CCINP 2021 — PC Modélisation

Mercredi 5 mai 2021

Partie II

Question 12 :

a) Ici on demande de calculer analytiquement l'intégrale.

$$\int_{T_1}^{T_2} f(T)\mathrm{d}T = \int_{T_1}^{T_2} \left(A_1 + \frac{A_2}{T+A_3}\right)\mathrm{d}T = A_1\left(T_2 - T_1\right) + A_2 \cdot \ln\left(\frac{T_2+A_3}{T_1+A_3}\right)$$

Ce qui donne en python :

```
A1 = 8.303 # les constantes sont fournies par l'énoncé
A2 = -2810
A3 = 485.6
def integ1(T1,T2):
   return A1*(T2-T1) + A2 * np.log((T2+A3)/(T1+A3))
```

b) Ici on demande la méthode des rectangles avec 100 pas, le pas vaut donc $\frac{T_2 - T_1}{100}$.
Les constantes A_1, A_2, A_3 définies dans le code python ci-dessus sont réutilisées dans la fonction `F_de_T_sur_T` qui calcule $\frac{f(T)}{T}$.

```
def f_de_T_sur_T(T):
  return ( A1 + A2/(T+A3) ) / T

def integ2(T1,T2):
   s = 0
   pas = (T2-T1) / 100
   for i in range(100):
      T = T1 + i*pas
      s = s + pas * f_de_T_sur_T(T)
   return s
```

Question 13 : Voici le programme complété, les **justifications** demandées sont plus bas.

```
1  import numpy as np
2  A1 = 8.303 ; A2 = -2810 ; A3 = 485.6
3  T0 = 473.15 ; m0 = 11.0
4  Pext = 1.01e5 ; Mw = 44e-3 ; Volume = 1.0 ; Text = 293
5  R = 8.3144
6  def chercheTB(T,m):
7        TB = 300
8        residu = 1   #[instruction1]
9        while residu > 1e-10 :
10           eq = integ2(T,TB) + np.log( m*R*T / (Mw*Volume*Pext) ) # [instruction2]
11           deq = f_de_T_sur_T(TB) # [instruction3]
12           TBold = TB
13           TB = TB - eq/deq
14           residu = abs(TBold - TB) # [instruction4]
15       return TB
```

Appelons $G(T_B)$ le membre gauche de l'équation 11, que l'on cherche à annuler :

$$G(T_B) = \int_{T}^{T_B} \frac{f(T)}{T}\mathrm{d}T + \ln\left(\frac{m \cdot R \cdot T}{M \cdot V \cdot P_{\text{ext}}}\right)$$

La méthode de Newton veut qu'à la ligne 13 la variable `eq` vaille $G(T_B)$ et que la variable `deq` vaille $\frac{\mathrm{d}G}{\mathrm{d}T_B} = \frac{f(T_B)}{T_B}$. C'est ce qui détermine les lignes 10 et 11.

Les lignes 8 et 14 concernent la condition d'arrêt, qui n'est pas clairement définie par l'énoncé. À la ligne 8, n'importe quelle valeur supérieure à 10^{-10} fait l'affaire, pour permettre à la boucle de démarrer. À la ligne 14 on devine, en voyant la définition de `TBold` ligne 12, que l'auteur de l'énoncé voulait prendre comme critère d'arrêt la différence entre `TBold` et `TB`.

Question 14 : Ignorons le terme de «logigramme» qui est hors-programme, et décrivons l'algorithme à utiliser en indiquant bien les procédures d'initialisation et les critères d'arrêt.

- On crée trois listes de taille 101 destinées à contenir les valeurs de T, m et P aux temps $t = 0\text{s}$; $t = 0,10\text{s}$; … ; $t = 10\text{s}$.
- On remplit les premières cases de ces trois listes avec les conditions initiales $T(0)$, $m(0)$ (données de l'énoncé) et $P(0)$ (calculé par $P = \dfrac{m \cdot R \cdot T}{M \cdot V}$)
- On répète pour i allant de 0 à 99 :
  - Calculer T_B au temps $i \cdot \Delta t$ en appliquant `chercheTB` à $T(i \cdot \Delta t)$ et $m(i \cdot \Delta t)$
  - Calculer $\dfrac{\mathrm{d}T}{\mathrm{d}t}$ et $\dfrac{\mathrm{d}m}{\mathrm{d}t}$ au temps $i \cdot \Delta t$ à l'aide des équations du système (10) de l'énoncé
  - Remplir $m((i+1) \cdot \Delta t) = m(i \cdot \Delta t) + \Delta t \cdot \dfrac{\mathrm{d}m}{\mathrm{d}t}$
  - Remplir $T((i+1) \cdot \Delta t) = T(i \cdot \Delta t) + \Delta t \cdot \dfrac{\mathrm{d}T}{\mathrm{d}t}$
  - Remplir $P((i+1) \cdot \Delta t)$ en appliquant $P = \dfrac{m \cdot R \cdot T}{M \cdot V}$
- Le critère d'arrêt de ce processus itératif est simple : on s'arrête quand on a fait les 100 itérations nécessaires pour aller de $t = 0\text{s}$ à $t = 10\text{s}$.

Question 15 : Dans le code suivant, on réutilise les variables globales définies à la question 13. Il reste à définir la section de la fuite $\Omega = 1\text{cm}^2 = 10^{-4}\text{m}^2$.
On commence aussi par définir la fonction $f(T)$ en python, on ne l'avait pas encore fait.
Le reste du code n'est pas mis à l'intérieur d'une fonction puisque l'énoncé demande un «code» et non une «fonction».

```
def f(T):
  return ( A1 + A2/(T+A3) )

Omega = 1e-4

m = [0]*101
T = [0]*101
P = [0]*101

m[0] = m0
T[0] = T0
P[0] = m[0]*R*T[0] / (Mw*Volume)

for i in range(100):
   TB = chercheTB(T[i],m[i])
   dm = - Omega*Pext*np.sqrt( 2*integ1(TB,T[i]) ) / (np.sqrt(R/Mw) * TB)
   dT = T[i] / ( m[i] * (f(T[i])-1) )  *  dm
   m[i+1] = m[i] + 0.1 * dm
   T[i+1] = T[i] + 0.1 * dT
   P[i+1] = m[i+1]*R*T[i+1] / (Mw*Volume)
```

Question 16 : «Intuitivement», la fuite s'arrêtera... quand la pression à l'intérieur du réservoir sera égale à la pression à l'extérieur ?

Partie III

Question 19 : On se sert de la fonction `np.loadtxt` décrite dans l'annexe. En observant le tableau 1 de l'énoncé on peut supposer que le délimiteur est le caractère tabulation «`\t`».

```
temp = np.loadtxt("cP.txt",delimiter="\t",usecols=[0])
CpR_exp = np.loadtxt("cP.txt",delimiter="\t",usecols=[1])
Nexp = len(temp)
```

Question 20 : On a deux raisons de mettre les écarts au carré :
- Cela produit une valeur qui est toujours positive ou nulle, même quand l'écart δ_i est négatif.
- On préfère le carré à la valeur absolue, car cela donne une fonction objectif de classe C^2, ce qui aidera la méthode «de type Newton» présentée plus tard à bien fonctionner.

Question 21 : donnons deux formulations possibles :

On se sert des opérations vectorisées pour manipuler les tableaux `temp` et `CpR` terme à terme :

```
def delta(vecA, temp, CpR):
    A1,A2,A3 = vecA
    return  (A1 +  A2/(temp+A3) )- CpR
```

Sans opérations vectorisées, on peut utiliser des boucles :

```
def delta(vecA, temp, CpR):
    A1,A2,A3 = vecA
    ecarts = np.zeros(Nexp)
    for i in range(Nexp):
       ecarts[i] = (A1 +  A2/(temp[i]+A3) )- CpR[i]
    return ecarts
```

Question 22 :

```
def fobj(vecA, temp, CpR):
    ecarts = delta(vecA, temp, CpR)
    s = 0
    for i in range(Nexp):
       s = s + ecarts[i]**2
    return s
```

Question 23 : Rappelons que $f_{\text{modèle 1}}(T) = A_1 + \dfrac{A_2}{T + A_3}$. On a alors $\begin{cases} \dfrac{\partial f_{\text{modèle 1}}}{\partial A_1} = 1 \\ \dfrac{\partial f_{\text{modèle 1}}}{\partial A_2} = \dfrac{1}{T + A_3} \\ \dfrac{\partial f_{\text{modèle 1}}}{\partial A_3} = \dfrac{-A_2}{(T + A_3)^2} \end{cases}$

Question 24 : Rappelons que $\delta_i = f_{\text{modèle 1}}(T_{\text{point n°}i}) - (C_{P,m}/R)_{\text{point n°}i}$

On a alors : $\dfrac{\partial f_{obj}}{\partial A_j} = \displaystyle\sum_{i=1}^{N_{\text{exp}}} \frac{\partial(\delta_i^2)}{\partial A_j} = \sum_{i=0}^{N_{\text{exp}}} 2\delta_i \frac{\partial \delta_i}{\partial A_j} = \sum_{i=0}^{N_{\text{exp}}} 2\delta_i \times \frac{\partial f_{\text{modèle 1}}}{\partial A_j}(T_{\text{point n°}i})$

Ce qui donne explicitement : $\begin{cases} \dfrac{\partial f_{obj}}{\partial A_1} = \displaystyle\sum_{i=1}^{N_{\text{exp}}} 2\delta_i \times 1 \\ \dfrac{\partial f_{obj}}{\partial A_2} = \displaystyle\sum_{i=1}^{N_{\text{exp}}} 2\delta_i \times \frac{1}{T_{\text{point n°}i} + A_3} \\ \dfrac{\partial f_{obj}}{\partial A_3} = \displaystyle\sum_{i=1}^{N_{\text{exp}}} 2\delta_i \times \frac{-A_2}{(T_{\text{point n°}i} + A_3)^2} \end{cases}$

Question 25 : On manipule des vecteurs `d1,d2,d3` de taille `Nexp` par des opérations vectorisées, puis on calcule les sommes ci-dessus. Ne sachant pas si l'utilisation de la fonction `sum` est tolérée par le jury, on peut la redéfinir par précaution :

```
def sum(L):
    s = 0
    for i in range(len(L)):
        s = s + L[i]
    return s

def deriv_fobj(vecA,temp, CpR_exp):
    ecarts = delta(vecA,temp,CpR_exp)
    A1,A2,A3 = vecA
    d1 = np.ones(Nexp)
    d2 = 1 / (temp + A3)
```

```
    d3 = -A2 / (temp + A3)**2
    return np.array( [ sum(2*ecarts*d1) , sum(2*ecarts*d2) , sum(2*ecarts*d3) ] )
```

Question 26 : On construit la Hessienne ligne par ligne : la ligne i contient la dérivée approchée de $\mathbf{f}'$ par rapport à A_i.

```
def ligne(i,vecA,temp,CpR_exp):
   eps = 1e-5
   vecA_apres = vecA.copy()
   vecA_apres[i] = vecA_apres[i]+eps
   return ( deriv_fobj(vecA_apres,temp,CpR_exp) -  deriv_fobj(vecA,temp,CpR_exp) ) / eps

def hessienne_fobj(vecA,temp,CpR_exp):
   L1 = ligne(0,vecA,temp,CpR_exp)
   L2 = ligne(1,vecA,temp,CpR_exp)
   L3 = ligne(2,vecA,temp,CpR_exp)
   return np.array ( [L1,L2,L3] )
```

Question 27 : On applique itérativement la règle $\mathbf{a}^{\text{itération } (k+1)} = \mathbf{a}^{\text{itération } k} - \mathbf{H}^{-1}\mathbf{f}'$, à l'aide des fonctions `np.dot` et `np.linalg.inv` fournies dans l'annexe.
Pour le critère d'arrêt, plusieurs sont possibles à partir du moment où on justifie son choix. On peut imiter ce qu'a fait l'énoncé à la question 13 et arrêter l'itération quand la distance (euclidienne) entre deux valeurs successives de $\mathbf{a}$ devient inférieure à 10^{-10}. Pour cela, on définit une fonction `norme`.

```
def norme(v):
   return np.sqrt( sum (v**2) )

vecA = np.array( [ 5.0 , -1000.0 , 500.0 ] )
distance = 1
while distance > 1e-10 :
   oldA = vecA
   fprime = deriv_fobj(vecA,temp,CpR_exp)
   H = hessienne_fobj(vecA,temp,CpR_exp)
   vecA = vecA - np.dot( np.linalg.inv(H) , fprime )
   distance = norme ( oldA - vecA )
```

(Remarque : avec le point de départ proposé par l'énoncé, la méthode diverge. Pour trouver la solution en pratique on peut choisir comme point de départ `[ 7.0 , -2000.0 , 500.0 ]`)

Question 28 :

```
def verif(H):
  vp = np.linalg.eig(H)[0]
  for val in vp :
      if val < 0 :
          return 1
  return 0
```

Question 29 : Il faut prendre $n = 5$ pour disposer de 6 degrés de liberté, et permettre que la courbe passe par les 6 points expérimentaux.

Question 30 : Appelons $T_1, \dots T_6$ et $C_1, \dots C_6$ les données expérimentales.

On a pour tout $k \in [\![1, 6]\!]$: $\displaystyle\sum_{i=0}^{5} B_i \left(\frac{T_k}{T_C}\right)^i = C_k$.

Question 31 : Ce système de 6 équations se met sous la forme $y_{\text{exp}} = Mb$, en prenant pour M une matrice de taille 6×6 dont le coefficient en ligne k, colonne i vaut $\left(\frac{T_k}{T_C}\right)^{i-1}$:

$$\begin{pmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \\ C_5 \\ C_6 \end{pmatrix} = \begin{pmatrix} \left(\frac{T_1}{T_C}\right)^0 & \left(\frac{T_1}{T_C}\right)^1 & \left(\frac{T_1}{T_C}\right)^2 & \left(\frac{T_1}{T_C}\right)^3 & \left(\frac{T_1}{T_C}\right)^4 & \left(\frac{T_1}{T_C}\right)^5 \\ \left(\frac{T_2}{T_C}\right)^0 & \left(\frac{T_2}{T_C}\right)^1 & \left(\frac{T_2}{T_C}\right)^2 & \left(\frac{T_2}{T_C}\right)^3 & \left(\frac{T_2}{T_C}\right)^4 & \left(\frac{T_2}{T_C}\right)^5 \\ \left(\frac{T_3}{T_C}\right)^0 & \left(\frac{T_3}{T_C}\right)^1 & \left(\frac{T_3}{T_C}\right)^2 & \left(\frac{T_3}{T_C}\right)^3 & \left(\frac{T_3}{T_C}\right)^4 & \left(\frac{T_3}{T_C}\right)^5 \\ \left(\frac{T_4}{T_C}\right)^0 & \left(\frac{T_4}{T_C}\right)^1 & \left(\frac{T_4}{T_C}\right)^2 & \left(\frac{T_4}{T_C}\right)^3 & \left(\frac{T_4}{T_C}\right)^4 & \left(\frac{T_4}{T_C}\right)^5 \\ \left(\frac{T_5}{T_C}\right)^0 & \left(\frac{T_5}{T_C}\right)^1 & \left(\frac{T_5}{T_C}\right)^2 & \left(\frac{T_5}{T_C}\right)^3 & \left(\frac{T_5}{T_C}\right)^4 & \left(\frac{T_5}{T_C}\right)^5 \\ \left(\frac{T_6}{T_C}\right)^0 & \left(\frac{T_6}{T_C}\right)^1 & \left(\frac{T_6}{T_C}\right)^2 & \left(\frac{T_6}{T_C}\right)^3 & \left(\frac{T_6}{T_C}\right)^4 & \left(\frac{T_6}{T_C}\right)^5 \end{pmatrix} \begin{pmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \end{pmatrix}$$

Question 32 : Pour obtenir les coefficients B_i il faut inverser la matrice M puis effectuer le produit matriciel $b = M^{-1} y_{\text{exp}}$.

Question 33 : Remarque : la plage des températures en abscisses correspond aux valeurs expérimentales : entre 100K et 4000K.

```
import matplotlib.pyplot as plt
x = np.linspace(100,4000,500)

A1,A2,A3 = vecA
y_modele1 = A1 + A2 / (x+A3)

B0,B1,B2,B3,B4,B5 =  vecB
Tc = 304.21
y_modele2 = B0 + B1*(x/Tc) + B2*(x/Tc)**2 + B3*(x/Tc)**3 + B4*(x/Tc)**4 + B5*(x/Tc)**5

plt.plot(x,y_modele1,"--k") # ligne pointillée noire
plt.plot(x,y_modele2,"-k") # ligne continue noire
plt.plot(temp,CpR_exp, "ok") # marqueurs ronds (o) noirs (k)
plt.xlabel("T(K)")
plt.ylabel("Cpm/R")
plt.show()
```