CodesMonteCarlo

June 6, 2024

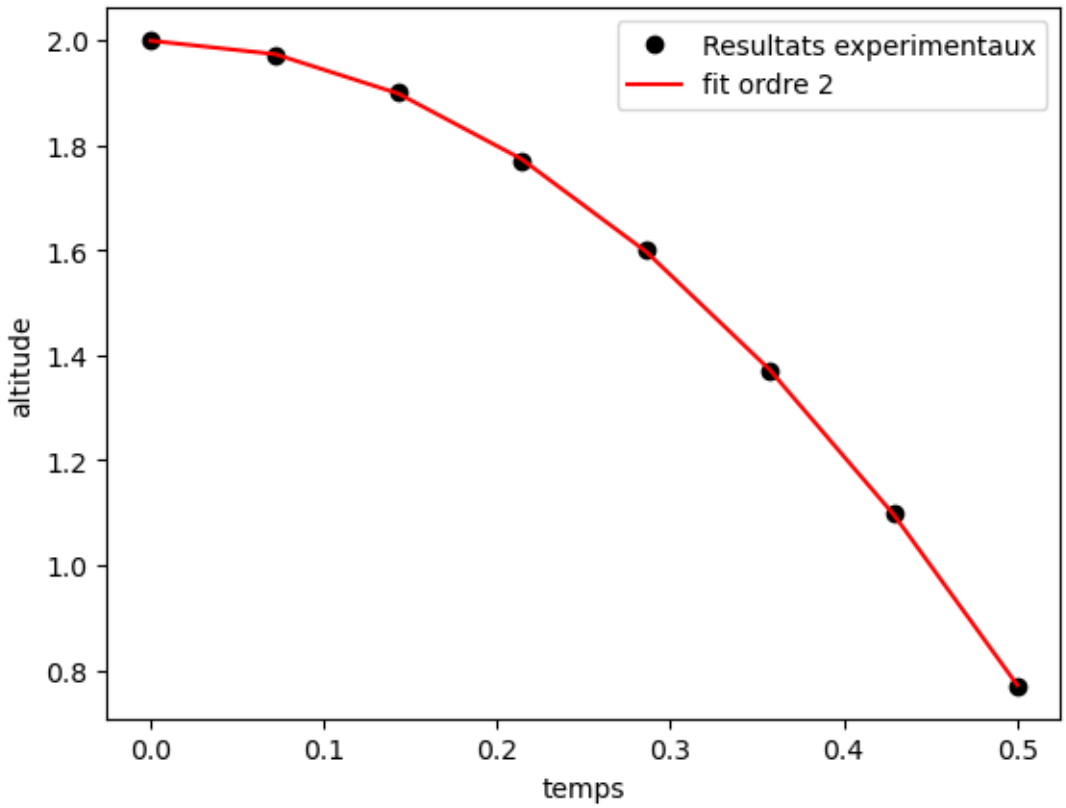
[5]:
```
#Volume equivalent chimie (dosage de a par b)
import numpy as np
N = 100000  #nombre d'expériences simulées
L = []      #liste pour contenir les résultats

for i in range(N) :
    Cb = np.random.uniform(0.099,0.101) #0.1 \pm 0.001
    Va = np.random.uniform(9.9,10.1)
    Veq = np.random.triangular(9.6,9.7,9.8)
    Ca = Cb*Veq/Va
    L.append(Ca)

ECa = np.mean(L)          #calcul de la valeur moyenne statistique (espérance)
uCa = np.std(L,ddof = 1) # incertitude-type

print("concentration Ca :", ECa)
print("incertitude-type :", uCa)
```

```
concentration Ca : 0.09700458661575034
incertitude-type : 0.0008923620553163799
```

[10]:
```
#Utilisation de polyfit, chute libre
import numpy as np
import matplotlib.pyplot as plt

T = np.array([0,72e-3,143e-3,214e-3,286e-3,357e-3,429e-3,500e-3])
Z = np.array([2.00,1.97,1.90,1.77,1.60,1.37,1.10,0.77])

p = np.polyfit(T,Z,2) # modélisation par polynôme degré 2
print('a=',p[0])
print('b=',p[1])
print('c=',p[2])
#print("g = ",-2*p[0])

Zmodele = p[0]*(T**2) + p[1]*T + p[2] # polynôme modèle de degré 2

# Représentation graphique des résultats
plt.figure()
```

```
plt.clf()
plt.plot(T,Z,"ko",label='Resultats experimentaux')  # résultats expérimentaux :␣
 ↪points noirs
plt.plot(T,Zmodele, "red", label='fit ordre 2') # polynôme modèle en rouge
plt.xlabel("temps")
plt.ylabel("altitude")
plt.legend()
plt.show()
```

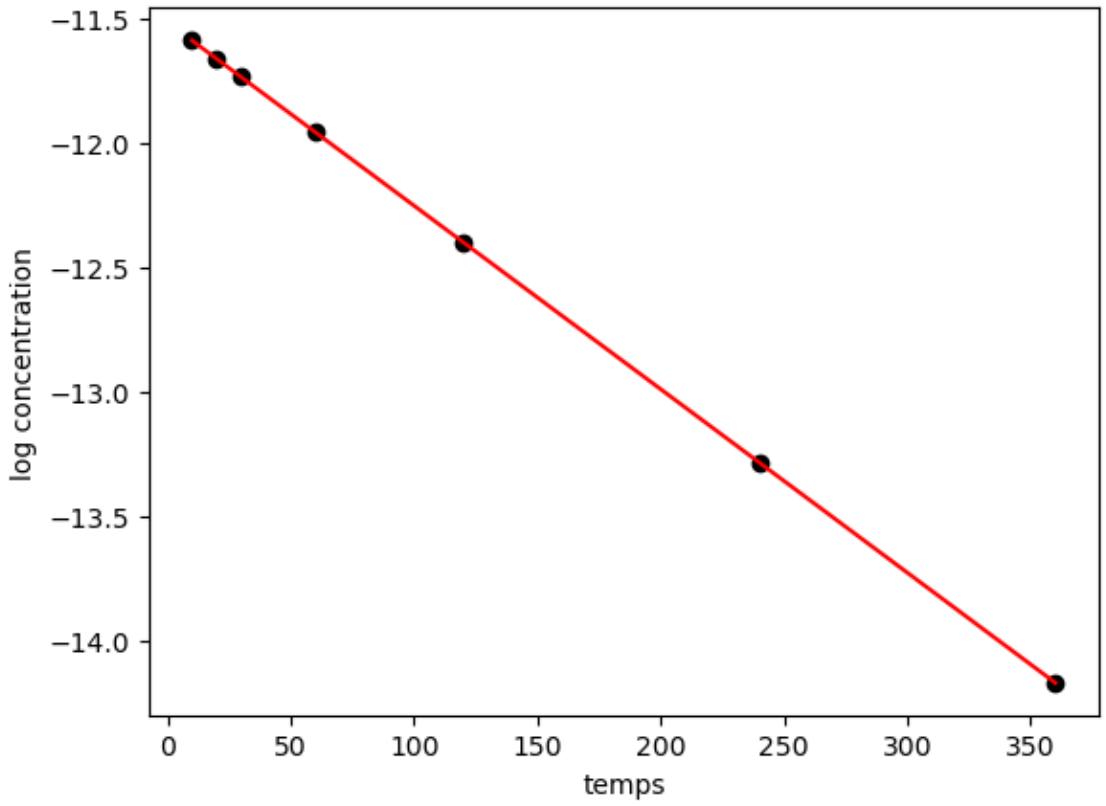
```
a= -4.897260522121453
b= -0.0036411738305963974
c= 1.998428942862882
```



[16]:
```
#loi vitesse reaction ordre 1
import numpy as np
import matplotlib.pyplot as plt

T = np.array([10,20,30,60,120,240,360])
C = np.array([9.3e-6,8.6e-6,8.0e-6,6.4e-6,4.1e-6,1.7e-6,0.70e-6])
lnC = np.log(C) # logarithme des concentrations
```

```
p = np.polyfit(T,lnC,1) # polynôme modèle de degré 1
print('a=',p[0])
print('b=',p[1])
print("kapp = ",-p[0])

lnCsim = p[0]*T + p[1] # droite modèle

# Représentation graphique des résultats
plt.figure()
plt.plot(T,lnC,"ko")  # résultats expérimentaux : points noirs
plt.plot(T,lnCsim, "red") # droite modèle en rouge
plt.xlabel("temps")
plt.ylabel("log concentration")
plt.show()
```

```
a= -0.007381210008831907
b= -11.515128726004752
kapp =  0.007381210008831907
```



[ ]:
```
#Monte Carlo pour une chute libre
import numpy as np
```

```
T = np.array([0,72e-3,143e-3,214e-3,286e-3,357e-3,429e-3,500e-3])
Z = np.array([2.00,1.97,1.90,1.77,1.60,1.37,1.10,0.77])

Delta_t = 1e-3 # précision sur le temps
Delta_z = 2e-3 # précision sur l'altitude

N = 10000 # nombre de points de mesure: len(T) == len(Z)
L = []
nbExp = len(T)

for i in range(N) :
    Tsim = T + Delta_t * np.random.uniform(-1,+1,size = nbExp)
    Zsim = Z + Delta_z * np.random.uniform(-1,+1,size = nbExp)
    p = np.polyfit(Tsim,Zsim,2)
    L.append(-2*p[0]) # stocker la valeur de g

Eg = np.mean(L)
ug = np.std(L, ddof = 1) #ddof Means Delta Degrees of Freedom. The divisor used
 ↪in calculations is N - ddof, where N represents the number of elements.
print("accélération pesanteur : ",Eg)
print("incertitude type : ",ug)
```