TP_5_correction_commentee

March 27, 2020

1 Calcul scientifique -- TP5 : Interpolation, ajustement

Une correction avec quelques explications

L'intensité de la radiation d'une substance radioactive se mesure au moyen de la 'demi-vie' b de la substance.

On sait que la décroissance de la radioactivité suit la loi :

$$\gamma(t) = ae^{-bt}$$

et on a mesuré les radiations suivantes :

t (années)	0	0.5	1	1.5	2	2.5
$\gamma(t)$	1.000	0.994	0.990	0.985	0.979	0.977

t (années)	3	3.5	4	4.5	5	5.5
$\gamma(t)$	0.972	0.969	0.967	0.960	0.956	0.952

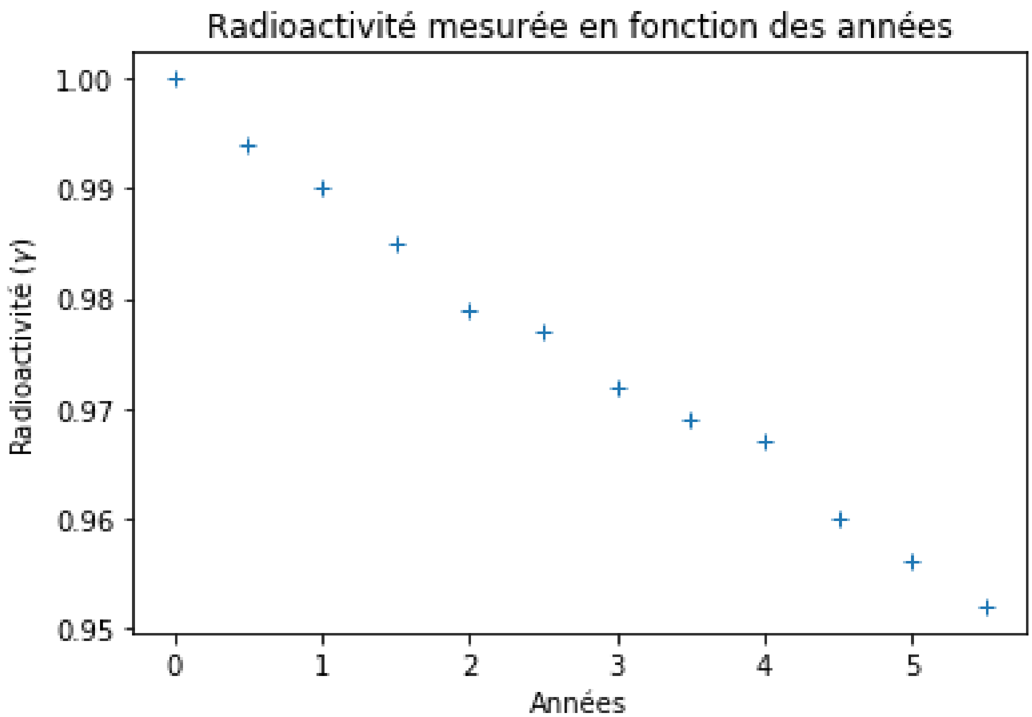
L'exercice vise à estimer les valeurs de a et b ou de trouver une fonction qui permet de déterminer $\gamma(t)$.

1.0.1 Tracé des données

Tracer l'évolution des mesures γ en fonction de t comme sur le shéma ci-dessous

In [1]:
```
import numpy as np
import scipy as sp
import scipy.optimize
import matplotlib.pyplot as plt
%matplotlib inline

tab = np.array([0,0.5,1,1.5,2,2.5,3,3.5,4,4.5,5,5.5])
gtab = np.array([1.000,0.994,0.990,0.985,0.979,0.977, 0.972,0.969,0.967,0.960,0.956,0.95

plt.plot(tab,gtab,'+')
plt.xlabel('Années')
plt.ylabel('Radioactivité ($\gamma$)')
```

```
plt.title('Radioactivité mesurée en fonction des années')
plt.show()
```



Nous allons désormais étudier des méthodes permettant de trouver les valeurs "optimales" de a et b. Par optimales, nous voulons désigner les valeurs qui minimisent la somme des erreurs quadratiques avec les données (c.f. cours).

1.0.2 Première méthode.

Nous allons calculer la somme des erreurs au carré en fonction de a et b :

$$S(a,b) = \sum_t (\gamma_t - ae^{-bt})^2,$$

et rechercher des valeurs minimales de cette erreur.

Cette erreur sera minimale pour les valeurs de a et b pour lesquelles $\frac{\partial S(a,b)}{\partial a} = 0$ et $\frac{\partial S(a,b)}{\partial b} = 0$.

Commencez par définir avec `sympy` l'expression correspondant à

$$(\gamma_t - ae^{-bt})^2$$

puis en déduire grâce à la dérivation de sympy les deux expressions suivantes :

$$f_1(a,b) = \frac{\partial\,(\gamma_t - ae^{-bt})^2}{\partial a},$$

et

$$f_2(a,b) = \frac{\partial\,(\gamma_t - ae^{-bt})^2}{\partial b}.$$

Convertissez ensuite ces deux expressions en fonctions `python` en utilisant sympy.lambdify.
On obtient deux fonctions f_1 et f_2 qui ont chacune 4 paramètres: a, b, t, g

```
In [2]: import  sympy

        #définition des symboles (il est conseillé de ne pas leur donner le même nom que les var
        t_s,a_s,b_s,gamma_s = sympy.symbols('t,a,b,gamma')

        # définition de l'expression
        err_s = (a_s*sympy.exp(-b_s*t_s)-gamma_s)**2

        # calcul des dérivées partielles
        d1=sympy.diff(err_s,a_s)
        d2=sympy.diff(err_s,a_s)

        print("err(a,b)= ", err_s)
        print("d err(a,b)/da = ",d1)
        print("d err(a,b)/db = ",d2)
```

```
err(a,b)=  (a*exp(-b*t) - gamma)**2
d err(a,b)/da =  2*(a*exp(-b*t) - gamma)*exp(-b*t)
d err(a,b)/db =  2*(a*exp(-b*t) - gamma)*exp(-b*t)
```

En utilisant les fonctions de la question précédente et *np.sum*, calculer les dérivées partielles de S par rapport à a et b.
Construire ensuite une fonction F qui retourne

$$F(a, b) = \left[\frac{\partial S(a,b)}{\partial a}, \frac{\partial S(a,b)}{\partial b}\right].$$

Pour cette fonction on prendra t=tab et g=gtab. On la codera avec un seul argument correspondant au tableau numpy $[a, b]$. Pour vérifier que cette fonction donne les bons résultats, calculer $F([0, 0])$.

```
In [3]: # on transforme les expressions sympy en fonctions python
        f1 = sympy.lambdify([a_s,b_s,t_s,gamma_s],d1)
        f2 = sympy.lambdify([a_s,b_s,t_s,gamma_s],d2)

        # on peut ensuite calculer les deux sommes qui constituent la fonction F
        def da(a,b,t,y):
            return np.sum(f1(a,b,t,y))

        def db(a,b,t,y):
            return np.sum(f2(a,b,t,y))

        # attention F doit être une fonction avec un seul paramètre qui est un tableau (ici p) c
        def F(p):
            return [da(p[0],p[1],tab,gtab),db(p[0],p[1],tab,gtab)]
```

```
print("F([0,0])=",F([0,0]))


# remarque: pour ceux qui ne sont pas arriver à utiliser lambdify on peut "réecrire" les
# ce qui est moins bien mais fonctionne

#def da1(a,b,t,y):
#   return np.sum(2*(a*np.exp(-b*t) - y)*np.exp(-b*t))


# def db1(a,b,t,y):
 #   return np.sum(-2*a*t*(a*np.exp(-b*t) - y)*np.exp(-b*t))
```

```
F([0,0])= [-23.401999999999997, -23.401999999999997]
```

Au moyen de la fonction `scipy.optimize.fsolve` vue en TP, estimez les valeurs de a et b annulant la dérivée.

Les afficher (on prendra comme tablau initial p0= np.array([1 , 0]).

```
In [4]: # defintion du tableau des conditions initiales:

        p0= np.array([ 1 , 0])

        # solve
        sol=scipy.optimize.fsolve(F,p0)

        # on récupère les valeurs de a et b
        print("[a,b]=",sol)
        a,b=sol
```

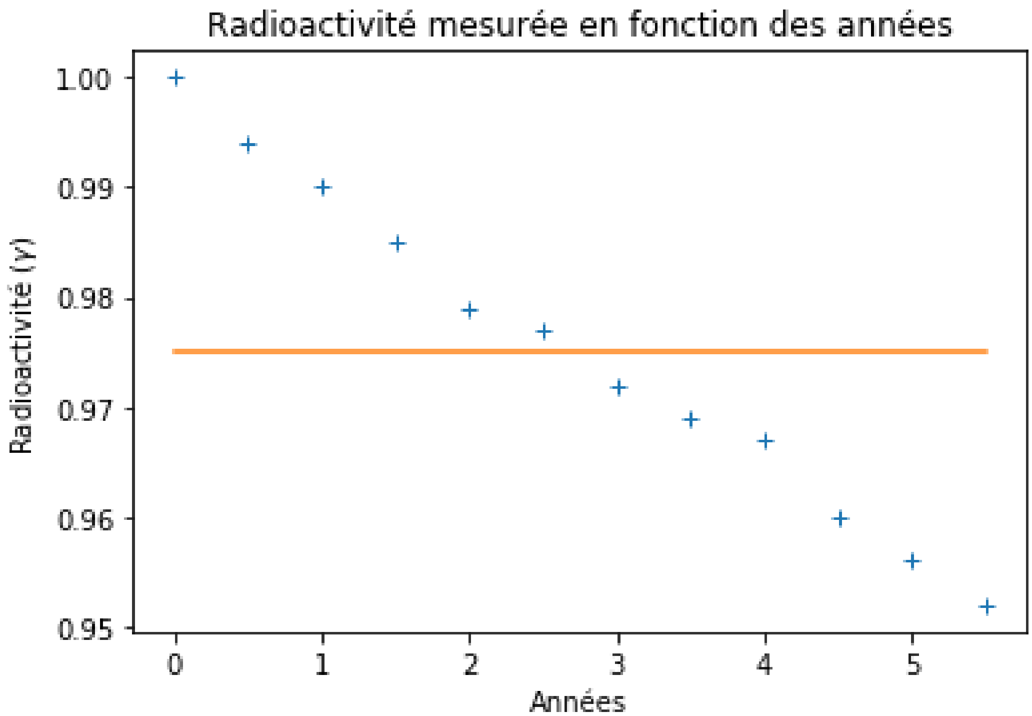
```
[a,b]= [ 9.75083333e-01 -1.42108547e-14]
```

1.0.3 Vérification graphique

Tracer la courbe $\gamma(t) = ae^{-bt}$, avec les valeurs estimées de a b en superposant cette courbe sur les points mesurés.

```
In [5]: # pour le dessin on définit f (on pourrait aussi utiliser subs avec l'expression sympy e

        def f(t,a,b) :
            return a*np.exp(-b*t)

        plt.plot(tab,gtab,'+')
        plt.xlabel('Années')
        plt.ylabel('Radioactivité ($\gamma$)')
        plt.title('Radioactivité mesurée en fonction des années')
```

```
plt.plot(tab,f(tab,a,b))
plt.show()
```



1.0.4 Seconde méthode.

Nous allons désormais utiliser la fonction `scipy.optimize.curve_fit` qui permet d'estimer directement les paramètres a et b minimisant, au sens des moindres carrés, le modèle avec les mesures. Affichez les valeurs estimées et comparez-les avec les précédentes.

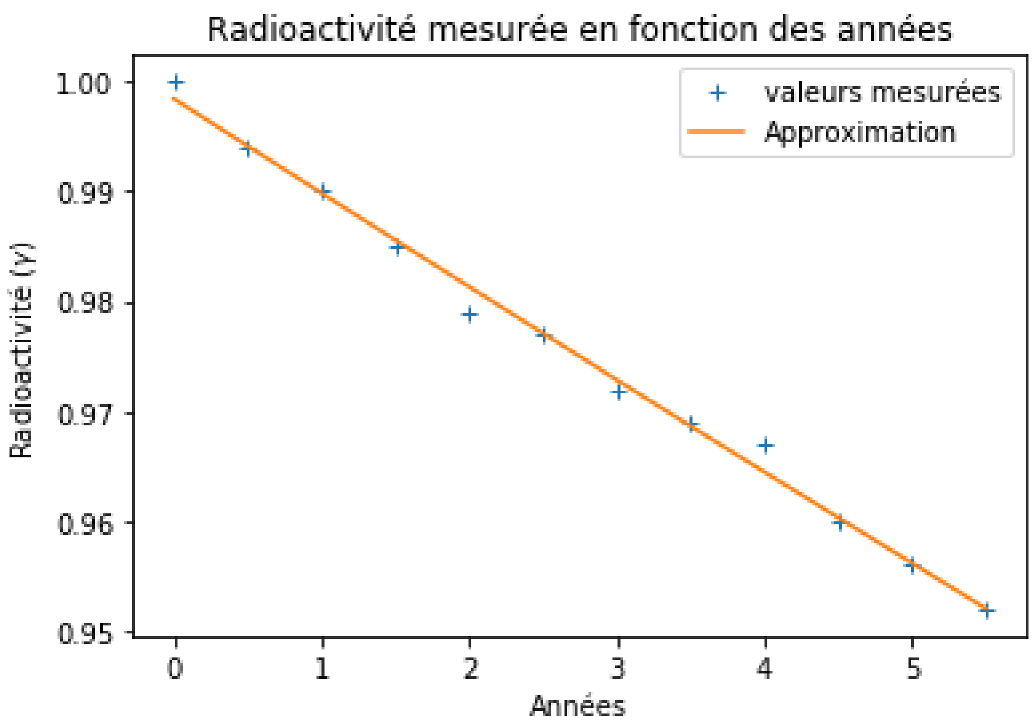
```
In [6]: #Il faut définir une fonction qui a la forme de ce que l'on recherche
        # la variable qui n'est pas dans les paramètres qu'on cherche doit être en premier argum
        def func (t,a,b):
            return a*np.exp(-b*t)

        #  valeurs initiales :
        a0,b0 = (0.5,0.5)

        # sp.optimize.curve_fit  renvoie un tuple dont le premier élément est le tuple des param

        coeffs, fiterr = sp.optimize.curve_fit(func, tab, gtab, (a0,b0))
        print("[a,b]=",coeffs)
```

```
[a,b]= [0.99843145 0.00864506]
```

1.0.5 Vérifications

A nouveau, tracer la fonction donnant l'évolution de la radioactivité avec les paramètres obtenus, en la superposant aux valeurs mesurées.

```
In [7]: a1=coeffs[0]
        b1=coeffs[1]

        plt.plot(tab,gtab,'+',label='valeurs mesurées')
        plt.plot(tab,func(tab,a1,b1),label='Approximation')
        plt.xlabel('Années')
        plt.ylabel('Radioactivité ($\gamma$)')
        plt.title('Radioactivité mesurée en fonction des années')
        plt.legend()
        plt.show()
```



Calculer les résidus r_i, leur somme et leur moyenne et l'écart-type (entre mesures et valeurs estimées)

```
In [8]: r = gtab - func(tab,coeffs[0],coeffs[1])
        print("Résidus :",r)
        s=np.sum(r)
        print("Somme des résidus", s)
        m=np.mean(r)
        print("Moyenne des résidus : ",m)
```

```
s=(r-m).std()
print("Ecart type : ",s)
```

```
Résidus : [ 1.56854604e-03 -1.25016373e-04  1.62846672e-04 -5.67784707e-04
 -2.31683074e-03 -8.42120002e-05 -8.69849405e-04  3.26335789e-04
  2.50442198e-03 -3.35512754e-04 -1.93390699e-04 -6.91344575e-05]
Somme des résidus 4.1935400529879985e-07
Moyenne des résidus :  3.494616710823332e-08
Ecart type :  0.0011361364161152849
```

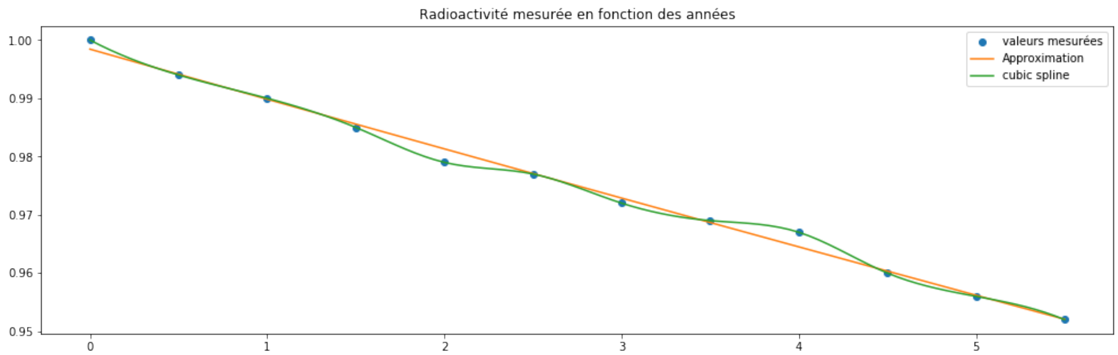
1.1 interpolation

On cherche maintenant à définir une fonction (dont on n'aura pas l'expression) qui permettra d'interpoler (de trouver une valeur approchée) des points situés "entre les mesures". Pour cela on utilisera scipy.interpolate.CubicSpline

Faire ensuite un shéma avec les valeurs mesurées, la fonction déterminer avec ajustement et celle avec interpolation.

```
In [9]: import scipy.interpolate

        # il suffit d'utiliser interpolate.CubicSpline avec les deux tableaux qui correspondent

        cs =  scipy.interpolate.CubicSpline(tab, gtab)

        # cs est alors une fonction
```

```
In [13]: fi,ax=plt.subplots(figsize=(17,5))

         # pour afficher la courbe du cubicspline , il faut prendre suffisamment de points
         #si on travaille avec tab on aura une ligne brisée
         val=np.linspace(0,5.5,100)

         ax.plot(tab,gtab,'o',label='valeurs mesurées')
         ax.plot(val,func(val,coeffs[0],coeffs[1]),label='Approximation')
         ax.plot(val,cs(val), label="cubic spline")
         #ax.xlabel('Années')
         #ax.ylabel('Radioactivité ($\gamma$)')
         ax.set_title('Radioactivité mesurée en fonction des années')
         ax.legend()
         print()
```


Radioactivité mesurée en fonction des années
valeurs mesurées
Approximation
cubic spline
1.00
0.99
0.98
0.97
0.96
0.95
0
1
2
3
4
5

```
In [ ]:
```