies sont prépondérantes, ce qui apaise le chaos.

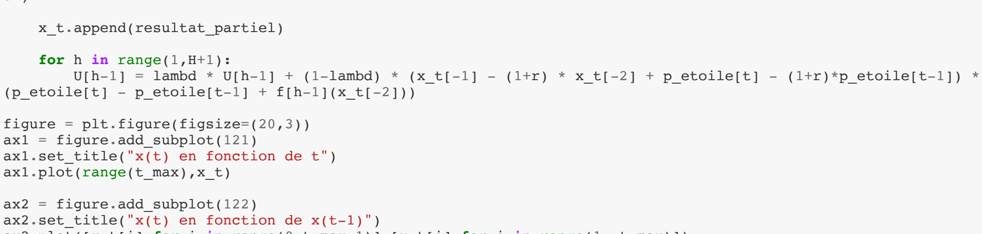
2) Stratégies traditionnelles

a) Prenons $\beta = 10$

Tracé de $x(t)$ en fonction de t et de $x(t)$ en fonction de $x(t-1)$

In [7]:
```
H = 3
U = [0.5] * H
beta = 10
g = 3

def f_0(x):
    return 0

def f_1(x):
    return 0.9 * x + 0.2

def f_2(x):
    return 0.9 * x - 0.2

def f_3(x):
    return (1+r) * x

f = [f_0,f_1,f_2,f_3]

x_t = [0.1]
n = [1] * H
p_etoile = np.random.normal(size = t_max +2)

for t in range(2,t_max + 1):

    resultat_partiel = 0

    for h in range(1,H+1):
        renormalisation = 0
        for j in range(1,H+1):
            renormalisation += mp.exp(beta * U[j-1])

        resultat_partiel += mp.exp(beta * U[h-1]) * (f[h-1](x_t[-1]) + p_etoile[t+1]) / ((1 + r) * renormalisati
on)

    x_t.append(resultat_partiel)

    for h in range(1,H+1):
        U[h-1] = lambd * U[h-1] + (1-lambd) * (x_t[-1] - (1+r) * x_t[-2] + p_etoile[t] - (1+r)*p_etoile[t-1]) *
(p_etoile[t] - p_etoile[t-1] + f[h-1](x_t[-2]))

figure = plt.figure(figsize=(20,3))
ax1 = figure.add_subplot(121)
ax1.set_title("x(t) en fonction de t")
ax1.plot(range(t_max),x_t)

ax2 = figure.add_subplot(122)
ax2.set_title("x(t) en fonction de x(t-1)")
ax2.plot([x_t[i] for i in range(0,t_max-1)],[x_t[i] for i in range(1, t_max)])
```

Out[7]: `[<matplotlib.lines.Line2D at 0x115b45898>]`

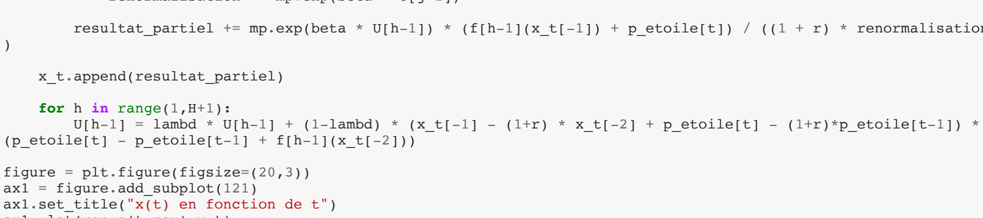


b) Prenons $\beta = 100$

Tracé de $x(t)$ en fonction de t et de $x(t)$ en fonction de $x(t-1)$

In [8]:
```
H = 3
U = [0.5] * H
beta = 100
g = 3

def f_0(x):
    return 0

def f_1(x):
    return 0.9 * x + 0.2

def f_2(x):
    return 0.9 * x - 0.2

def f_3(x):
    return (1+r) * x

f = [f_0,f_1,f_2,f_3]

x_t = [0.1]
n = [1] * H
p_etoile = np.random.normal(size = t_max +2)

for t in range(2,t_max + 1):

    resultat_partiel = 0

    for h in range(1,H+1):
        renormalisation = 0
        for j in range(1,H+1):
            renormalisation += mp.exp(beta * U[j-1])

        resultat_partiel += mp.exp(beta * U[h-1]) * (f[h-1](x_t[-1]) + p_etoile[t]) / ((1 + r) * renormalisation
)

    x_t.append(resultat_partiel)

    for h in range(1,H+1):
        U[h-1] = lambd * U[h-1] + (1-lambd) * (x_t[-1] - (1+r) * x_t[-2] + p_etoile[t] - (1+r)*p_etoile[t-1]) *
(p_etoile[t] - p_etoile[t-1] + f[h-1](x_t[-2]))

figure = plt.figure(figsize=(20,3))
ax1 = figure.add_subplot(121)
ax1.set_title("x(t) en fonction de t")
ax1.plot(range(t_max),x_t)

ax2 = figure.add_subplot(122)
ax2.set_title("x(t) en fonction de x(t-1)")
ax2.plot([x_t[i] for i in range(0,t_max-1)],[x_t[i] for i in range(1, t_max)])
```

Out[8]: `[<matplotlib.lines.Line2D at 0x115eba438>]`

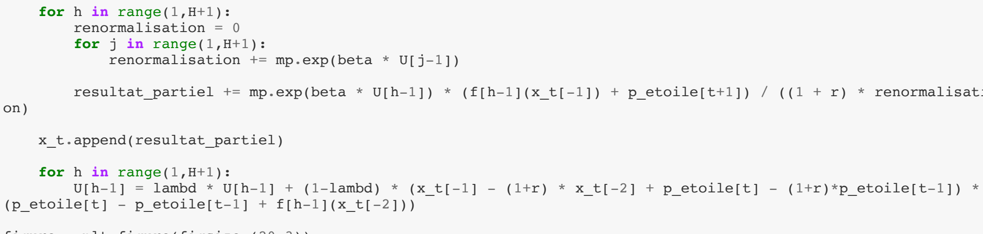


c) Revenons à $\beta = 10$ et diminuons λ

Tracé de $x(t)$ en fonction de t et de $x(t)$ en fonction de $x(t-1)$

In [9]:
```
H = 3
U = [0.5] * H
beta = 10
g = 3
lambd = 0.5

def f_0(x):
    return 0

def f_1(x):
    return 0.9 * x + 0.2

def f_2(x):
    return 0.9 * x - 0.2

def f_3(x):
    return (1+r) * x

f = [f_0,f_1,f_2,f_3]

x_t = [0.1]
n = [1] * H
p_etoile = np.random.normal(size = t_max +2)

for t in range(2,t_max + 1):

    resultat_partiel = 0

    for h in range(1,H+1):
        renormalisation = 0
        for j in range(1,H+1):
            renormalisation += mp.exp(beta * U[j-1])

        resultat_partiel += mp.exp(beta * U[h-1]) * (f[h-1](x_t[-1]) + p_etoile[t+1]) / ((1 + r) * renormalisati
on)

    x_t.append(resultat_partiel)

    for h in range(1,H+1):
        U[h-1] = lambd * U[h-1] + (1-lambd) * (x_t[-1] - (1+r) * x_t[-2] + p_etoile[t] - (1+r)*p_etoile[t-1]) *
(p_etoile[t] - p_etoile[t-1] + f[h-1](x_t[-2]))

figure = plt.figure(figsize=(20,3))
ax1 = figure.add_subplot(121)
ax1.set_title("x(t) en fonction de t")
ax1.plot(range(t_max),x_t)

ax2 = figure.add_subplot(122)
ax2.set_title("x(t) en fonction de x(t-1)")
ax2.plot([x_t[i] for i in range(0,t_max-1)],[x_t[i] for i in range(1, t_max)])
```

Out[9]: `[<matplotlib.lines.Line2D at 0x115f68be0>]`

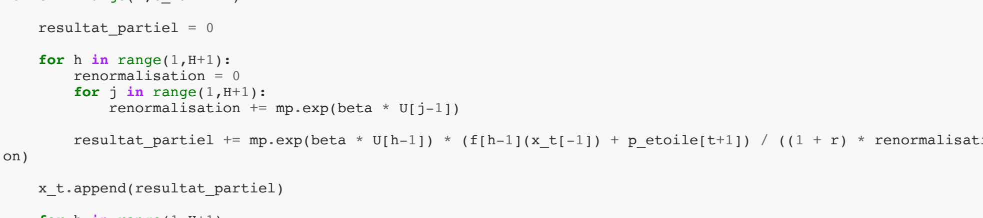


d) Diminuons encore plus λ

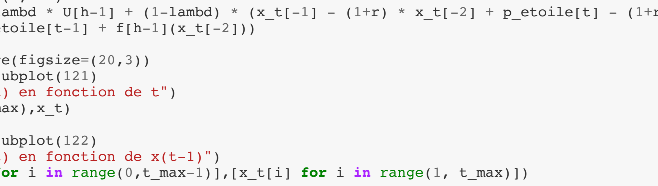
Tracé de $x(t)$ en fonction de t et de $x(t)$ en fonction de $x(t-1)$

In [10]:
```
H = 3
U = [0.5] * H
beta = 10
g = 3
lambd = 0.2

def f_0(x):
    return 0

def f_1(x):
    return 0.9 * x + 0.2

def f_2(x):
    return 0.9 * x - 0.2

def f_3(x):
    return (1+r) * x

f = [f_0,f_1,f_2,f_3]

x_t = [0.1]
n = [1] * H
p_etoile = np.random.normal(size = t_max +2)

for t in range(2,t_max + 1):

    resultat_partiel = 0

    for h in range(1,H+1):
        renormalisation = 0
        for j in range(1,H+1):
            renormalisation += mp.exp(beta * U[j-1])

        resultat_partiel += mp.exp(beta * U[h-1]) * (f[h-1](x_t[-1]) + p_etoile[t+1]) / ((1 + r) * renormalisati
on)

    x_t.append(resultat_partiel)

    for h in range(1,H+1):
        U[h-1] = lambd * U[h-1] + (1-lambd) * (x_t[-1] - (1+r) * x_t[-2] + p_etoile[t] - (1+r)*p_etoile[t-1]) *
(p_etoile[t] - p_etoile[t-1] + f[h-1](x_t[-2]))

figure = plt.figure(figsize=(20,3))
ax1 = figure.add_subplot(121)
ax1.set_title("x(t) en fonction de t")
ax1.plot(range(t_max),x_t)

ax2 = figure.add_subplot(122)
ax2.set_title("x(t) en fonction de x(t-1)")
ax2.plot([x_t[i] for i in range(0,t_max-1)],[x_t[i] for i in range(1, t_max)])
```

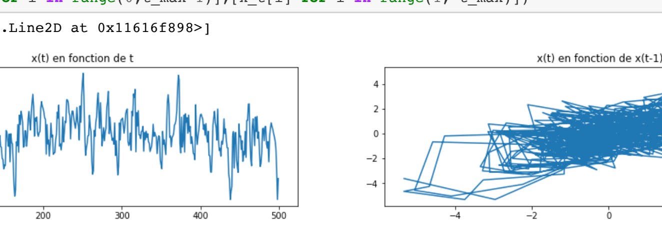
Out[10]: `[<matplotlib.lines.Line2D at 0x11616f898>]`



Résultats :
$\beta = 10$ et $\lambda = 0.9$



$\beta = 10$ et $\lambda = 0.5$



$\beta = 10$ et $\lambda = 0.2$

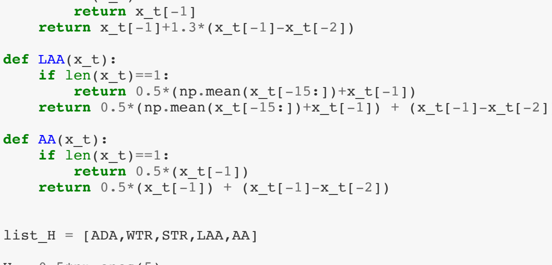
La dépendance en λ intervient dans la récompense de l'influence qu'a le passé dans les décisions prises dans le cadre d'une stratégie donnée. Il est difficile de comparer ces graphes mais on devrait pouvoir discerner pour un λ proche de zéro une quasi-indépendance de la stratégie présente par rapport à son passé (corrélation entre $x(t)$ et $x(t-1)$ quasiment nulle).

3) Stratégies traditionnelles : Stratégies empiriques

$$ADA : x^e_{1,t+1} = 0.65x_{t-1} + 0.35x_{1,t}$$
$$WTR : x^e_{2,t+1} = x_{t-1} + 0.4(x_{t-1} - x_{t-2})$$
$$STR : x^e_{3,t+1} = x_{t-1} + 1.3(x_{t-1} - x_{t-2})$$
$$LAA_l : x^e_{4,t+1} = 0.5(\frac{1}{l}\sum_{i=1}^{l} x_{t-i} + x_{t-1}) + (x_{t-1} - x_{t-2})$$
$$AA : x^e_{4,t+1} = 0.5x_{t-1} + (x_{t-1} - x_{t-2})$$

In [11]:
```
H = 5
beta = 10
lambd = 0.5

list_ADA=[]

def ADA(x_t):
    if len(x_t)==1:
        list_ADA.append(x_t[0])
        return list_ADA[-1]
    list_ADA.append(0.65*x_t[-1] + 0.35*list_ADA[-1])
    return list_ADA[-1]

def WTR(x_t):
    if len(x_t)==1:
        return x_t[-1]
    return x_t[-1]+0.4*(x_t[-1]-x_t[-2])

def STR(x_t):
    if len(x_t)==1:
        return x_t[-1]
    return x_t[-1]+1.3*(x_t[-1]-x_t[-2])

def LAA(x_t):
    if len(x_t)==1:
        return 0.5*(np.mean(x_t[-15:])+x_t[-1])
    return 0.5*(np.mean(x_t[-15:])+x_t[-1]) + (x_t[-1]-x_t[-2])

def AA(x_t):
    if len(x_t)==1:
        return 0.5*(x_t[-1])
    return 0.5*(x_t[-1]) + (x_t[-1]-x_t[-2])


list_H = [ADA,WTR,STR,LAA,AA]

U = 0.5*np.ones(5)
n = [1]*H
z = [0]*5
x_t = [0.5]

for t in range(1,t_max):
    n = np.exp(beta*U)
    n /= np.sum(n)
    z = np.array([f(x_t) for f in list_H])
    x_t.append(1/(1+r) * np.sum(n*z))
    U = U*lambd + (1-lambd)*(x_t[t]-(1+r)*x_t[t-1])*(np.array([f(x_t) for f in list_H])-r*x_t[t-1])

figure = plt.figure(figsize=(20,3))
ax1 = figure.add_subplot(121)
ax1.set_title("x(t) en fonction de x(t-1)")
ax1.plot(x_t[1:len(x_t)],x_t[0:len(x_t)-1])
```

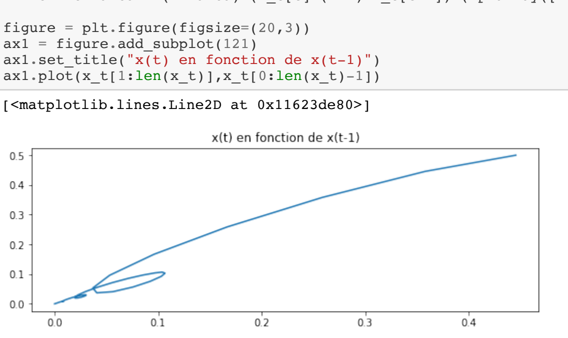
Out[11]: `[<matplotlib.lines.Line2D at 0x11623de80>]`



Interprétation :

In [12]:
```
figure = plt.figure(figsize=(20,3))
ax1 = figure.add_subplot(121)
ax1.set_title("x(t) en fonction de x(t-1)")
ax1.plot(x_t[1:len(x_t)],x_t[0:len(x_t)-1],label="x(t) en fonction de x(t-1)")
ax1.plot(np.linspace(0,0.5,1000),np.linspace(0,0.5,1000),label="fonction identité")
ax1.plot(np.linspace(0,0.5,1000),[0.1]*1000,label="fonction constante à 0.1")
plt.legend()
```

Out[12]: `<matplotlib.legend.Legend at 0x11697ee48>`



Pour étudier la stabilité de ce modèle, intéressons-nous à l'image d'une valeur de $x(t-1)$. Remarquons que pour $x(t-1)$ strictement supérieur à 0.1, son image $x(t)$ se retrouve au dessus de la première bissectrice. La valeur de x va donc augmenter au fur et à mesure jusqu'à atteindre un éventuel point fixe ou continuer à augmenter indéfiniment. Cependant, si $x(t-1)$ est inférieur à 0.1, le modèle borne $x(t)$ à des valeurs entre 0 et 0.1, nous pouvons déduire de ces variations raisonnables que le modèle est stable dans ce domaine.