Devoir d'informatique

D'après Concours Mines-Ponts.

Modélisation numérique d'un matériau magnétique

Partie I : Transition paramagnétique/ferromagnétique sans champ magnétique extérieur

Question 1 *Écrire les instructions nécessaires pour importer exclusivement les fonctions exponentielle (* `exp` *) et tangente hyperbolique (* `tanh` *) du module* `math`*, ainsi que les fonctions* `randrange` *et* `random` *du module* `random`*. Ces fonctions seront ainsi utilisables dans tous les programmes que vous écrirez ultérieurement.*

Correction

```
from math import exp , tanh
from random import randrange , random
```

Question 2 *À partir de l'équation 2, indiquer une équation* `f(x, t) = 0`*, d'inconnue* `x` *que l'on doit résoudre et écrire en Python la définition de la fonction* `f` *correspondante (paramètres* `x` *et* `t`*, valeur renvoyée* `f(x, t)`*).*

Correction

```
def f(x, t):
    return x - tanh(x / t)
```

Question 3 *Écrire une fonction* `dicho(f, t, a, b, eps)` *qui calcule une valeur approchée à* `eps` *près du zéro d'une fonction* `f(x, t)` *de variable* `x` *et de paramètre* `t` *fixé sur un intervalle* $[a, b]$*. On supposera pour simplifier que la fonction dont on recherche le zéro est continue et s'annule une fois et une seule sur l'intervalle* $[a, b]$*.*

Correction

```
def dicho(f, t, a, b, eps ):
    while b - a > eps:
        m = (a + b) / 2
        if f(m, t) * f(b, t) < 0:
            a = m
        else:
            b = m
    return (a + b) / 2
```

Question 4 *Établir l'expression de la complexité temporelle asymptotique de la fonction* `dicho` *en fonction*

de `a` , `b` *et* `eps` .

Correction

Posons p, le nombre de tours de boucle while nécessaires, avec à chaque tour une division par 2 de la largeur de l'intervalle. L'algorithme se termine lorsque l'intervalle atteint une largeur de eps, ainsi $(b-a)/2^p = eps$. On en déduit $p = \log_2\left(\frac{b-a}{eps}\right)$. L'algorithme est donc en $\mathcal{O}\left(\log\left(\frac{b-a}{eps}\right)\right)$.

Question 5 *En utilisant la fonction* `dicho` *, écrire une fonction* `construction_liste_m(t1, t2)` *qui construit et retourne une liste de 500 solutions de l'équation (1), pour t variant linéairement de t_1 à t_2 (bornes incluses). On cherchera les valeurs de m à 10^{-6} près avec un intervalle de recherche initial $m \in [0.001, 1]$.*

Correction

```
def construction_liste_m (t1 , t2):
    solutions = []
    pas = (t2 - t1) / 499 # Pour avoir les 2 bornes incluses
    for i in range (500):
        t = t1 + pas*i
        if t >= 1:
            m = 0
        else:
            m = dicho(f, t, 0.001 , 1, 1e -6)
        solutions .append(m)
    return solutions
```

Partie II : Modèle microscopique d'un matériau magnétique

Question 6 *Écrire une fonction* `initialisation()` *renvoyant une liste d'initialisation des domaines contenant* `n` *spins de valeur 1 comme sur la figure* **??**.

Correction

```
def initialisation ():
    return [1] * n
```

Question 7 *Écrire une fonction* `initialisation_anti()` *renvoyant une liste* `s` *d'initialisation des domaines contenant h spins en largeur et h en hauteur en alternant les 1 et -1 comme sur la figure* **??**.

Correction

```
def initialisation_anti ():
    L = []
    for i in range(h):
        if i%2 == 0:
            L.extend ([1] * h)
        else:
            L.extend ([ -1] * h)
    return L
```

Question 8 *Pour afficher l'état global du matériau, il est nécessaire de convertir la liste s utilisée en un tableau de taille $h \times h$ représenté par une liste de listes. Écrire une fonction* `repliement(s)` *qui prend en argument la liste de spins s et qui renvoie une liste de h listes de taille h représentant le domaine.*

Correction

```
def repliement (s):
    L = []
    for i in range(h):
        L.append(s[h*i:h*(i+1)])
    return L
```

Question 9 *Définir une fonction* `liste_voisins(i)` *qui renvoie la liste des indices des plus proches voisins du spin s_i d'indice* `i` *dans la liste* `s` *(dans l'ordre gauche, droite, dessous, dessus). On pourra utilement utiliser les opérations % et // de Python, qui renvoient le reste et le quotient de la division euclidienne.*

Correction

```
def liste_voisins (i):
    v = []
    if i%h == 0: # Voisin de gauche
        v.append(i + h -1)
    else:
        v.append(i -1)
    if i%h == h -1: # Voisin de droite
        v.append(i - (h -1))
    else:
        v.append(i+1)
    if i//h == h -1: # Voisin du dessous
        v.append(i%h)
    else:
        v.append(i + h)
    if i//h == 0: # Voisin du dessus
        v.append ((h -1)*h + i%h)
    else:
        v.append(i - h)
    return v
```

Question 10 *Définir la fonction* `energie(s)` *qui calcule l'énergie d'une configuration* `s` *donnée (cf. équation 3).*

Correction

```
def energie(s):
    E = 0
    for i in range(n):
        E_i = 0
        L_voisins = liste_voisins (i)
        for iv in L_voisins :
            E_i += s[i] * s[iv]
        E += E_i
    return - 1/2 * E # En prenant J = 1 comme le demande le sujet
```

Question 11 *Définir une fonction* `test_boltzmann(delta_e, T)` *qui renvoie* `True` *si le spin change de signe, et* `False` *sinon.*

Correction

```
def test_boltzmann (delta_e , T):
    if delta_e <= 0:
        return True
    p = exp(-delta_e / T)
```

```
    if random () < p:
        return True
    return False
```

Question 12 *Juste après avoir sélectionné au hasard l'indice* `i` *d'un spin de la liste* `s` *à basculer éventuellement, pour évaluer l'écart d'énergie* `delta_e` *entre les deux configurations avant/après, on propose deux solutions sous forme des fonctions* `calcul_delta_e1` *et* `calcul_delta_e2` . `s[i]` *est le spin choisi pour être éventuellement retourné. Indiquer la solution qui vous paraît la plus efficace pour minimiser le temps de calcul en justifiant votre réponse.*

Correction

La solution 2, `calcul_delta_2e` est clairement la plus rapide, puisqu'elle part du principe que l'énergie étant la même presque partout, la variation à calculer ne nécessite que le calcul de l'énergie autour du spin à inverser. Elle se fait en temps constant, contrairement à la première qui se fait en temps linéaire.

Question 13 *En utilisant la fonction* `test_boltzmann` *, définir une fonction* `monte_carlo(s, T, n_tests)` *qui applique la méthode de Monte-Carlo et qui modifie la liste* `s` *où l'on a choisi successivement* `n_test` *spins au hasard, que l'on modifie éventuellement suivant les règles indiquées dans les explications.*

Correction

```
def monte_carlo (s, T, n_tests ):
    for i_test in range(n_tests ):
        i = randrange (0,n)
        if test_boltzmann ( calcul_delta_e2 (s, i), T):
            s[i] *= -1
```

Question 14 *Écrire la fonction* `aimantation_moyenne(n_tests, T)` *qui :*

- *initialise une liste des spins (avec la fonction* `initialisation` *par exemple);*
- *la fait évoluer en effectuant* `n_tests` *tests de Boltzmann et les inversions éventuelles qui en découlent;*
- *calcule et renvoie l'aimantation moyenne de la configuration à la température* `T` *(définie ici comme la somme des valeurs des spins divisée par le nombre total de spins).*

Correction

```
def aimantation_moyenne (n_tests , T):
    s = initialisation ()
    monte_carlo (s, T, n_tests)

    # Calcul de l'aimantation moyenne
    m = 0
    for i in range(n):
        m += s[i]
    return m/n
```

Question 15 *Évaluer la complexité asymptotique de la fonction* `aimantation_moyenne(n_tests, T)` *en fonction de* `n` *, nombre de spins dans le système, et de* `n_tests` .

Correction

La fonction initialisation se fait en temps linéaire de n. `monte_carlo` est en $\mathcal{O}(n_{\text{tests}})$) grâce à `calcul_delta_e2` qui est en temps constant. Enfin, le calcul de l'aimantation moyenne est en temps linéaire de n. Ainsi, la fonction `aimantation_moyenne` est en $\mathcal{O}(n_{\text{tests}}+n)$ ou $O(n_{\text{tests}}+h^2)$.

Question 16 *Cette complexité asymptotique serait-elle modifiée si on avait voulu prendre en compte toutes les*

interactions entre deux spins quelconques dans le système, et plus seulement entre les plus proches voisins ? Justifier.

Correction
Dans le cas où l'on prendrait en compte toutes les interactions, le calcul de ΔE aurait nécessité $\mathcal{O}(n)$ calculs. Cela aurait donc changé la complexité symptotique en $\mathcal{O}(n_{\text{tests}} \times n)$.

Question 17 *Indiquer l'influence de l'augmentation de la température sur le comportement du matériau ferromagnétique.*

Correction
L'augmentation de la température a tendance à rendre aléatoire la répartition des domaines $+1$ et -1, et donc à supprimer l'aimantation du matériaux.

Partie III : Exploration des domaines de Weiss

Question 18 *Écrire le code de la fonction récursive* `explorer_voisinage(s, i, weiss, num)` *conforme à la description ci-dessus.*

Correction
```
def explorer_voisinage (s, i, weiss , num ):
    for iv in liste_voisins (i):
        if weiss[iv] == -1 and s[iv] == s[i]:
            weiss[iv] = num
            explorer_voisinage (s, iv , weiss , num)
```

Question 19 *Écrire le code de la fonction itérative* `explorer_voisinage_pile(s, i, weiss, num, pile)` *conforme à la description ci-dessus.*

Correction
```
def explorer_voisinage_pile (s, i, weiss , num , pile ):
    pile.append(i)
    while len(pile) > 0:
        iv = pile.pop ()
        weiss[iv] = num
        for ivv in liste_voisins (iv):
            if weiss[ivv] == -1 and s[ivv] == s[i]:
                pile.append(ivv)
```

Question 20 *Écrire le code de la fonction* `construire_domaines_weiss(s)` *qui construit et renvoie la liste* `weiss` *contenant le numéro des domaines de Weiss de chaque spin du domaine.*

Correction
```
def construire_domaines_weiss (s):
    num , i = 0, 0
    weiss = [-1] * n
    while i < n:
        # On marque tous les spins du domaine
        pile = []
        explorer_voisinage_pile (s, i, weiss , num , pile)
        # On cherche un nouveau point de départ
        i += 1
        while i < n and weiss[i] != -1:
            i += 1
```

```
        num += 1
    return weiss
```