PROGRAMME 1 :

Equation de titrage : 2 MnO_4^-(aq) + 5 H_2O_2(aq) + 6 H^+(aq) = 2 Mn^{2+}(aq) + 5 O_2(g) + 8 H_2O(l)

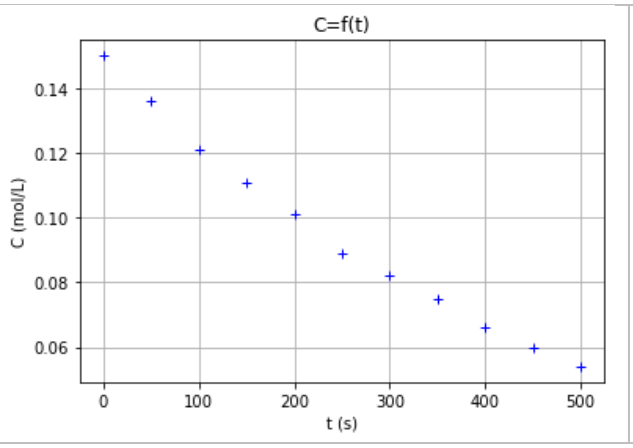
- Relation à l'équivalence : $\frac{n(MnO_4^-)_{eq}}{2} = \frac{n(H_2O_2)_{titré}}{5}$

💣💣💣attention aux coeff stœchiométriques

Soit $\frac{Cper \times Veq}{2} = \frac{C \times V0}{5} \Rightarrow \boxed{C = \frac{5 \times Cper \times Veq}{2 \times V0}}$

```
# 3- Calcul de C
C=[C0]
for i in range(len(Veq)) :
    C.append(5*Cper*Veq[i]/(2*V0))
```

- On trace C=f(t) ⇒ ce n'est pas une droite (la cinétique n'est pas d'ordre 0 !)



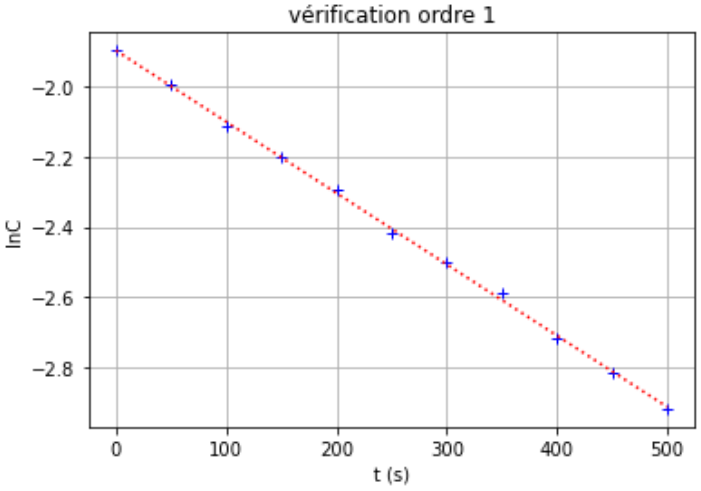
Valeurs de C

```
[0.13599999999999998,
 0.121,
 0.111,
 0.10099999999999999,
 0.089,
 0.08199999999999999,
 0.075,
 0.066,
 0.06,
 0.054000000000000006]
```

- On trace lnC=f(t) puisque $v = -\frac{d[H_2O_2]}{dt} = k[H_2O_2]$ soit $= -\frac{dC}{dt} = kC \Rightarrow lnC = lnC_0$-k×t

Pour cela : on calcule lnC (attention il faut faire une boucle si on manipule des listes)

```
#3- Calcul de lnC puisqu'on suppose un ordre 1
lnC=[]
for i in range(len(C)):
    lnC.append(np.log(C[i]))
```



C'est une droite !

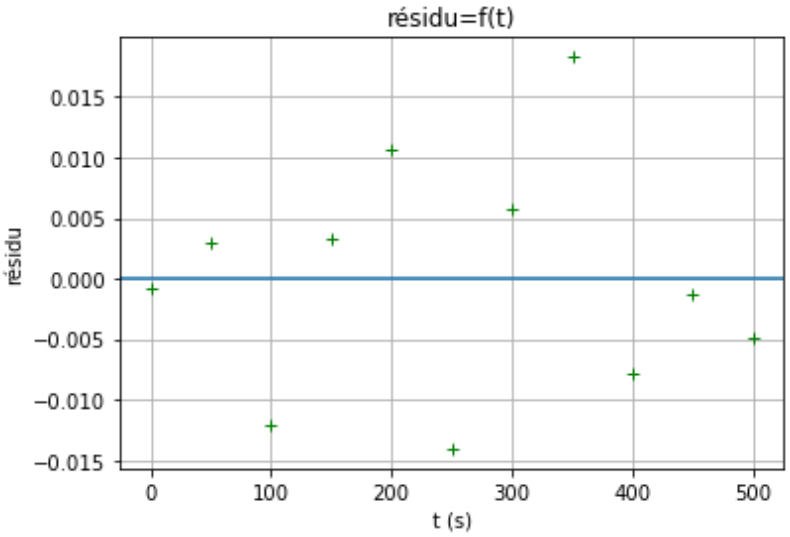
⇒ on fait la régression linéaire et on demande au programme de nous afficher k avec son unité (et si possible un nombre de CS raisonnable)

```
k=2.03e-03 s-1
```

⇒ on calcule les résidus : différence entre la valeur expérimentale et celle attendue par la régression linéaire.

```
# 3- Calcul de la regression linéaire attendue et exploitation
reglin=np.polyfit(t,lnC,1)
k=-reglin[0]
lnC_theo=np.polyval(reglin,t) #calcule les valeurs obtenues par
residu=[]
for i in range(len(lnC)):
    residu.append(lnC[i]-lnC_theo[i]) #cad la différence entre l

# 4- affichage de k
print('k={:.2e} s-1'. format(k))
```

On trace les résidus :



SI les résidus sont bien répartis autour de la valeur 0 (l'axe des abscisses) c'est bon signe !
mais il reste **à calculer les incertitudes**

PROGRAMME 2

On définit bien

- les demi-étendues : pour des distributions rectangulaires (uniforme en python), comme les tolérance des verreries
- les incertitudes (ou écrat type) pour les distributions gaussienne (normale en python) comme celles définies avec un %tage d'erreur.
- Le nombre d'expériences virtuelles que l'ont compte faire N (prendre N≥ 10000)

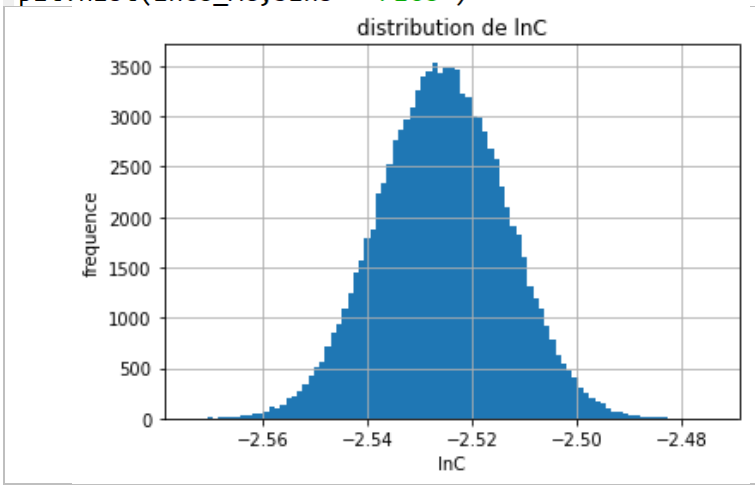
```
D_V0=0.1 #mL
D_Veq=0.05 #mL
u_Cper=0.01*Cper #mol/L
VVeq= 8 #mL pour le calcul d
N=100000
```

On génère les N expériences qui vont conduire à une liste de N valeurs de Cper, V0 et Veq pour cela on utilise le module random. Puis on calcule les N valeur de C et le N valeur de lnC

```
Cper_MC=rd.normal(Cper, u_Cper,N)
Veq8_MC=Veq8+D_Veq*rd.uniform(-1,1,N)
V0_MC=V0+D_V0*rd.uniform(-1,1,N)

C8_MC=5*Cper_MC*Veq8_MC/(2*V0_MC)
lnC8_MC=np.log(C8_MC)
```

Il reste à faire le calcul de l'écart type qui correspond à l'incertitude et à demander son affichage :

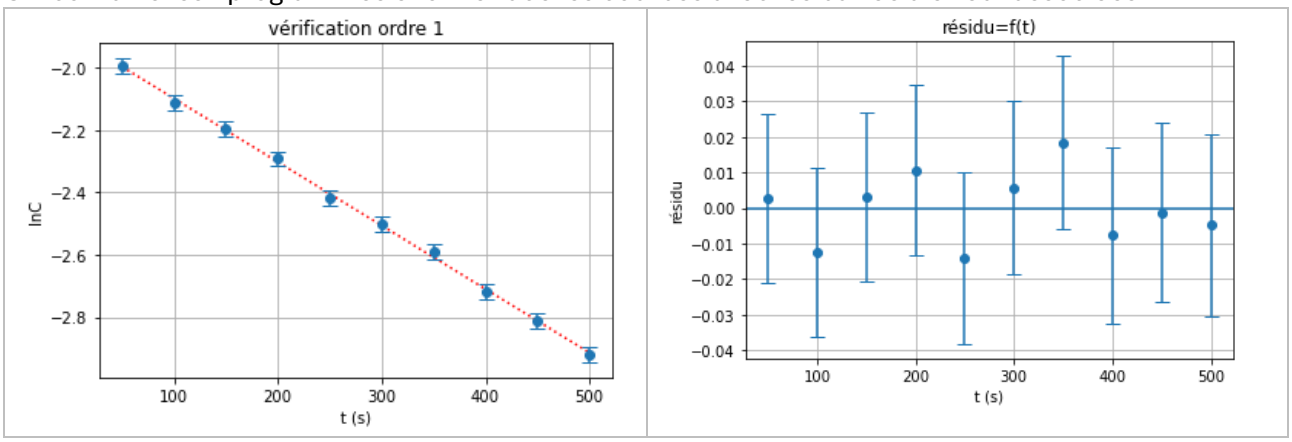
```
#3- calcul de l'incertitude
u_lnC8=np.std(lnC8_MC,ddof=1)

#4- affichage de k
print('u_lnC8={:.1e} s-1'. format(u_lnC8))

# 4-tracé de l'histogramme
plt.hist(lnC8_MC,bins= 'rice')
```




```
u_lnC8=1.2e-02
```

PROGRAMME 3

ON combine les 2programmes et on retrace les courbes avec les barres d'erreur associées :



Avec les barres d'erreur on voit que la modélisation est incluse dans la valeur expérimentale $\pm$ u.
La modélisation est donc cohérente.

(Valeurs de u_lnC :

```
[0.011797328075126352,
 0.011842114791541606,
 0.011873501491402183,
 0.011948280802892829,
 0.012037901839850389,
 0.012107959806237245,
 0.012204396849529415,
 0.01237543116560074,
 0.012562382974591908,
 0.01276714429048911]
```

)