UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN PHYSIQUE

PAR
ÉRIC MÉLANÇON

ÉTUDE THÉORIQUE DE LA CONTRIBUTION DE LA PHYSISORPTION SUR L'ADSORPTION D'HYDROGÈNE À BASSE PRESSION SUR DES NANOSTRUCTURES DE CARBONE.

JUILLET 2003

RÉSUMÉ

Les récents changements climatiques et l'épuisement éventuel des réserves de pétrole poussent l'humanité à trouver des sources d'énergie propres et renouvelables. L'hydrogène pourrait remplacer l'essence dans le domaine du transport automobile si ce n'était du problème du stockage. L'hydrogène gazeux possédant une très faible quantité d'énergie par unité de volume, différentes méthodes de stockage (compression, liquéfaction, hydrures métalliques, adsorption) insatisfaisantes sont utilisées pour le stocker.

L'adsorption sur des nanotubes de carbone semble une avenue fort prometteuse pour stocker l'hydrogène, bien que les résultats expérimentaux soient encore très controversés et que les mécanismes pouvant les expliquer soient encore bien mal compris. C'est pourquoi nous étudierons dans ce travail la contribution de la physisorption d'hydrogène à basses pressions dans les nanostructures et nanotubes de carbone au moyen d'une expansion en série du viriel de la quantité adsorbée en excès. Dans la limite des faibles pressions considérée ici, l'isotherme d'adsorption est linéaire (loi de Henry) et ne comporte alors que le second coefficient du viriel B_{AS}. Ce coefficient permettra de bien identifier l'interaction d'une molécule d'adsorbat avec la surface.

Nous allons étudier le comportement du potentiel d'adsorption des différentes nanostructures et nous comparerons leurs énergies d'interaction avec l'adsorbat. Nous examinerons également les effets de corrugation du potentiel dû au fait que la surface des nanotubes est discrète. Les effets quantiques, présents aux faibles températures, seront étudiés au moyen d'un potentiel d'adsorption effectif et nous verrons dans quelles conditions ils pourront être négligés. Nous déterminerons les dimensions optimales du nanotube et du faisceau de nanotubes qui maximisent la quantité d'hydrogène adsorbée (et B_{AS}) et nous identifierons les sites d'adsorption qui contribuent le plus à l'adsorption totale. Finalement, nous pourrons déduire la surface spécifique et la taille moyenne des pores d'un charbon activé en extrayant les B_{AS} des isothermes d'adsorption et en les ajustant aux courbes théoriques obtenues pour un pore en fente.

REMERCIEMENTS

Le travail présenté dans ce mémoire a été effectué au sein de l'Institut de recherche sur l'hydrogène (IRH), rattaché au département de physique de l'Université du Québec à Trois-Rivières. Tout au long de ce travail, j'ai été soutenu par différentes personnes et organismes dont je tiens à témoigner ma gratitude.

Premièrement, je tiens à remercier chaleureusement mon directeur de recherche, monsieur Pierre Bénard, pour ses judicieux conseils, pour son appui et pour m'avoir guidé tout au long de ce travail. De plus, monsieur Bénard m'a apporté une aide financière précieuse.

Je tiens également à remercier monsieur Tapan K. Bose, directeur de l'Institut, pour ses encouragements et pour l'aide financière fournie. Je désire aussi souligner qu'en dépit de son emploi du temps des plus chargé il a su trouver du temps pour des rencontres qui m'ont été des plus enrichissantes.

Des remerciements vont également à monsieur Benjamin Angers, étudiant au double baccalauréat en physique et informatique, qui lors d'un stage à l'Institut a effectué des simulations Monte Carlo grand canonique et des calculs du potentiel d'adsorption sur un nanotube discret afin de pouvoir corroborer mes résultats.

De manière plus générale, j'aimerais également remercier les professeurs et chargés de cours du département de physique pour l'enseignement et les conseils qu'ils m'ont prodigués tout au long de ma formation. Je tiens également à remercier mes parents, Robert et Sylvie Mélançon pour m'avoir offert leur appui inconditionnel tout au long de ce travail.

Les organismes suivants méritent également toute ma gratitude pour m'avoir supporté financièrement : le conseil de recherches en sciences naturelles et en génie du Canada (CRSNG), la Fondation du C.E.U. de Trois-Rivières et la Fondation de l'Université du Québec à Trois-Rivières.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SYMBOLES ET ABRÉVIATIONS

( )*	De manière générale, sert à noter la quantité réduite (sans dimensions) correspondant à ( )
(n, m)	Indices permettant de caractériser un nanotube à paroi simple
(p, q)	L'état classique du système dans l'espace des phases
A	L'aire d'une feuille de graphène et de chacune des parois d'un pore en fente
a_0	Distance entre deux atomes de carbone sur une feuille de graphène (1.42 Å)
$\mathbf{a_1}$, $\mathbf{a_2}$	Vecteurs primitifs du réseau hexagonal formant la surface du nanotube
A_{acc}	La projection du volume accessible aux molécules d'adsorbat dans un plan perpendiculaire au faisceau
a_{ni}	Coefficients des séries servant à calculer les fonctions $M_n(x)$
B_{AS}	Le second coefficient du viriel de la quantité adsorbée en excès
BET	Modèle de Brunauer-Emmett-Teller utilisé pour modéliser l'adsorption
C	Vecteur d'enroulement le long de la circonférence du nanotube
c−SWNT	SWNT fermé aux extrémités (closed SWNT)
C_{AAS}	Le troisième coefficients du viriel de la quantité adsorbée en excès
cf.	Utilisé pour renvoyer à un autre élément
D	La profondeur du puits du potentiel d'adsorption
d	Selon le contexte, dénote le diamètre du nanotube, la largeur d'un pore en fente ou la distance séparant les centres des nanotubes d'un faisceau
d_0	Largeur où le potentiel devient répulsif dans tout le pore en fente
d_c	Largeur critique où le minimum de V_{slit} cesse d'être au centre du pore
DFT	Théorie de la densité fonctionnelle (Density Functional Theory)
d_{min}	Largeur donnant le puits de potentiel le plus profond dans le pore en fente
DOE	Département de l'énergie américain (U.S. Department of Energy)
$\|E_n\rangle$	Un état propre de l'hamiltonien du système
e.g.	Utilisé pour donner un exemple
E_A	L'énergie d'activation
E_n	Les énergies propres du système
f	L'intégrand $\exp(-\beta V)-1$ entrant dans le calcul des B_{AS}
F	L'énergie libre de Helmholtz
$F(a, b, c; u)$	La fonction hypergéométrique
f(H)	la distribution de la taille des pores du charbon activé
F_0	Une approximation de l'énergie libre de Helmholtz
g	Distance de Van der Waals séparant les nanotubes d'un faisceau (3.15 Å)
i	Indice (≥ 0) servant à numéroter les couches concentriques de nanotubes formant un faisceau
I_{i1}	La règle du trapèze obtenue par i bissections consécutives des sous-intervalles
I_{i2}	La $(i + 1)^{ème}$ règle de Simpson
$J_0(\tilde{x})$, $I_0(\tilde{x})$	Fonction de Bessel et fonction de Bessel modifiée
k	La constante de Boltzmann
K	La constante de force dans l'approximation harmonique du potentiel
L	La longueur d'un nanotube
LJ 12−6	Potentiel de Lennard-Jones ayant les exposants 12 et 6

M_C	La masse molaire du carbone
M_{H_2}	La masse molaire de l'hydrogène moléculaire
$M_n(x)$	Fonctions apparaissant dans la définition de V(r, R) (où n = 5 ou 11)
MWNT	Nanotube à parois multiples (multi-walled nanotube)
N	Le nombre de molécules d'adsorbat
$\overline{N}^0$, $\overline{N}$	Le nombre moyen de molécules d'adsorbat en absence et en présence de la surface
N_a	La quantité d'adsorbat adsorbée en excès
N_{SI}	Nombre de sites intersticiels compris à l'intérieur de la $i^{ème}$ couche d'un faisceau
N_{SP}	Nombre de sites périphériques à l'extérieur de la $i^{ème}$ couche d'un faisceau
N_t	Nombre de nanotubes compris à l'intérieur de la $i^{ème}$ couche d'un faisceau. Utilisé également pour désigner le nombre de nanotubes formant un faisceau quelconque ($\equiv N_{tubes}$)
N_{tubes}	Le nombre de nanotubes formant un faisceau
o–SWNT	SWNT ouvert aux extrémités (opened SWNT)
p	La pression dans le contenant, loin de la surface de l'adsorbant
P	Période de l'enveloppe du minimum du potentiel d'adsorption à l'intérieur d'un SWNT discret. Sert également à désigner la pression exprimée en atmosphères
points A	Désigne les atomes de carbone à la surface d'une feuille de graphène et d'un SWNT
points S	Désigne les centres d'hexagones à la surface d'une feuille de graphène et d'un SWNT
$Q_N(V,T)$, $Q(N,V,T)$	La fonction de partition canonique
Q_N^0, Q_N	Les fonctions de partition canonique classiques en absence et en présence de la surface
r	La distance mesurée à partir de l'axe du nanotube
R	Le rayon d'un nanotube
$\vec{r}$	La position de la molécule d'adsorbat
R_0	Rayon où le potentiel devient répulsif dans tout le nanotube
R_c	Rayon critique où le minimum de V(r, R) cesse d'être sur l'axe du nanotube
$\vec{R}_j$	La position de la molécule d'adsorbant (i.e. de l'atome de carbone)
r_{min}	Position du minimum de u(r) ou de V(r, R). Sert également à noter l'endroit où commence l'adsorption sur un MWNT.
R_{min}	Rayon donnant le puits de potentiel le plus profond dans le nanotube
$\vec{R}_{pq}$	La position des centres des nanotubes d'un faisceau
$S[x(u)]$, $S_0[x(u)]$	Des fonctionnelles de la trajectoire x(u)
SWNT	Nanotube à paroi simple (single-walled nanotube)
T	La température de l'adsorbat
T_{Boyle}	La température de Boyle où $B_{AS} = 0$

u	$u = \beta\hbar$, le "temps" apparaissant dans la théorie des intégrales de parcours de Feynman menant au potentiel effectif
u(r)	Potentiel d'interaction entre une molécule d'adsorbant et d'adsorbat (dans ce travail, c'est un LJ 12−6)
U_N^0, U_N	L'énergie potentielle totale du gaz en absence et en présence de la surface
$u_s(\vec{r})$	Le potentiel d'adsorption d'une surface évalué au point $\vec{r}$
V	Le volume accessible aux molécules d'adsorbat
$V(\vec{r},d,R,N_t)$	Potentiel d'adsorption d'un faisceau de nanotubes à parois simples (aussi noté par V_{bundle})
$V(r,R)$	Potentiel d'adsorption d'un nanotube à paroi simple
V_{bundle}	Potentiel d'adsorption d'un faisceau de nanotubes à parois simples
V_{eff}	Le potentiel d'adsorption effectif
$V_{flat}(z)$	Potentiel d'adsorption d'une feuille de graphène
$V^*_{flat,\ tronq}$	Le potentiel d'adsorption tronqué d'une feuille de graphène
$V_{slit}(z,d)$	Potentiel d'adsorption d'un pore en fente
w_n	La probabilité que le système soit dans l'état propre $\lvert E_n \rangle$
x	Le rapport de la distance de l'axe sur le rayon du nanotube (= r/R)
$\bar{x}$	La position moyenne sur la trajectoire x(u)
z_{min}	Position du minimum de $V_{flat}(z)$ ou de V(r, R), mesurée à partir de la surface
Z_N^0, Z_N	Les intégrales de configurations classiques en absence et en présence de la surface
Δ_{1e}	Écart relatif entre l'énergie d'activation et le puits de potentiel à l'extérieur d'un SWNT isolé
Δ_{1i}	Écart relatif entre l'énergie d'activation et le puits de potentiel à l'intérieur d'un SWNT isolé
Δ_I	Écart relatif entre l'énergie d'activation et le puits de potentiel à l'intérieur des interstices du faisceau
Δ_P	Écart relatif entre l'énergie d'activation et le puits de potentiel à l'intérieur des sites périphériques du faisceau
Δ_T	Écart relatif entre l'énergie d'activation et le puits de potentiel à l'intérieur d'un SWNT du faisceau
Λ	La longueur d'onde thermique des molécules d'adsorbat
β	$1/kT$
ε	Profondeur du puits de potentiel de Lennard-Jones entre une molécule d'adsorbat et d'adsorbant (ε_{C-H_2} = 30.5 K)
ε_s'	Énergie (= $\pi\theta\varepsilon\sigma^2$) apparaissant dans la définition des potentiels d'adsorption (= 371 K = 3.08 kJ/mol pour l'interaction C−H_2)
ϕ	Angle chiral du nanotube (formé par les rangées d'hexagones et l'axe du nanotube)
μ	Le potentiel chimique de l'adsorbat

θ	Densité surfacique uniforme d'atomes de carbone formant la surface des nanostructures (= 0.38 $Å^{-2}$)
ρ	L'opérateur densité permettant de décrire le système
$\rho(x,x';u)$	L'opérateur densité dans la représentation des coordonnées
σ	Diamètre coeur dur du potentiel de Lennard-Jones entre une molécule d'adsorbat et d'adsorbant ($\sigma_{C-H_2} = 3.19$ Å)
σ_T	L'écart type de la gaussienne entrant dans la définition du potentiel d'adsorption effectif
Ξ^0, Ξ	Les fonctions de partition grand canonique classiques en absence et en présence de la surface

INTRODUCTION

Depuis quelques années, les médias parlent fréquemment des changements climatiques qui affectent notre planète : inondations fréquentes, sécheresses, étés anormalement secs et chauds… Force est de constater que les phénomènes météorologiques extrêmes sont de plus en plus fréquents et que la vaste majorité de la communauté scientifique croit que tout ceci est causé par les gaz à effet de serre, notamment le gaz carbonique, produits par l'homme. Les gaz d'échappement libérés par les automobiles sont une cause importante du smog et de la mauvaise qualité de l'air dans les grandes villes. À mesure que la population mondiale s'accroît, nous avons besoin de plus d'énergie et nous consommons de plus en plus de combustibles fossiles, ce qui en plus d'accélérer le changement climatique et de polluer l'environnement urbain nous amène à un second problème beaucoup moins médiatisé mais tout aussi important : l'épuisement inévitable des combustibles fossiles. On prévoit [1, 2] que la production annuelle maximale de pétrole sera atteinte d'ici environ 2010, après quoi elle diminuera inévitablement d'année en année. Cela va alors provoquer une crise pétrolière sans précédent et aura un impact négatif sur l'économie mondiale. Certains pays pourraient alors être tentés d'accaparer les réserves restantes et ainsi provoquer des conflits internationaux. Il est donc primordial de trouver des sources d'énergies propres et renouvelables.

Dans le domaine du transport automobile, l'hydrogène est un carburant prometteur qui pourrait remplacer l'essence. Cependant un des obstacles majeurs à son utilisation est sa faible densité volumétrique d'énergie. Il est donc primordial d'améliorer les techniques de stockage de l'hydrogène (compression, liquéfaction, hydrures métalliques, adsorption). Le département de l'énergie américain (DOE) a déterminé que pour avoir un réservoir de dimensions et de poids comparables aux réservoirs à essence actuels et une autonomie de 500 km, un système de stockage à l'hydrogène doit atteindre l'objectif de 62 kg H_2 / m^3 et 6.5 % en poids d'hydrogène (6.5 wt%) [3, 4].

Suite à une première publication de A.C. Dillon [4] mentionnant un résultat expérimental de 5 à 10 wt% d'hydrogène adsorbé sur un échantillon contenant des nanotubes de carbone à température ambiante, l'adsorption sur les nanotubes est apparue comme une solution prometteuse au problème du stockage. Cependant ses résultats ont été contestés et il existe

actuellement une controverse quant à la quantité réelle d'hydrogène qu'on peut stocker dans les nanotubes. De nouveaux résultats expérimentaux indiquent que la capacité de stockage varie de 0 à 10 wt% [5] :

Tableau I.1. Résumé des capacités de stockage expérimentales d'hydrogène dans les nanotubes.

Référence et année	**Capacité de stockage (wt%)**	**Température et pression**
[4] (1997)	5-10	275 – 300 K, P < 1atm
[6] (1999)	8	80 K, 120 atm
[7] (1999)	4.2	300 K, 100 atm
[8] (1999)	0.39	300 K, 1 atm
[9] (2000)	3 - 6	300 K, 80 – 100 atm
[10] (2000)	2.9	300 K, 1 atm

Il est donc important de faire des études théoriques sur l'adsorption d'hydrogène dans les nanotubes (et autres nanostructures) afin d'identifier les mécanismes et afin de fixer une limite supérieure théorique sur la quantité d'hydrogène qu'on peut y adsorber. Des études théoriques permettent notamment de savoir si l'adsorption a lieu principalement à l'intérieur ou à l'extérieur des nanotubes et si l'adsorption est due à de la physisorption et/ou de la chimisorption.

À ce jour, plusieurs types de calculs et de simulations d'adsorption dans les nanotubes et autres nanostructures ont été effectués. Nous allons mentionner ici les principales méthodes utilisées et leurs références respectives :

1) Utilisation du second coefficient du viriel B_{AS} [11-14]. Une méthode valide dans la limite des basses pressions et faibles densités et qui consiste à exprimer linéairement l'isotherme d'adsorption.

2) Méthode Monte Carlo grand canonique classique [15, 16]. Une méthode qui consiste à créer, déplacer ou détruire une molécule d'hydrogène du système de manière aléatoire en suivant certaines probabilités.

3) Méthode Monte Carlo grand canonique quantique [17, 18]. Une méthode Monte Carlo qui tient compte des effets quantiques (significatifs quand m, $T \rightarrow 0$) au moyen des intégrales de parcours de R. P. Feynman.

4) Simulation avec dynamique moléculaire [19]. Une méthode qui consiste à intégrer les équations du mouvement des molécules d'hydrogène.

5) Théorie de la densité fonctionnelle (DFT) [20-22]. Une méthode qui consiste à trouver la densité à l'intérieur des pores faisant en sorte que le grand potentiel $\Omega[\rho(\mathbf{r})]$ est minimal. On peut également obtenir la distribution de la taille des pores d'un échantillon au moyen de cette théorie.

Au cours de ce travail, notre objectif principal sera d'étudier la contribution de la physisorption à l'adsorption d'hydrogène dans les nanostructures de carbone (et surtout dans les nanotubes). Dans cette étude, nous étudierons le cas de la limite des faibles pressions et densités au moyen d'un développement en viriel au second ordre (B_{AS}) de la quantité d'hydrogène adsorbée en excès (N_a). Cette approche permet de bien identifier l'énergie d'interaction entre une molécule d'adsorbat et la surface et de déterminer les sites d'adsorption privilégiés et les dimensions optimales de l'adsorbant. La limite des faibles pressions est intéressante car elle permet de calculer l'isotherme d'adsorption avec un minimum d'approximations (i.e. négliger les interactions adsorbat-adsorbat).

Notre exposé se déroulera comme suit : au chapitre 1, nous présenterons les méthodes de synthèse et la structure des différentes nanostructures étudiées. Puis, au chapitre 2 nous verrons comment se comporte le potentiel d'adsorption dans les différentes nanostructures et nous discuterons d'une tentative d'améliorer la théorie en incluant un potentiel d'adsorption effectif pour inclure les effets quantiques. Ensuite, je donnerai la théorie générale du second coefficient du viriel au chapitre 3 de même que la forme de ce coefficient pour les nanostructures étudiées. Au chapitre 4, nous verrons les algorithmes et les principes utilisés dans les programmes calculant les B_{AS}. Et au chapitre 5, nous donnerons les différents résultats que nous avons obtenus lors de ce travail. Puis pour terminer, nous énoncerons nos conclusions.

CHAPITRE 1. PRÉSENTATION DES NANOSTRUCTURES DE CARBONE ÉTUDIÉES.

1.1. Les charbons activés.

Lors de ce travail, nous avons étudié l'adsorption d'hydrogène sur le charbon activé AX-21 et l'adsorption de méthane sur le CNS-201 afin de valider expérimentalement la théorie du second coefficient du viriel (cf. chapitre 5). Nous allons donc commencer par présenter brièvement cet adsorbant communément utilisé dans le domaine de l'adsorption.

Le charbon activé est obtenu à partir d'un précurseur contenant du carbone (e.g. bois, coquilles de noix de coco) qui est brûlé en absence d'air dans un four pour éliminer les composés organiques volatiles contenus dans celui-ci. Ensuite, le charbon obtenu est activé en le chauffant dans un four en présence de vapeur d'eau afin de développer un réseau complexe de pores ayant des formes et tailles variables. La présence d'une multitude de pores très étroits, appelés micropores, a pour effet de donner au charbon activé une grande surface spécifique variant de 1000 à 3000 m^2/g.

Malgré la grande complexité du réseau de pores contenus dans le charbon activé, le modèle du pore en fente ("slit pore model") est très souvent utilisé dans la littérature (e.g. réfs. [18, 22, 23]) pour représenter la structure des pores du charbon activé lors de l'adsorption. Ce modèle consiste (Fig. 1.1) en deux feuilles de graphène infinies séparées d'une distance d, mesurée à partir des centres des atomes de carbone.

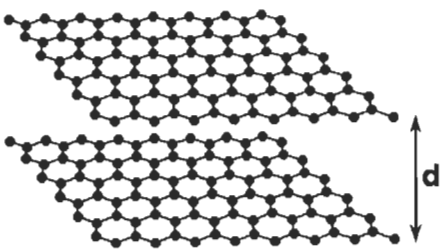


Figure 1.1. Modèle du pore en fente.

Le fait que les pores ont une forme plutôt aplatie que cylindrique est justifié par Everett et Powl dans la référence [23] où ils mentionnent que certains charbons peuvent laisser passer des grosses molécules plates comme le benzène alors que des molécules sphériques de mêmes dimensions

n'arrivent pas à pénétrer dans ceux-ci. De plus, ils ont montré que l'utilisation de pores cylindriques pour modéliser une série de trois charbons activés permettait la pénétration de molécules dans les pores qui sont normalement bloquées par ces charbons. La largeur moyenne d des pores en fente représentant les ultramicropores a été estimée à 7.7 Å, à partir de l'adsorption de Ar et de Kr sur des charbons microporeux polymérisés ("microporous polymer carbons").

1.2. Les nanotubes de carbone.

Les nanotubes sont des nanostructures de carbone tubulaires que l'on retrouve dans les suies produites par l'évaporation d'électrodes de graphite, contenant du cobalt, dans un arc électrique [4]. Mais cette méthode produit des petits échantillons contenant une faible proportion de nanotubes (~0.1 wt%). C'est pourquoi une nouvelle méthode consistant en l'évaporation d'une cible contenant un mélange de graphite, Co et Ni au moyen d'un laser est maintenant fréquemment utilisée [24] (les échantillons produits contiennent >70 % de nanotubes).

Les nanotubes à parois simples ("single-walled nanotube", SWNT) sont des fullérènes géantes, des molécules qui par définition sont des structures convexes, refermées sur elles-mêmes et ne comportant que des faces hexagonales et pentagonales. La famille des fullérènes contient notamment le "buckyball" (C60), une molécule de carbone en forme de ballon de soccer servant de point de départ à l'élaboration de structures plus complexes, dont notamment les nanotubes. Une caractéristique intéressante de la famille des fullérènes est que chacun de ses membres doit comporter 12 faces pentagonales (cf. annexe 1). La figure 1.2 montre un buckyball et un nanotube à parois simple fermé par deux hémisphères (c-SWNT).

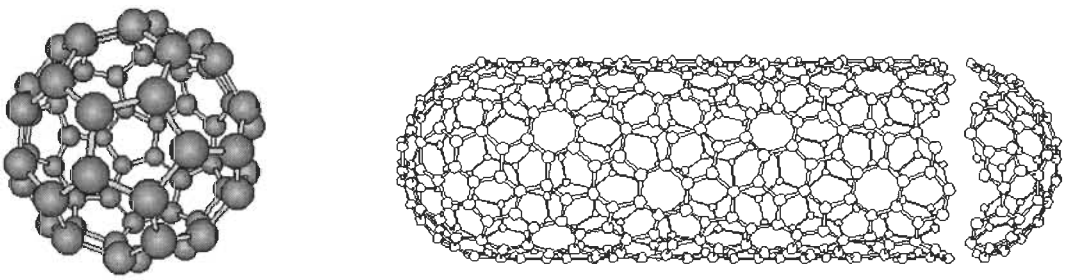


Figure 1.2. Deux fullérènes remarquables : le buckyball et le nanotube fermé.

Formellement, un SWNT peut s'obtenir en enroulant une bande découpée dans une feuille de graphène sur elle-même (Fig. 1.3 a) [25] de manière à ce que les sites correspondants de part et d'autre de la bande coïncident parfaitement. Après avoir enroulé la bande sur elle-même, nous obtenons un nanotube sans défauts entièrement spécifié par les indices (n, m) qui déterminent le vecteur de circonférence $\mathbf{C} = n\mathbf{a}_1 + m\mathbf{a}_2$. Le vecteur de circonférence permet de ramener un site donné sur lui-même après avoir refermé le tube (par exemple l'atome de carbone (0, 0) devient (11, 7)). La figure 1.3 b montre l'angle chiral ϕ, formé entre l'axe T du nanotube et les rangées d'hexagones (dirigées selon l'axe H) enroulées en spirales autour du tube. Dépendant des indices (n, m), nous obtenons différents types de nanotubes :

(n, 0) : nanotube zigzag, $\phi = 30°$. Il est obtenu en roulant la bande le long d'une rangée d'hexagones (indiquée par la ligne pointillée zigzag).

(n, n) : nanotube armchair, $\phi = 0°$. Il est obtenu en roulant la bande le long de la ligne pointillée armchair.

(n, m) : nanotube chiral, $0 \leq \phi \leq 30°$. Il est obtenu en roulant la bande le long du vecteur de circonférence **C**.

La figure 1.3 c montre le nanotube (11, 7) tel qu'il apparaît sous un microscope à effet tunnel.

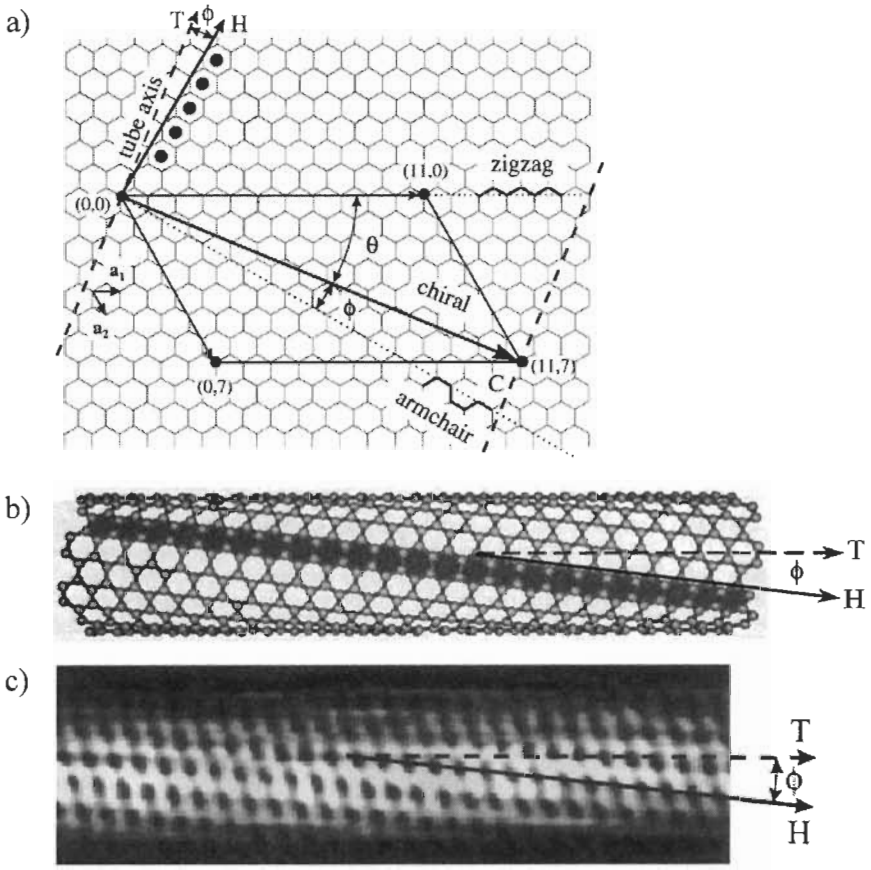


Figure 1.3. Formation d'un nanotube.

Sachant que la distance entre deux atomes de carbone contenus dans une feuille de graphène vaut $a_0 = 1.42$ Å, il est aisé d'obtenir la relation entre les indices (n, m) et le diamètre d du nanotube, de même que son angle chiral. Ainsi, de la loi du cosinus :

$$(\pi\, d)^2 = C^2 = a^2\left(n^2 + m^2 - 2mn\cos 120°\right)$$

Où la longueur a des vecteurs primitifs, donnée par $a = a_0\sqrt{3}$, permet d'obtenir :

$$d = \frac{a_0\sqrt{3}}{\pi}\sqrt{n^2 + m^2 + mn} = 0.783\sqrt{n^2 + m^2 + mn} \quad \text{Å} \tag{1.1}$$

De manière similaire, la loi du sinus permet d'obtenir l'angle chiral :

$$\frac{C}{\sin 120°} = \frac{ma}{\sin\theta} = \frac{na}{\sin\left(180° - \left(120° + \theta\right)\right)}$$

Qui devient, après avoir utilisé l'identité trigonométrique du sin(A-B) :

$$m\left(\sqrt{3}/2\cos\theta - 1/2\sin\theta\right) = n\sin\theta$$

$$\tan\theta = \sqrt{3}\frac{m}{m+2n} \tag{1.2}$$

$$\phi = 30° - \arctan\left[\sqrt{3}\frac{m}{m+2n}\right] \tag{1.3}$$

1.3. Les faisceaux de nanotubes de carbone.

Lors de leur formation, les nanotubes ont tendance à s'organiser naturellement en faisceaux ("bundles") entremêlés pouvant compter quelques centaines d'individus (Fig. 1.4 a) [24]. Les faisceaux ainsi obtenus ont plusieurs micromètres de longueur et comme le montre un agrandissement de la figure 1.4 a, chaque nanotube qui les constitue se situe sur un réseau hexagonal (visible sur la Fig. 1.4 b lorsque le faisceau se replie dans le plan d'observation du microscope).

a) b)

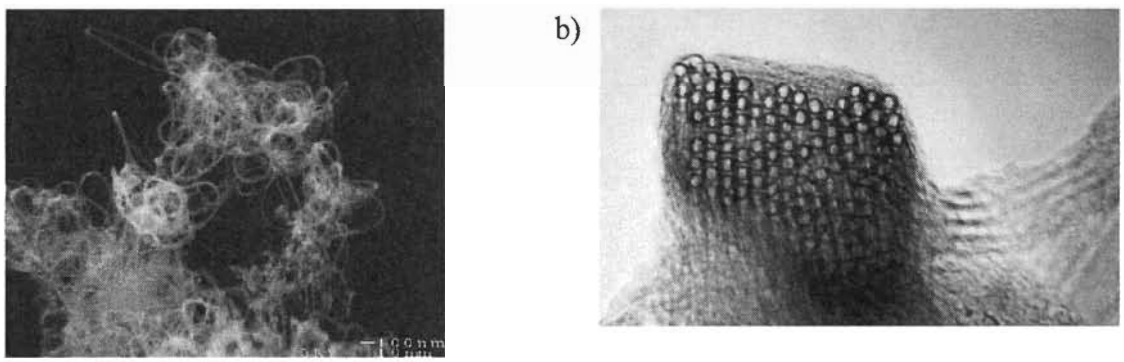


Figure 1.4. Échantillon contenant des faisceaux de nanotubes.

Les nanotubes, tels qu'obtenus après leur formation sont fermés aux bouts par deux hémisphères qu'il faut enlever si l'on veut bénéficier des sites d'adsorption très attractifs situés à l'intérieur des tubes. Une méthode utilisée [26] consiste à couper chimiquement les faisceaux au moyen d'un traitement aux acides et aux ultrasons après quoi ils sont chauffés pour enlever les groupes fonctionnels qui bloquent encore les ouvertures. Ceci fait, nous obtenons finalement des faisceaux de nanotubes ouverts (o-SWNT).

Dépendant du type d'appareil utilisé pour les produire, le diamètre des SWNTs peut varier entre 12 et 14 Å (e.g. réf. [27]). Cependant, la distance de Van der Waals qui les séparent doit

rester fixée à 3.15 Å (cf. [24]), un espacement très similaire à celui existant entre les feuilles de graphène d'un bloc de graphite.

La figure suivante montre les différents sites d'adsorption intéressants présents sur un faisceau idéalisé formé de couches concentriques de nanotubes.

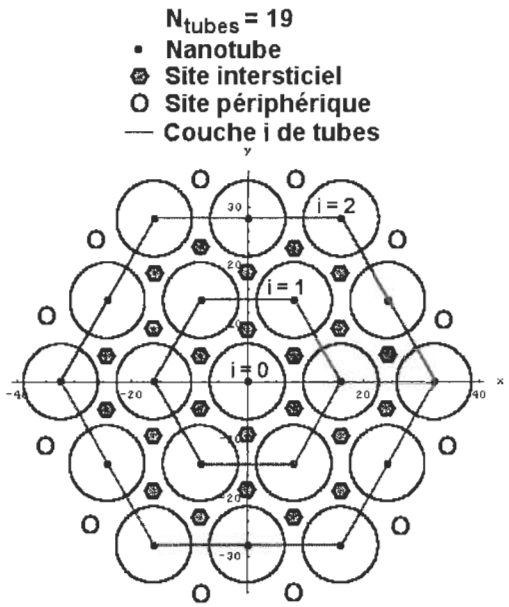


Figure 1.5. Évaluation du nombre de sites d'adsorption dans un faisceau.

On voit que sur une couche i > 0, il y aura 6(i + 1) – 6 nanotubes, de sorte que le nombre de tubes compris à l'intérieur de la couche i sera :

$$N_t = 1 + 6\sum_{j=1}^{i} j = 1 + 6\left(\frac{1}{2} i(i+1)\right)$$

$$N_t(i \geq 0) = 3i^2 + 3i + 1 = 1, 7, 19, 37, 61, 91, \ldots \tag{1.4}$$

On voit également qu'il y a entre les couches i et i + 1, (i) + (i + 1) sites intersticiels par côté d'hexagone compris entre les deux couches. Sachant cela, le nombre de sites intersticiels compris à l'intérieur de la couche i est :

$$N_{SI}(i \geq 1) = \sum_{j=0}^{i-1} 6(2j+1) = \sum_{j=1}^{i} 6(2j-1) = 6i^2 = 6, 24, 54, \ldots \tag{1.5}$$

Et de la Fig. 1.5, on a que le nombre de sites périphériques à la couche i est donné par :

$$N_{SP}(i \geq 1) = 6i \qquad \left(\propto \sqrt{N_t} \ quand \ N_t \to \infty\right) \tag{1.6}$$

Il est utile de comparer le nombre de tubes et de sites périphériques par rapport au nombre d'interstices :

$$\frac{N_t}{N_{SI}}(i \geq 1) = \frac{1}{2} + \frac{1}{2i} + \frac{1}{6i^2} = 1.17, 0.792, 0.685, \ldots \qquad (\to 1/2) \tag{1.7}$$

$$\frac{N_{SP}}{N_{SI}}(i \geq 1) = \frac{1}{i} = 1, 0.5, 0.333, \ldots \qquad (\to 0) \tag{1.8}$$

CHAPITRE 2. LE POTENTIEL D'ADSORPTION.

2.1. Le potentiel intermoléculaire.

Avant de calculer le potentiel d'adsorption des différentes nanostructures, nous avons besoin d'un modèle pour représenter l'interaction existant entre une molécule d'adsorbant et d'adsorbat. Plusieurs formes pour le potentiel intermoléculaire sont possibles [28] :

1) Potentiel de Lennard-Jones (LJ 12-6) : $u(r) = \frac{A}{r^{12}} - \frac{B}{r^6}$
2) Potentiel (Exp-6) : $u(r) = C\exp(-\alpha r) - \frac{D}{r^6}$ (Un potentiel qui doit être tronqué à $r \approx 0$ à cause de la divergence attractive en $-D/r^6$).
3) Potentiel de Yukawa-6 : $u(r) = E\,r^{-1}\exp(-\alpha r) - \frac{F}{r^6}$ (Un potentiel qui doit être tronqué à $r \approx 0$ à cause de la divergence attractive en $-F/r^6$).
4) Potentiels anisotropes : u dépend aussi de l'orientation des molécules entre elles.

Dans ce travail, nous nous sommes limités au potentiel LJ 12-6 étant donné sa simplicité et le fait qu'il est utilisé très souvent dans la littérature pour représenter le potentiel de physisorption (pour des molécules considérées sphériques). Le terme attractif $-B/r^6$ provient de la force de London causée par des dipôles instantanés mutuellement induits dans les deux molécules. Tandis que le terme A/r^{12} est un terme empirique utilisé pour représenter la répulsion des nuages électroniques des molécules lorsque celles-ci s'approchent de trop près. La plupart du temps, le potentiel de LJ est réécrit sous la forme (dépendant également de deux paramètres ε, σ) :

$$u(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] \tag{2.1}$$

Qui a son minimum $-\varepsilon$ à $r_{\min} = 2^{1/6}\sigma = 1.122\sigma$. σ est appelé le diamètre coeur dur puisque $u(\sigma) = 0$ et que le potentiel est répulsif lorsque les centres des deux molécules deviennent plus rapprochés que σ. L'allure de ce potentiel est illustrée ci-dessous.

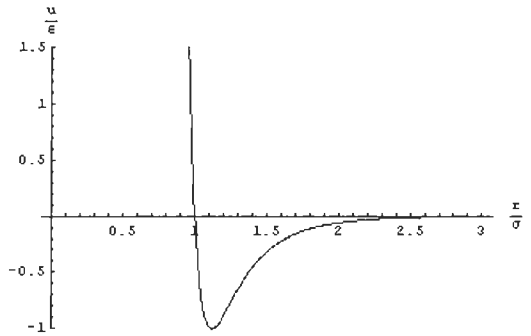


Figure 2.1. Potentiel intermoléculaire de Lennard-Jones.

La plupart du temps, les paramètres ε et σ sont déterminés expérimentalement à partir de mesures du second coefficient du viriel sur les gaz purs ($p/kT = \rho + B_2(T)\rho^2$). Les règles de Lorentz-Berthelot [29] données ci-dessous sont utilisées pour obtenir l'interaction entre deux molécules de sortes différentes (carbone et gaz) .

$$\sigma_{gC} = \frac{\sigma_{gg} + \sigma_{CC}}{2}$$
$$\varepsilon_{gC} = \sqrt{\varepsilon_{gg}\,\varepsilon_{CC}} \qquad (2.2)$$

Ces deux règles sont en fait des moyennes arithmétiques et géométriques respectivement.

Lors de ce travail, nous avons utilisé les valeurs suivantes : $\varepsilon_{C-H_2} = 30.5$ K, $\sigma_{C-H_2} = 3.19$ Å [15], $\varepsilon_{C-CH_4} = 66.0$ K , $\sigma_{C-CH_4} = 3.60$ Å [30], les valeurs de ε étant réduites par la constante de Boltzmann. Durant l'étude de différents articles de la littérature, différentes valeurs de ε et σ furent trouvées pour les mêmes gaz. Afin de clarifier cela et afin d'être sûr de mes valeurs, j'ai regroupé dans le tableau suivant les différentes valeurs de ε, σ que j'ai trouvées dans la littérature :

Tableau 2.1 Valeurs des paramètres de Lennard-Jones pour le carbone, l'hydrogène et le méthane.

Interaction pour la paire X-Y	**ε (K)**	**σ (Å)**	**Références**	**Méthode utilisée (commentaires)**	**Numéro de la valeur (#)**
C-C	28.2	3.4	16	Potentiel LJ fictif	1
	28.0	3.40	31, 33, 34	Potentiel LJ fictif	2
	33	3.469	32	Valeur peu répandue	3
H_2-H_2	37.0	3.05	31		4
	37.00	2.928	29	Vient du $B_2(T)$	5
	36.90	2.928	19		6
	36.7	2.958	16		7
	34.2	2.96	33		8
	33.3	2.968	15, 29	Valeur obtenue de la viscosité (moins bonne ici)	9
	59.7	2.827	35	Valeur disparate	10
CH_4-CH_4	148.2	3.817	29, 30	Vient du $B_2(T)$	11
	148.1	3.81	34		12
	148.6	3.758	35	Valeur obtenue de la viscosité (moins bonne ici)	13
	148	3.45	31		14
C-H_2	32.2	3.23	31	Équ. (2.2) + #4 + #2	15
	30.5	3.19	15	Équ. (2.2) + #9 + #2	16
	32.2	3.18	16	Équ. (2.2) + #7 + #1	17
	32.05	3.179	19	De l'adsorption de H_2 / graphite	18
	30.95	3.18	33	Équ. (2.2) + #8 + #2	19
	32.2	3.16		Équ. (2.2) + #5 + #2	20
	42.8	2.97	36	Valeur disparate	21
C-CH_4	66	3.60	30	De l'adsorption de CH_4 / graphite	22
	64.4	3.43	31	Équ. (2.2) + #14 + #2	23
	64.4	3.61	34	Équ. (2.2) + #12 + #2	24
	64.4	3.61		Équ. (2.2) + #11 + #2	25

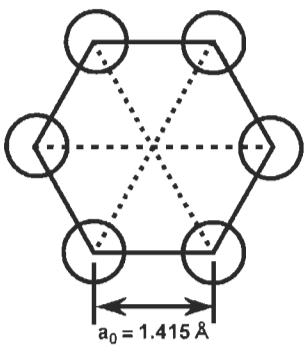
Si l'on néglige les valeurs disparates, ce tableau permet de conclure que l'incertitude maximale présente sur les paramètres de Lennard-Jones tels que choisis ci-haut est de 5.6 % pour ε_{C-H_2}, 1.3 % pour σ_{C-H_2}, 2.5 % pour ε_{C-CH_4} et 5.0 % pour σ_{C-CH_4}. C'est donc dire que les valeurs que j'ai utilisées sont fiables, malgré la grande diversité de valeurs présentes dans la littérature.

2.2. Le potentiel du pore en fente.

Le potentiel d'adsorption d'un adsorbant quelconque est généralement obtenu en supposant que l'interaction entre une molécule d'adsorbat et la surface est donnée en additionnant paire par paire l'interaction entre la molécule d'adsorbat et chacun des atomes de carbone formant la surface :

$$V(\vec{r}) = \sum_j u(\vec{r} - \vec{R}_j) \tag{2.3}$$

Où u(r) est le potentiel de LJ 12-6, $\vec{r}$ est la position de la molécule d'adsorbat et $\vec{R}_j$ est celle de l'atome de carbone. De plus, lorsque la température est suffisamment élevée, l'énergie cinétique des molécules d'adsorbat ($\propto$ kT) est assez grande pour que les molécules puissent passer aisément d'un site d'adsorption à l'autre sur la surface. C'est ce qu'on appelle de l'adsorption délocalisée. Dans ce cas, il est d'usage de considérer que la surface forme un continuum ayant une densité surfacique $\theta = 0.38$ Å^{-2} d'atomes de carbone. Le calcul de cette densité surfacique est montré ci-dessous pour une feuille de graphène.



Chaque hexagone de la feuille comporte $N_C = 6\ (1/3)$ atomes de C.

Chaque hexagone comporte 6 triangles d'aire $A_T = \frac{1}{2}\frac{\sqrt{3}}{2}a_0^{\,2} = 0.8670$ Å

Donc : $\theta = 2/6A_T = 0.38$ Å^{-2}

Figure 2.2. Densité surfacique d'atomes dans une feuille de graphène et un SWNT.

Alors, pour une simple feuille de graphène, le potentiel d'adsorption est simplement donné en intégrant le potentiel LJ 12-6 sur la surface :

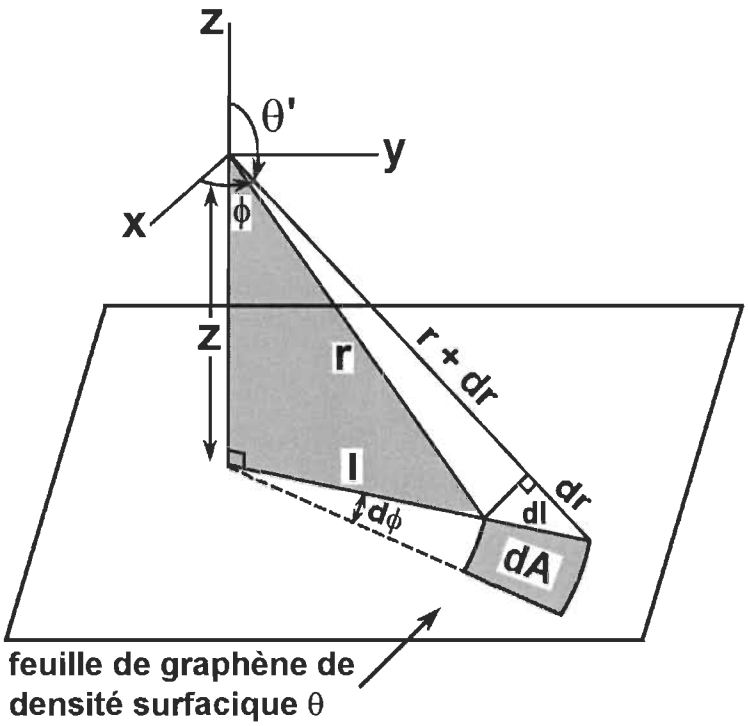


Figure 2.3. Calcul du potentiel d'adsorption d'une feuille de graphène.

De la Fig. 2.3, $dl = \frac{r}{l} dr$ et $dA = (l\, d\phi)\, dl = r\, d\phi\, dr$

$$V_{flat}(z) = \int_z^\infty \int_0^{2\pi} \theta\, u(r)\, r\, d\phi\, dr$$

$$= 8\pi\theta\varepsilon \int_z^\infty r \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] dr$$

Qui en introduisant les quantités réduites $r^* = r/\sigma$ et $z^* = z/\sigma$ devient :

$$V_{flat}(z) = 8\pi\theta\varepsilon\sigma^2 \int_{z^*}^\infty \left[r^{*-11} - r^{*-5} \right] dr^*$$

$$= 8\pi\theta\varepsilon\sigma^2 \lim_{K\to\infty} \left[-\frac{r^{*-10}}{10} + \frac{r^{*-4}}{4} \right]_{z^*}^{K}$$

$$V_{flat}(z) = 2\pi\theta\varepsilon\sigma^2 \left[\frac{2}{5}\left(\frac{\sigma}{z}\right)^{10} - \left(\frac{\sigma}{z}\right)^{4} \right] \qquad (2.4)$$

L'allure de ce potentiel est la même que celle du LJ 12-6, mais son zéro et son minimum sont situés à des valeurs différentes :

$$0 = V_{flat}(z) = 2\pi\theta\varepsilon\sigma^2\left[\frac{2}{5}\left(\frac{\sigma}{z}\right)^{10} - \left(\frac{\sigma}{z}\right)^{4}\right] \quad \Rightarrow \quad z_0 = (2/5)^{1/6}\,\sigma = 0.8584\sigma \tag{2.5}$$

$$0 = V'_{flat}(z^*) = 2\pi\theta\varepsilon\sigma^2\left[\frac{2}{5}(-10)z^{*-11} - (-4)z^{*-5}\right]$$

$$\Rightarrow \quad z_{\min} = \sigma \text{ et } V_{\min} = -\frac{6}{5}\pi\theta\varepsilon\sigma^2 = -\frac{6}{5}\varepsilon_s' \tag{2.6}$$

Où $\varepsilon_s' = \pi\theta\varepsilon\sigma^2$ vaut 371 K (3.08 kJ/mol) pour l'hydrogène et 1021 K (8.49 kJ/mol) pour le méthane.

Le potentiel d'un pore en fente, formé de deux feuilles de graphène espacées d'une distance d est alors donné par :

$$V_{slit}(z,d) = V_{flat}(z + d/2) + V_{flat}(z - d/2) \tag{2.7}$$

Le puits de potentiel dans ce pore a sa profondeur maximale lorsque $d = 2\sigma$ et vaut alors $2.4\varepsilon_s'$. L'allure de ce potentiel est donnée à la figure suivante, pour quelques largeurs typiques du pore :

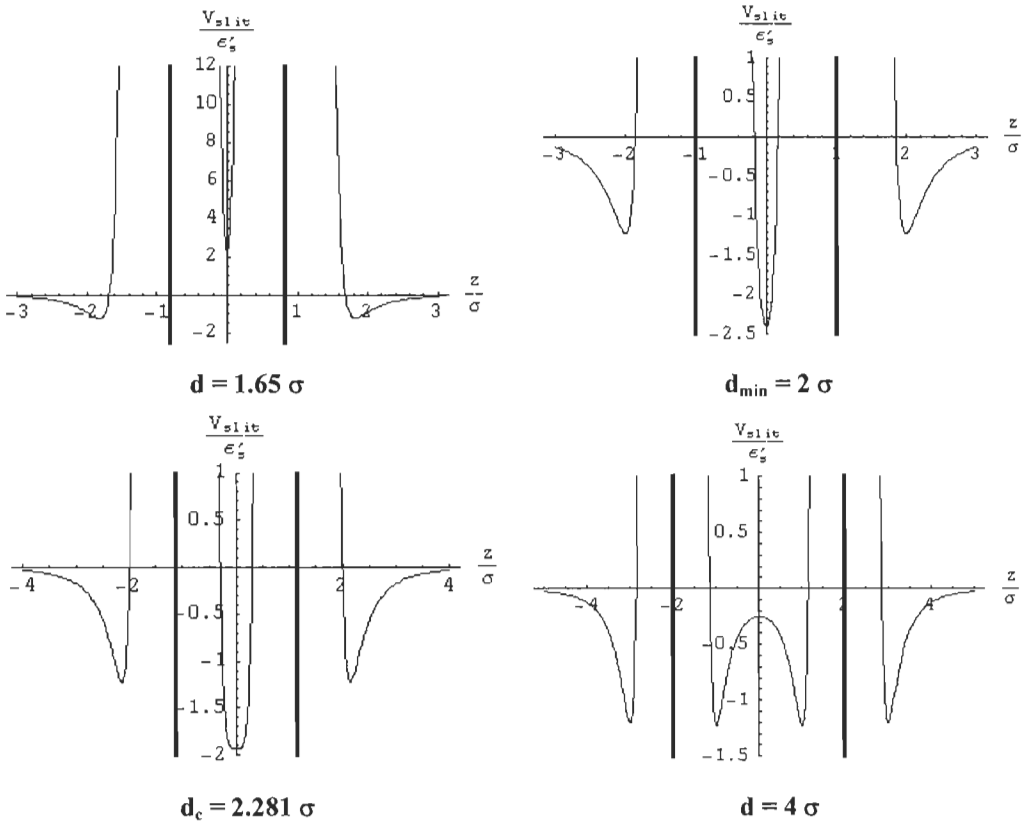


Figure 2.4. Comportement du potentiel d'un pore en fente selon sa largeur.

Comme le laisse voir la figure précédente, il y a une largeur de pore d_0 où le potentiel devient répulsif dans tout l'intérieur du pore. Cela se produit lorsque $d/\sigma < 2\,(2/5)^{1/6} = 1.717$. De plus, il y a une distance critique d_c où le minimum cesse d'être au centre du pore et se divise en deux. Ceci se produit lorsque la concavité de V_{slit} (i.e. la constante de force de l'approximation harmonique) change au centre du pore :

$$V''_{flat}(z) = 2\varepsilon_s'\left\{\frac{2}{5}(-10)(-11)\left(\frac{z}{\sigma}\right)^{-12}\frac{1}{\sigma^2} - (-4)(-5)\left(\frac{z}{\sigma}\right)^{-6}\frac{1}{\sigma^2}\right\}$$

$$= 8\pi\theta\varepsilon\left(\frac{\sigma}{z}\right)^6\left\{11\left(\frac{\sigma}{z}\right)^6 - 5\right\}$$

$$0 = V''_{slit}(0,d) = V''_{flat}(d/2) + V''_{flat}(-d/2)$$

$$= 2\cdot 8\pi\theta\varepsilon\cdot 2^6\left(\frac{\sigma}{d}\right)^6\left\{11\cdot 2^6\left(\frac{\sigma}{d}\right)^6 - 5\right\}$$

$$= 1024\pi\theta\varepsilon\left(\frac{\sigma}{d}\right)^6\left\{704\left(\frac{\sigma}{d}\right)^6 - 5\right\}$$

$$\Rightarrow \quad d_c = (704/5)^{1/6}\sigma = 2.281\sigma \text{ et } V_{slit}(0,d_c) = -1.935\varepsilon_s' \tag{2.8}$$

La figure suivante montre la position et la valeur des minimums de V_{slit} à l'intérieur du pore en fonction de sa largeur :

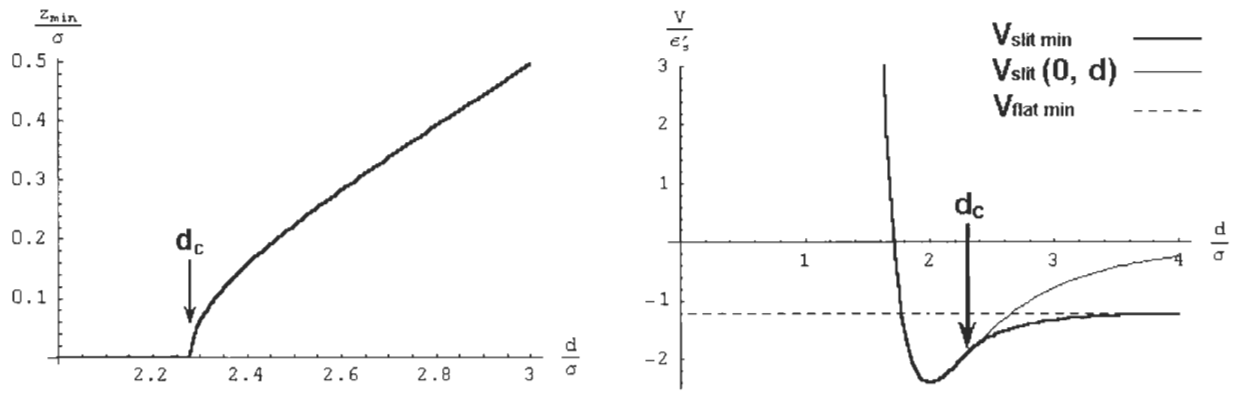


Figure 2.5. Comportement des minimums de potentiel à l'intérieur d'un pore en fente selon sa largeur.

2.3. Le potentiel d'un nanotube à paroi simple (SWNT).

De manière similaire au cas de la simple feuille de graphène, le potentiel d'adsorption d'un SWNT infiniment long s'obtient en intégrant le potentiel LJ 12–6 sur la surface cylindrique du tube. Ainsi, le potentiel d'une molécule M située à une distance r = Rx de l'axe du nanotube sera calculé selon la figure suivante :

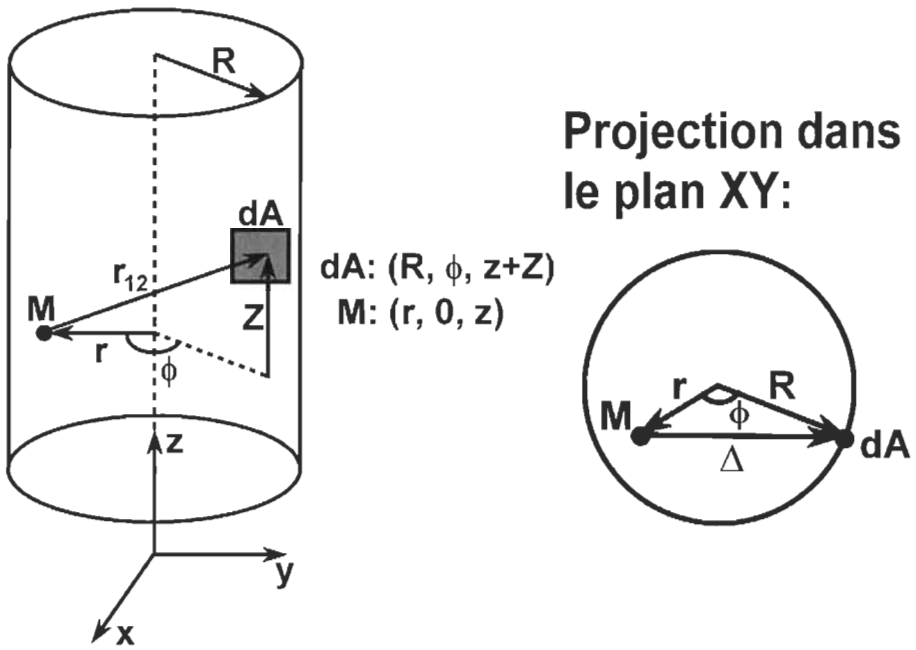


Figure 2.6. Calcul du potentiel d'adsorption d'un nanotube à paroi simple.

La contribution dV au potentiel total obtenu au point M, due à l'élément de surface $dA = R\, d\phi\, dZ$ qui contient $\theta\, dA$ atomes de carbone, est donnée par:

$$dV = u(r_{12})\,\theta\, dA = 4\theta\varepsilon \left\{ (\sigma/r_{12})^{12} - (\sigma/r_{12})^{6} \right\} R\, d\phi\, dZ$$

Alors, le potentiel total à une distance r de l'axe du nanotube s'obtient en intégrant les contributions dV sur toute la surface cylindrique du tube:

$$V(r,R) = \int_{\phi=0}^{2\pi} \int_{Z=-\infty}^{\infty} \left[4\theta\varepsilon \left(\left(\frac{\sigma}{r_{12}} \right)^{12} - \left(\frac{\sigma}{r_{12}} \right)^{6} \right) \right] R\, d\phi\, dZ \qquad (2.9)$$

La projection Δ dans le plan XY de la distance r_{12} entre M et dA est obtenue de la loi du cosinus:

$$\Delta^2 = r^2 + R^2 - 2rR\cos\phi$$

de sorte que la distance r_{12} est finalement donnée par:

$$r_{12} = \sqrt{\Delta^2 + Z^2} = \sqrt{r^2 + R^2 + Z^2 - 2rR\cos\phi}\ .$$

Définissant x = r/R et Z = Δ sh u, nous obtenons $\Delta^2 = R^2\left[1 + x^2 - 2x\ \cos\phi\right]$ et :

$$V = \int_{\phi=0}^{2\pi} 4\theta\varepsilon R\, d\phi \int_{u=-\infty}^{\infty} \Delta ch u\, du \left\{ \left(\frac{\sigma}{\Delta ch\, u}\right)^{12} - \left(\frac{\sigma}{\Delta ch\, u}\right)^{6} \right\} = \int_0^{2\pi} \frac{4\pi\theta\varepsilon R\, d\phi}{8} \left[\frac{63}{32}\frac{\sigma^{12}}{\Delta^{11}} - 3\frac{\sigma^6}{\Delta^5} \right].$$

Puisque cos ϕ est paire et périodique de période 2π, le dernier intégrand l'est aussi, ce qui permet d'avoir des intégrales de 0 à π et d'obtenir finalement :

$$V(r,R) = 3\pi\theta\varepsilon\sigma^2 \left[\frac{21}{32}\left(\frac{\sigma}{R}\right)^{10} M_{11}(x) - \left(\frac{\sigma}{R}\right)^4 M_5(x) \right] \tag{2.10}$$

$$M_n(x) = \int_0^{\pi} \frac{d\phi}{\left(1 + x^2 - 2x\cos\phi\right)^{n/2}}\ . \tag{2.11}$$

L'allure de ce potentiel (V* = V/ε_s') est donnée ci-dessous pour plusieurs rayons réduits (R* = R/σ) du SWNT. Le potentiel est donné en fonction de la distance $z^* = z/\sigma = |r - R|/\sigma$ de la surface, ce qui permet de comparer le potentiel du SWNT avec celui d'une feuille de graphène ($V^*_{flat}(z^*) = V_{flat}(z^*\sigma)/\varepsilon_s$').

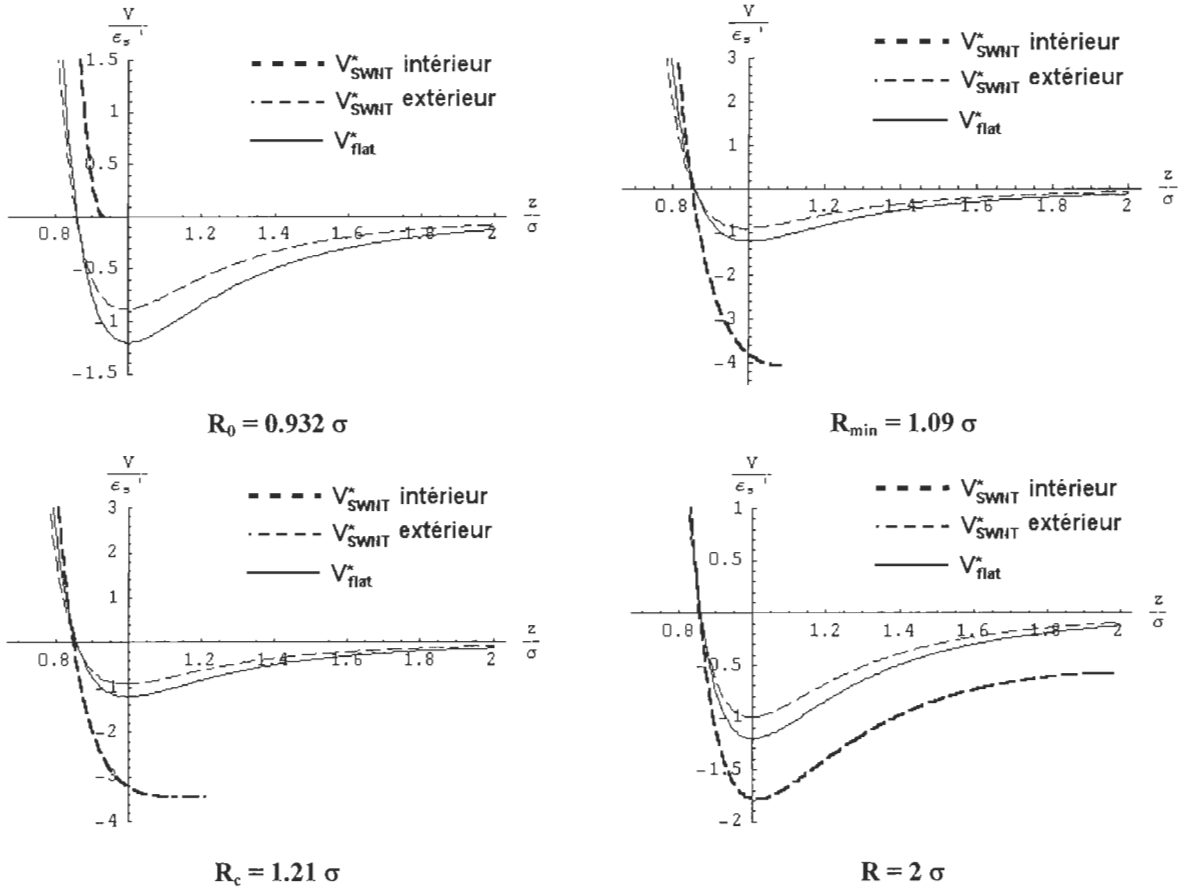


Figure 2.7. Comportement du potentiel d'un SWNT selon son rayon.

Sur ces graphiques, on constate que la courbure du nanotube a pour effet de creuser le potentiel à l'intérieur du tube et à l'atténuer à l'extérieur, par rapport à la surface plane. Ceci est dû au fait que les atomes voisins de carbone deviennent plus rapprochées de la molécule d'adsorbat à l'intérieur du tube et inversement à l'extérieur. Ainsi, le rayon R_{min} correspondant au puits de potentiel le plus profond dans le tube donnera une énergie d'interaction beaucoup plus forte que celles obtenues pour une feuille de graphène et pour un pore en fente de largeur optimale ($d_{min} = 2\sigma$) :

$$V_{\min}(R \le R_c) = V(0,R) = 3\varepsilon_s' \left[\frac{21}{32}\left(\frac{\sigma}{R}\right)^{10} M_{11}(0) - \left(\frac{\sigma}{R}\right)^{4} M_5(0) \right]$$ (le potentiel sur l'axe).

Et puisque $M_n(0) = \int_0^{\pi} \frac{d\phi}{(1)^{n/2}} = \pi$:

$$V(0,R) = 3\pi\varepsilon_s' \left[\frac{21}{32}\left(\frac{\sigma}{R}\right)^{10} - \left(\frac{\sigma}{R}\right)^{4} \right] \tag{2.12}$$

Dont le minimum est déterminé aisément :

$$0 = V_R'(0,R) = 3\pi\varepsilon_s' \left(\frac{1}{\sigma}\right) \left[\frac{21}{32}(-10)\left(\frac{R}{\sigma}\right)^{-11} - (-4)\left(\frac{R}{\sigma}\right)^{-5} \right]$$

$$\Rightarrow \quad R^*_{\min} = (105/64)^{1/6} = 1.086 \quad \text{et} \quad V^*_{\min} = -\frac{48\pi}{5}\left(\frac{3}{1225}\right)^{1/3} = 4.065 \tag{2.13}$$

$$\text{Ainsi, } \frac{V_{\min,\,SWNT}}{V_{\min,\,flat}} = \frac{4.065\varepsilon_s'}{1.2\varepsilon_s'} = 3.39 \text{ et } \frac{V_{\min,\,SWNT}}{V_{\min,\,slit}} = \frac{4.065\varepsilon_s'}{2.4\varepsilon_s'} = 1.69 \tag{2.14}$$

Il est intéressant de constater que l'énergie maximale d'interaction d'une molécule d'hydrogène avec un SWNT, trouvée à l'Équ. (2.13), est de 12.5 kJ/mol. Ce résultat est de 36 % inférieur à la très forte énergie d'activation de 19.6 kJ/mol trouvée par Dillon et al. [4] pour des nanotubes ayant un diamètre non optimal de 12 Å (i.e. $R^* = 1.88 > R^*_{min}$).

Comme obtenu précédemment pour le pore en fente, il y a un certain rayon R_0 à partir duquel le potentiel devient répulsif dans tout le nanotube. Cela se produit lorsque le potentiel V(0, R) devient nul sur l'axe du tube, à $R_0 = (21/32)^{1/6}\sigma = 0.9322\sigma$. De plus, comme pour le pore en fente, il y a un certain rayon "critique" R_c à partir duquel le minimum situé sur l'axe du tube commence à se séparer de manière continue (cf. Fig. 2.8). Cela se produit lorsque la constante de force $\alpha(R)$ de l'approximation harmonique du potentiel près du centre du tube change de signe. En effet, développant V(r, R) en série de Taylor pour r petit :

$$V(r,R) = V(0,R) + V'(0,R)r + \frac{V''(0,R)}{2}r^2 + \frac{V^{(3)}(0,R)}{3!}r^3 + \frac{V^{(4)}(0,R)}{4!}r^4 \tag{2.15}$$

$$\text{avec : } V^{(m)}(0,R) = 3\pi\theta\varepsilon\sigma^2 \left[\frac{21}{32}\left(\frac{\sigma}{R}\right)^{10} R^{-m} M_{11}^{(m)}(0) - \left(\frac{\sigma}{R}\right)^{4} R^{-m} M_5^{(m)}(0) \right] \tag{2.16}$$

$$M'_n(0) = n\int_0^{\pi} \cos\phi \, d\phi = 0 \quad ; \quad M''_n(0) = \frac{n^2\pi}{2}$$
$$M_n^{(3)}(0) = 0 \quad ; \quad M_n^{(4)}(0) = \frac{3\pi}{8}[n(n+2)]^2 \tag{2.17}$$

$V'(0,R)=0$ des Équs. (2.16) et (2.17 a).

$$\begin{aligned}\alpha(R)&=\frac{V''(0,R)}{2}=\frac{3}{2}\pi\theta\varepsilon\frac{\sigma^2}{R^2}\left[\frac{21}{32}\left(\frac{\sigma}{R}\right)^{10}M_{11}^{(2)}(0)-\left(\frac{\sigma}{R}\right)^{4}M_5^{(2)}(0)\right]\\&=\frac{3}{4}\pi^2\theta\varepsilon\left[\frac{2541}{32}\left(\frac{\sigma}{R}\right)^{12}-25\left(\frac{\sigma}{R}\right)^{6}\right]\end{aligned}\tag{2.18}$$

$V^{(3)}(0,R)=0$ des Équs. (2.16) et (2.17 c).

$$\begin{aligned}\beta(R)&=\frac{V^{(4)}(0,R)}{24}=\frac{3}{24}\pi\theta\varepsilon\frac{\sigma^2}{R^4}\left[\frac{21}{32}\left(\frac{\sigma}{R}\right)^{10}M_{11}^{(4)}(0)-\left(\frac{\sigma}{R}\right)^{4}M_5^{(4)}(0)\right]\\&=\frac{3}{24}\pi^2\theta\varepsilon\frac{1}{R^2}\left[\frac{21}{32}\cdot\frac{61347}{8}\left(\frac{\sigma}{R}\right)^{12}-\frac{3675}{8}\left(\frac{\sigma}{R}\right)^{6}\right]\\&=\frac{3675}{64}\pi^2\theta\varepsilon\frac{1}{R^2}\left[\frac{61347}{5600}\left(\frac{\sigma}{R}\right)^{12}-\left(\frac{\sigma}{R}\right)^{6}\right]\end{aligned}\tag{2.19}$$

Ainsi : $V(r,R)\approx V(0,R)+\alpha(R)r^2+\beta(R)r^4$ (2.20)

Alors le rayon critique du nanotube et le potentiel sur l'axe qui y correspond seront finalement donnés par :

$$R_c=(2541/800)^{1/6}\sigma=1.212\sigma\tag{2.21}$$

$$V(0,R_c)=-\frac{3840\pi}{1331}\left(\frac{30}{539}\right)^{1/3}\varepsilon_s'=-3.461\varepsilon_s'\tag{2.22}$$

La position du minimum de potentiel à l'intérieur du nanotube est donnée à la figure ci-dessous en fonction de son rayon. La position du minimum est donnée en terme de sa distance par rapport à l'axe du tube (r_{min}) ou en terme de sa distance par rapport à la paroi (z_{min}).

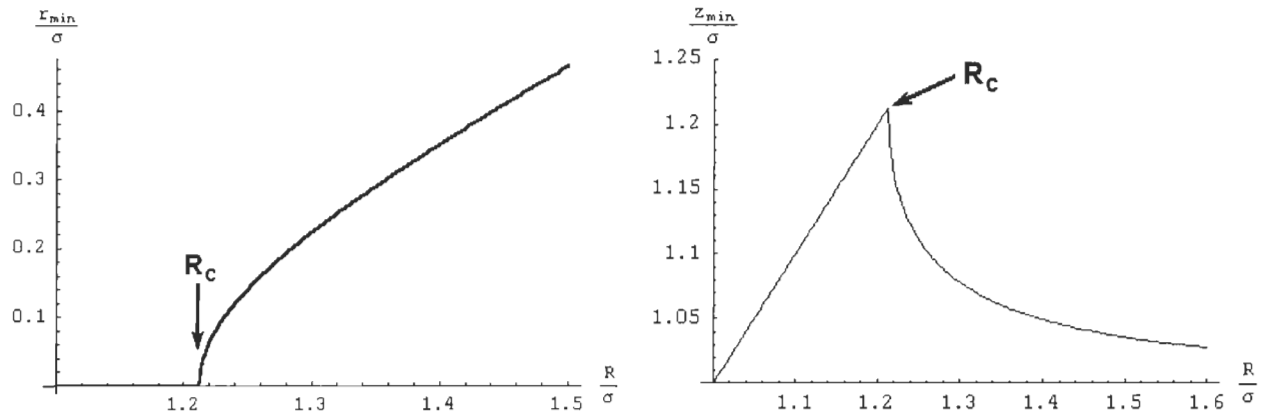


Figure 2.8. Position du minimum de potentiel à l'intérieur d'un SWNT en fonction de son rayon.

La figure 2.8 montre que lorsque le rayon du tube est inférieur au rayon critique R_c, le minimum de potentiel reste sur l'axe du tube, tandis que lorsqu'il continue d'augmenter au delà de R_c, le minimum se rapproche de la surface jusqu'à ce qu'il se stabilise à une distance σ de la surface, correspondant ainsi à une feuille plane de graphène ($R = \infty$). Le minimum de potentiel coïncide alors avec celui du plan de graphite tandis que l'énergie d'interaction avec la molécule d'adsorbat devient nulle sur l'axe du tube (Fig. 2.9). On voit également sur la figure 2.8 que l'augmentation de r_{min} près de R_c est beaucoup plus brutale que lorsque $R >> R_c$, ce qui se traduit par la partie courbée sur le graphique de gauche. Il est possible d'obtenir une expression décrivant ce comportement, pour $R \gtrsim R_c$ en développant à nouveau l'expression (2.20) en série de Taylor pour R autour de R_c (on garde alors r fixé) :

$$V(r,R) = \left\{V(0,R_c) + \alpha(R_c)r^2 + \beta(R_c)r^4\right\} + \left[\frac{\partial}{\partial R}\left\{V(0,R) + \alpha(R)r^2 + \beta(R)r^4\right\}\right|_{R_c} (R - R_c) + O(R - R_c)^2 \tag{2.23}$$

Qui devient, en négligeant des petits termes en $(R - R_c)$ et en utilisant l'équation $\alpha(R_c) = 0$:

$$\begin{aligned} V(r,R) &\approx V(0,R_c) + \beta(R_c)r^4 + \left[V'(0,R_c) + \alpha'(R_c)r^2 + \beta'(R_c)r^4\right](R - R_c) \\ &\approx \left\{V(0,R_c) + V'(0,R_c)(R - R_c)\right\} + \alpha'(R_c)(R - R_c)r^2 + \left[\beta(R_c) + \beta'(R_c)(R - R_c)\right]r^4 \\ V(r,R) &\approx V(0,R_c) + \alpha'(R_c)(R - R_c)r^2 + \beta(R_c)r^4 \end{aligned} \tag{2.24}$$

La position r_{min} du minimum est alors finalement obtenue comme suit :

$$0 = \frac{\partial}{\partial r}V(r,R) = 2\alpha'(R_c)(R - R_c)r + 4\beta(R_c)r^3$$

$$r_{\min} = \sqrt{\frac{1}{2}\left|\frac{\alpha'(R_c)}{\beta(R_c)}\right|(R - R_c)} \tag{2.25}$$

Dont l'allure pour $R \gtrsim R_c$ est celle d'une demie parabole couchée sur l'axe des R et débutant à R_c. Cela explique donc la partie courbée présente sur le graphique de gauche de la figure 2.8.

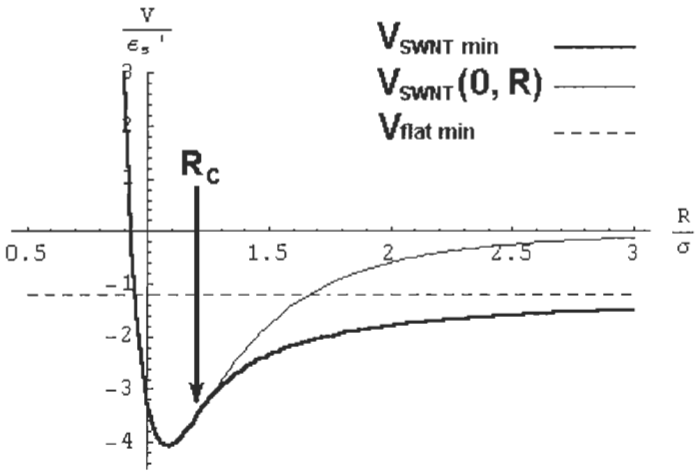


Figure 2.9. Comportement du minimum de potentiel à l'intérieur d'un SWNT selon son rayon.

Comme le laisse voir la figure 2.9 et la partie droite de la figure 2.8, on s'attend à ce que le comportement du potentiel V(r, R) d'un SWNT devienne identique à celui du plan de graphite lorsque $R \to \infty$ (pendant que la distance z de la molécule d'adsorbat reste fixe) ou lorsque $r \to R$. En effet, dans ces conditions se traduisant par $x \to 1$, le tube apparaîtra comme une surface plane pour la molécule d'adsorbat et on devrait avoir que l'Équ. (2.10) du potentiel d'un SWNT se réduit à l'Équ. (2.4) pour une feuille de graphène. Puisque la présence de la paroi du nanotube en $x = r/R = 1$ fait en sorte que le potentiel diverge lorsque $x \to 1$, les fonctions $M_n(x)$ possèderont une singularité à x = 1. Dans le but de connaître la forme de la divergence de ces fonctions lorsque $x \to 1$, nous avons développé les fonctions $M_n(x)$ en séries autour de la singularité en x = 1 puis nous n'avons gardé que le terme divergeant le plus rapidement dans ces séries. Les séries obtenues ne sont ni des séries de Taylor, ni des séries de Laurent (cf. annexe 2 pour les détails des développements) :

$$\begin{aligned} M_5(x) &\approx \frac{2}{3}(x-1)^{-4} + \ldots \qquad \text{lorsque} \quad x \to 1 \\ M_{11}(x) &\approx \frac{128}{315}(x-1)^{-10} + \ldots \qquad \text{lorsque} \quad x \to 1 \end{aligned} \tag{2.26}$$

Qui donnent alors pour le potentiel du SWNT :

$$V(r,R) \approx 3\varepsilon_s' \left[\frac{21}{32}\left(\frac{\sigma}{R}\right)^{10} \left\{ \frac{128}{315}\left(\frac{R}{r-R}\right)^{10} \right\} - \left(\frac{\sigma}{R}\right)^{4} \left\{ \frac{2}{3}\left(\frac{R}{r-R}\right)^{4} \right\} \right]$$

Qui se réduit finalement à (2.4) en utilisant la définition $z = |r - R|$:

$$V(r,R) \approx 3\varepsilon_s' \left[\frac{21}{32} \cdot \frac{128}{315}\left(\frac{\sigma}{z}\right)^{10} - \frac{2}{3}\left(\frac{\sigma}{z}\right)^{4} \right] = V_{flat}(z) \quad \text{pour} \quad r \to R \tag{2.27}$$

Jusqu'à maintenant, nous avons considéré que la surface de l'adsorbant pouvait être traitée comme un continuum, ce qui a permis d'obtenir des expressions simplifiées (Équs. (2.4) et (2.10)) pour le potentiel d'adsorption. Mais en réalité, il faudrait utiliser une sommation explicite (2.3) pour calculer le potentiel de la feuille de graphène et du SWNT. Ce calcul a été effectué[1] pour un nanotube (12, 7) de rayon 6.515 Å afin de pouvoir vérifier l'exactitude de l'approximation d'un nanotube discret par un cylindre continu ayant une densité surfacique uniforme θ d'atomes de carbone. Nous avons généré la distribution d'atomes de carbone en utilisant le programme de M. Brandbyge [37]. Le nanotube utilisé avait une longueur de 128.193 Å afin de pouvoir l'assimiler à un tube infiniment long (Fig. 2.10). Voici donc l'essentiel des calculs effectués sur Mathematica[2] :

Nous avons calculé les positions des atomes de carbone du SWNT et nous les avons placés dans une matrice ListeTube = {{x1, y1, z1}, {x2, y2, z2}, ...} réorganisée pour être utilisable facilement dans les calculs :

Forme obtenue dans le notebook:

$$\begin{pmatrix} \begin{pmatrix} 6.51492 \\ 0. \\ 0. \end{pmatrix} & \begin{pmatrix} 6.36424 \\ 0.213299 \\ -1.39305 \end{pmatrix} \\ \begin{pmatrix} -5.72042 \\ 0.127979 \\ -3.11785 \end{pmatrix} & \begin{pmatrix} -6.25479 \\ 0.341278 \\ -1.82257 \end{pmatrix} \\ \begin{pmatrix} 3.5307 \\ 0.255959 \\ 5.47524 \end{pmatrix} & \begin{pmatrix} 4.61979 \\ 0.469257 \\ 4.59366 \end{pmatrix} \end{pmatrix}$$

Forme réorganisée:

$$\begin{pmatrix} 6.51492 & 0. & 0. \\ 6.36424 & 0.213299 & -1.39305 \\ -5.72042 & 0.127979 & -3.11785 \\ -6.25479 & 0.341278 & -1.82257 \\ 3.5307 & 0.255959 & 5.47524 \\ 4.61979 & 0.469257 & 4.59366 \end{pmatrix}$$

Les points contenus dans cette matrice ont pu alors être tracés pour donner un aperçu du nanotube (12, 7) utilisé pour calculer le potentiel d'adsorption "discret" :

---

[1] Je remercie Benjamin Angers pour ce calcul.
[2] Mathematica version 4.1, commercialisé par Wolfram Research, Inc.

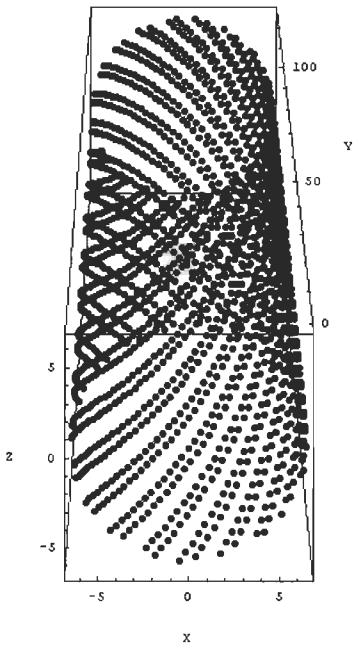


Figure 2.10. Nanotube (12, 7) utilisé dans Mathematica pour calculer le potentiel d'adsorption d'un nanotube discret.

Le potentiel "discret", évalué le long d'une ligne radiale passant par le centre du nanotube (choisie comme étant parallèle à l'axe des x) est alors donné par l'énoncé ci-dessous (où `LJ[x]` = Équ. (2.1) et $Y_{-} = (Y_{\min} + Y_{\max})/2$) et est tracé à la figure 2.11. Le potentiel du nanotube continu correspondant y est également tracé pour fins de comparaison. On peut constater que les deux potentiels coïncident de manière remarquable.

```
VSWNTdiscret[r_, Y_] := Sum[
  LJ[√((r - ListeTube[[i, 1]])^2 + (Y - ListeTube[[i, 2]])^2 + (0 - ListeTube[[i, 3]])^2)],
  {i, 1, Length[ListeTube]}
 ]
(* la ligne radiale où on évalue V(r) est située à "Y" *)
```

(2.28)

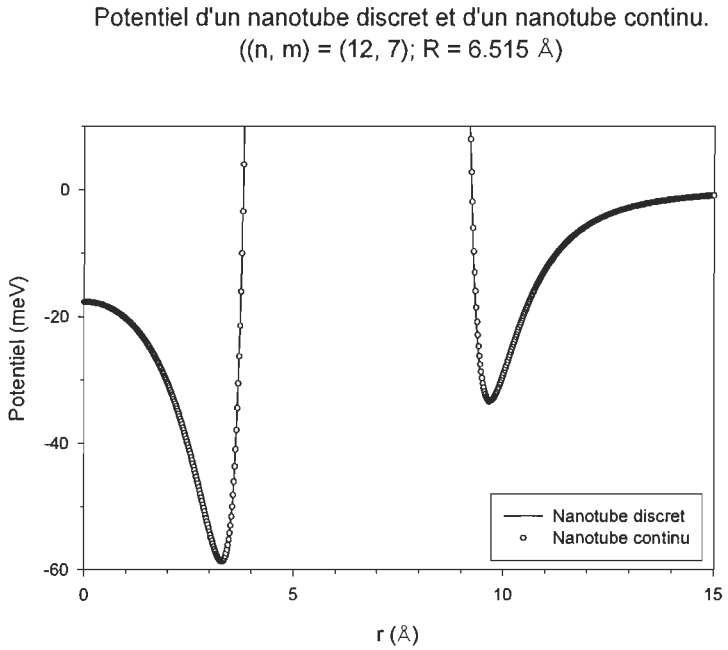


Figure 2.11. Comparaison des potentiels d'adsorption d'un nanotube (12, 7) discret et continu.

Malgré que les deux potentiels d'adsorption coïncident de manière remarquable lorsqu'on les évalue le long d'une ligne radiale (où z = 0) passant au milieu de la longueur du tube, on sait que le potentiel "discret" sera différent selon la position y de cette ligne radiale le long du tube. Le potentiel sera donc ondulé ("corrugé") à cause du réseau d'hexagones présent à la surface du tube. Cela est visible sur la figure 2.23 (pour un nanotube (13, 0)) où la courbe tiretée représente le potentiel évalué le long d'une ligne radiale passant par le centre d'un hexagone (points S) et la courbe pointillée représente le potentiel évalué le long d'une ligne radiale passant par un atome de carbone (points A). Afin de voir cette "corrugation", nous avons calculé puis tracé le minimum du potentiel "discret" selon la position y de la ligne radiale. Nous avons donc calculé le minimum (situé à l'intérieur du SWNT) au moyen de l'instruction :

```
Vminimum[Y_] := (FindMinimum[VSWNTdiscret[r, Y], {r, 4.0}])[[1]]
```
(2.29)

Où la partie `{r, 4.0}` indique à Mathematica qu'on cherche à minimiser le potentiel par rapport à r en utilisant la valeur 4.0 comme estimé initial du rayon où se trouve le minimum.

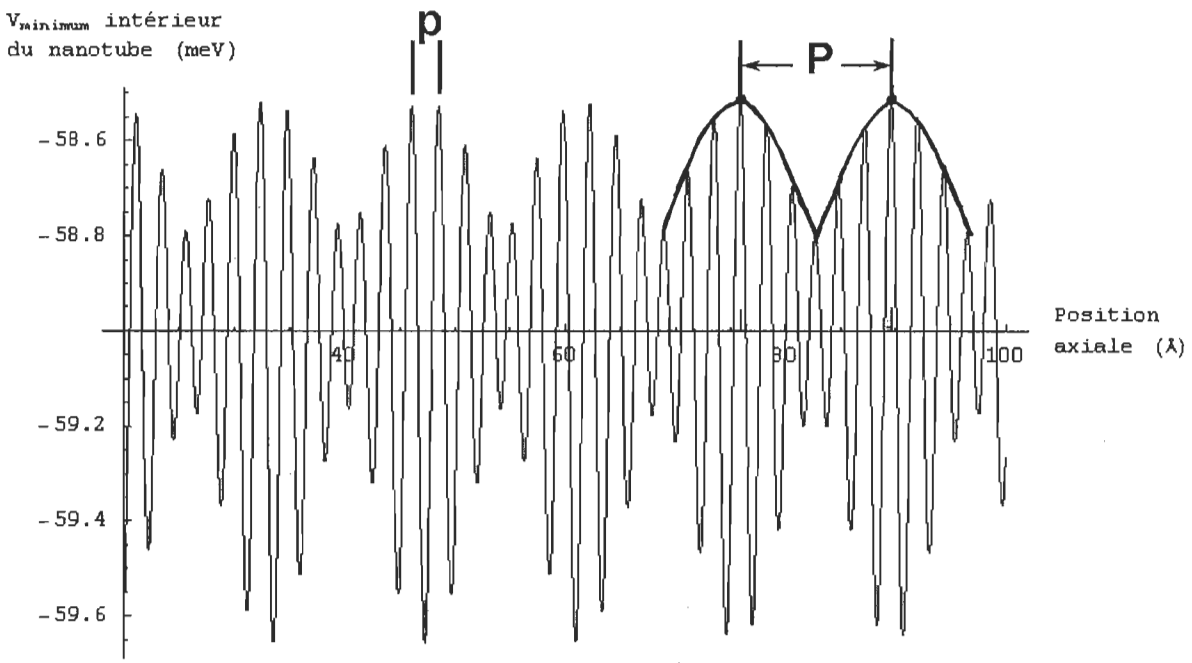


Figure 2.12. Variation du minimum du potentiel d'adsorption à l'intérieur d'un nanotube discret (12, 7) en fonction de la position le long du tube.

Ce graphique est très intéressant car il montre bien la "corrugation" du potentiel dû à un nanotube discret. En effet, pour un nanotube continu équivalent de rayon 6.515 Å, un programme calculant l'équation (2.10) a permis de trouver que le minimum valait –58.74 meV (–681.6 K) alors que le véritable potentiel a un minimum oscillant entre –59.65 meV (–692.2 K) et –58.55 meV (–679.4 K) selon la position où on le calcule le long du tube. Lorsque la température est beaucoup plus élevée que la barrière de potentiel maximale existant entre deux sites d'adsorption voisins (1.10 meV; 12.8 K de la Fig. 2.12), ces petites ondulations peuvent être négligées car l'adsorption est alors délocalisée. Cela sera le cas aux températures expérimentales d'intérêt ($\geqslant$ 77.4 K). De la Fig. 2.12 on peut voir également que l'écart entre le minimum de potentiel d'un SWNT discret et continu est d'au plus 1.5 %, ce qui est très peu. Donc, l'hypothèse du nanotube continu utilisée dans ce travail est acceptable et allégera de beaucoup les calculs numériques (deux intégrales sur la variable polaire Équ. (2.11) au lieu d'une somme sur la surface Équ. (2.3)).

Un autre aspect intéressant de la Fig. 2.12 est qu'elle ressemble au graphique d'une modulation d'amplitude ($f(y) = A\sin(\varpi_1 y)\sin(\varpi_2 y)$), ce qui suggère le lien avec le réseau

d'hexagones formant la surface du nanotube et sa périodicité. En effet, lorsqu'on déplace la ligne radiale le long de l'axe du tube en suivant la ligne pointillée de droite indiquée sur la Fig. 2.13, nous franchissons des régions situées dans des hexagones et d'autres régions situées près des atomes de carbone, ce qui a pour effet de faire varier rapidement le minimum de potentiel sur la figure 2.12. Cela se traduit donc par des petites périodes p valant 2.29 et 2.73 Å (obtenues par transformée de Fourier discrète de la figure 2.12) et qui sont assez similaires à la distance entre deux centres d'hexagones adjacents ($\sqrt{3}\ a_0 = 2.46$Å). Il est également logique d'associer la plus longue période P de "l'enveloppe" à la périodicité des hexagones le long de la ligne pointillée de droite. Lorsque nous nous déplaçons le long de cette ligne à partir d'un hexagone donné, on fini par rencontrer à nouveau un hexagone positionné exactement de la même manière par rapport à cette ligne, ce qui implique une répétition dans le comportement du potentiel d'adsorption. Cette période P est illustrée à la figure ci-dessous et après avoir effectué la transformée de Fourier discrète, sa valeur fut établie à 14.1 Å.

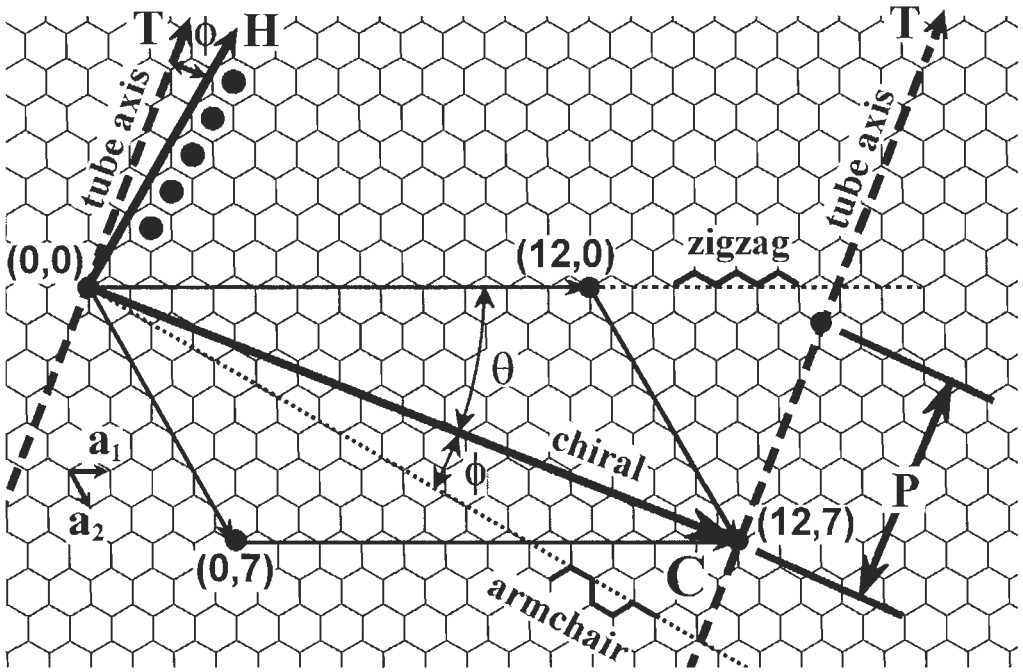


Figure 2.13. Périodicité des hexagones d'un nanotube (12, 7) le long de l'axe du tube.

De la figure précédente, la période P s'obtient aisément :

$$P_{(12,7)} = \left|5\vec{a}_1 - 6\vec{a}_2\right| = \sqrt{5^2 + 6^2 - 2\cdot 5\cdot 6\cos 60°}\,\left|\vec{a}_1\right| \\ = 5.568\cdot\sqrt{3}\,a_0 = 13.69\,\text{Å} \tag{2.30}$$

La période $P_{(12,7)}$ trouvée ci-haut permet donc d'associer la présence de l'enveloppe à la périodicité des hexagones le long de l'axe du tube. L'écart entre les deux valeurs n'est que de 2.8 %.

2.4. Le potentiel d'un faisceau de nanotubes à parois simples.

Le potentiel d'adsorption d'un faisceau de N_t nanotubes de rayon R centrés sur un réseau hexagonal de pas d (cf. Fig 2.14) est lui aussi obtenu au moyen de l'Équ. (2.3), valide pour tout adsorbant. Cependant, pour des nanotubes continus, le calcul se réduit à une simple addition des contributions de chaque nanotube au potentiel total :

$$V(\vec{r}, d, R, N_t) = \sum_{\substack{noeuds \\ (p,q)}}^{N_t} V\left(\left|\vec{r} - \vec{R}_{pq}\right|, R\right) \tag{2.31}$$

Où la position des noeuds du réseau est donnée par :

$$\vec{R}_{pq} = p\vec{a} + q\vec{b} = \frac{d}{2}\left[(2p+q)\hat{x} + \sqrt{3}\,q\,\hat{y}\right] \tag{2.32}$$

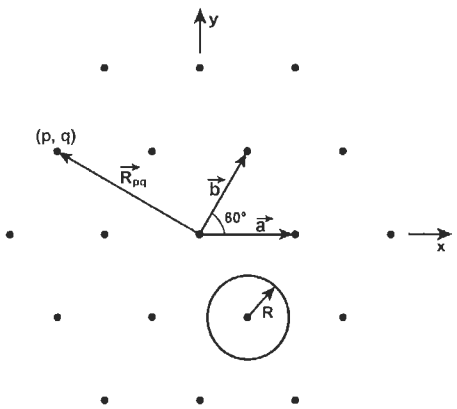


Figure 2.14. Arrangement de 19 SWNTs en faisceau.

Comme nous avons vu à la section précédente, il est commode de travailler avec le potentiel réduit ($V^* = V/\varepsilon_s$') dont le comportement ne dépendra pas de l'adsorbat utilisé. Mais puisque la

plupart de mes programmes sont conçus pour calculer le potentiel non réduit, nous devrons choisir dans ces programmes ε et σ de manière à ce que les deux potentiels coïncident (ce qui sera le cas lorsque $\varepsilon_s' = 1$ et $\sigma = 1$) :

Prenant $\sigma = 1$ et $\varepsilon = 1/\pi\theta = 0.8377$:

$$V \to V^* = V/\varepsilon_s' \qquad \vec{r} \to \vec{r}^* = \vec{r}/\sigma \qquad R \to R^* = R/\sigma \tag{2.33}$$

Le potentiel d'adsorption réduit d'un faisceau de SWNTs a été tracé au moyen de Mathematica pour des valeurs particulières de d, R et N_t correspondant le plus souvent aux minimums et maximums des courbes du second coefficient du viriel B_{AS} tracées en fonction de d et R (cf. Figs. 5.4 et 5.7). Le potentiel d'adsorption au point (x, y) fut calculé au moyen de la fonction suivante :

```
Vbundle[x_, y_, PointsReseau_, R_, Ntubes_, θ_, ϵ_, σ_] :=
 Sum[VTubei[x, y, i, PointsReseau, R, θ, ϵ, σ], {i, 1, Ntubes}]
```
(2.34)

Où **PointsReseau** est la matrice contenant les noeuds du réseau sur ses différentes lignes et où **Vtubei[ ]** sert à calculer la contribution du i ème tube au potentiel total :

```
VTubei[x_, y_, i_, PointsReseau_, R_, θ_, ϵ_, σ_] :=
 VSWNT[((x - PointsReseau[[i, 1]])^2 + (y - PointsReseau[[i, 2]])^2)^(1/2), R, θ, ϵ, σ]
```
(2.35)

Où **VSWNT[ ]** = Équ. (2.10).

La surface et les équipotentielles représentant le potentiel d'adsorption d'un faisceau de SWNTs ayant les dimensions $d^* = 5.235$, $R^* = 2.038$ et $N_t = 19$ (16.7 Å, 6.5 Å et $N_t = 19$ pour l'hydrogène sur des SWNTs de carbone) furent obtenues au moyen des expressions suivantes et sont tracées à la figure ci-dessous.

```
Plot3D[Vbundle[x, y, ReseauConsiderer, 2.038, 19, 0.38`20,
  1/(Pi*0.38`20), 1], {x, -16, 16}, {y, -14, 14}, PlotPoints→ 200,
 PlotRange→ {-5, 2}, AxesLabel→ {x, y, "V_bundle"},
 ViewPoint -> {2.439, 1.141, 2.050}]
```
(surface) (2.36)

```
ContourPlot[Vbundle[x, y, ReseauConsiderer, 2.038, 19,
  0.38`20, 1/(Pi*0.38`20), 1], {x, -15, 15}, {y, -15, 15},
 AspectRatio→ Automatic, PlotPoints→ 400,
 Contours→ {-2.25, -1.75, -1.25, -0.75, -0.25, 0.25, 10.00},
 PlotRange→ {-10, 10}, ContourShading→ False,
 ContourLines→ True, Axes→ True, AxesLabel→ {x, y}]
```

(équipotentielles) (2.37)

Où la valeur du potentiel réduit ($V^* = V/\varepsilon_s$') sur les différentes équipotentielles est donnée par l'option `Contours→ {V*i}`.

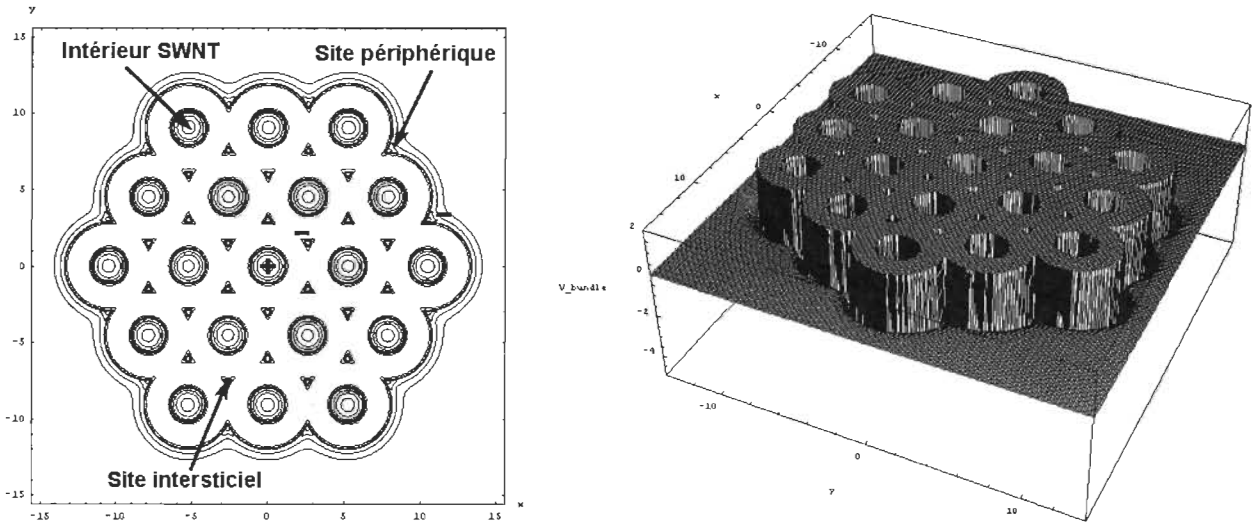
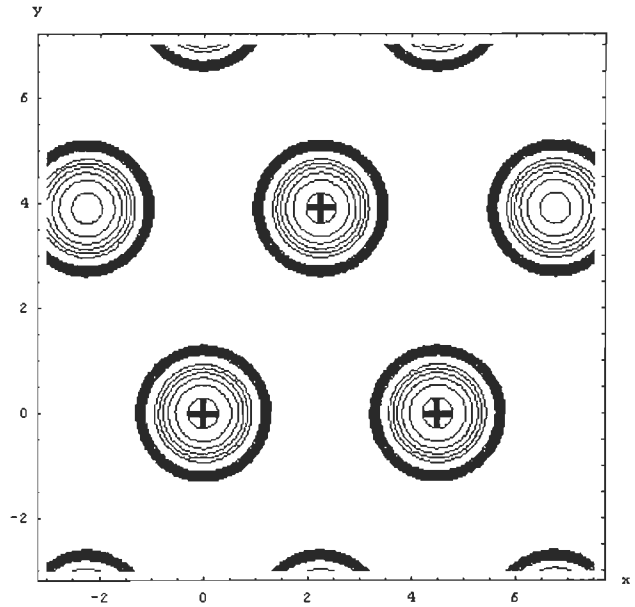
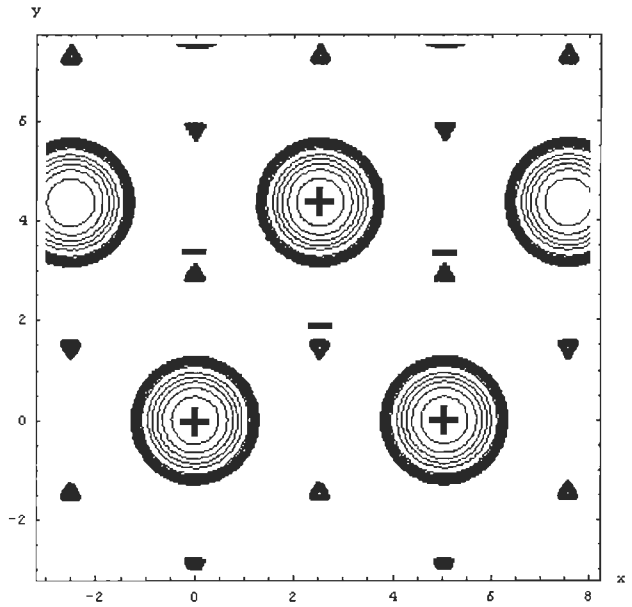


Figure 2.15. Potentiel réduit d'adsorption pour un faisceau de 19 SWNTs ($d^* = 5.235$, $R^* = 2.038$). Les zones de potentiel plus (moins) élevées sont notées avec des signes + (–).

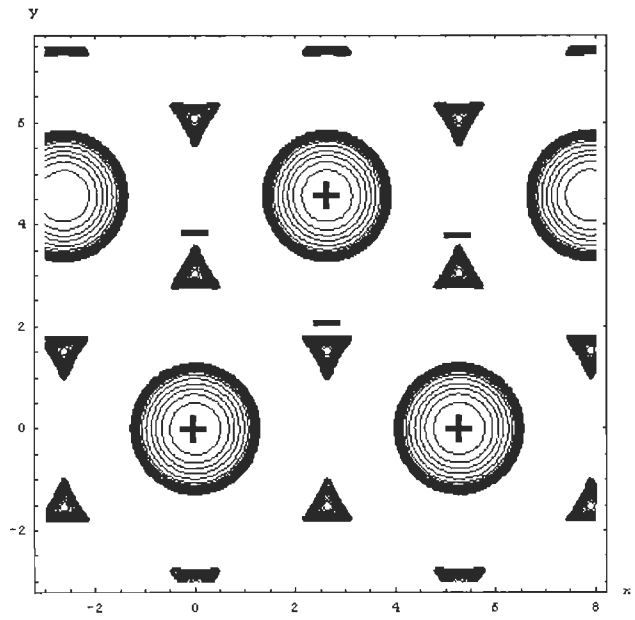
À la figure suivante, nous avons tracé les équipotentielles du potentiel d'adsorption réduit d'un faisceau de SWNTs ayant 25 tubes de rayon réduit $R^* = 2.038$ pour plusieurs séparations d^* entre leurs centres telles qu'indiquées sur la figure 5.4. La différence de potentiel entre deux équipotentielles consécutives sera notée par Δ^* sur les figures suivantes.



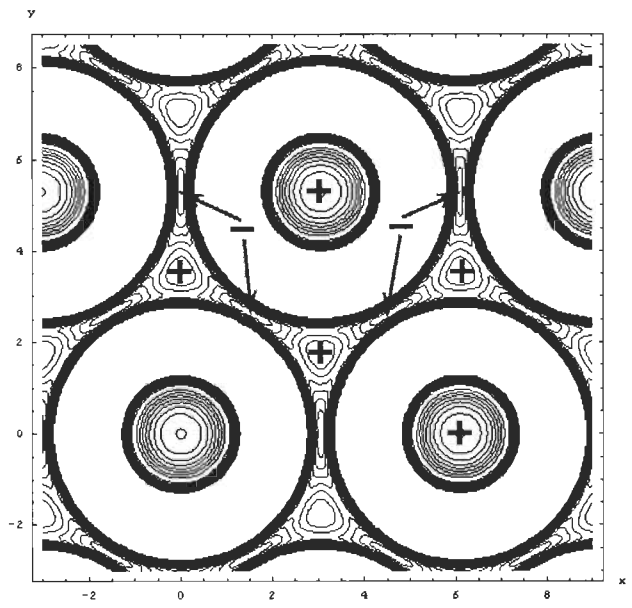
d* = 4.50, Δ* = 0.247



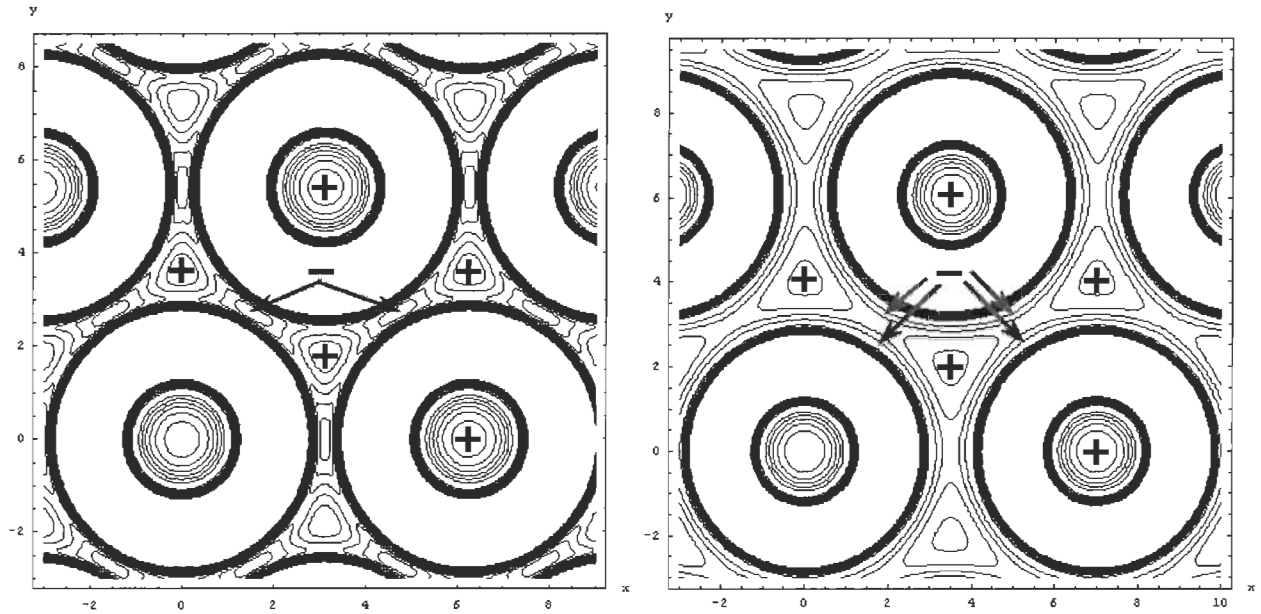
d* = 5.04, Δ* = 0.247



d* = 5.27, Δ* = 0.247



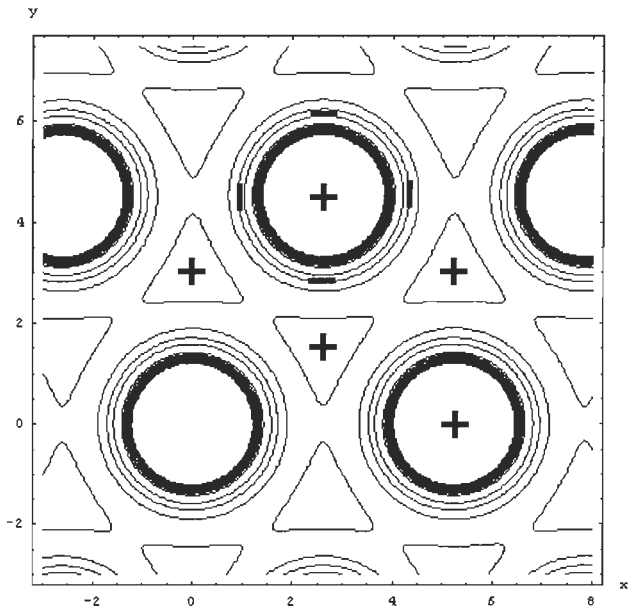
d* = 6.11, Δ* = 0.165



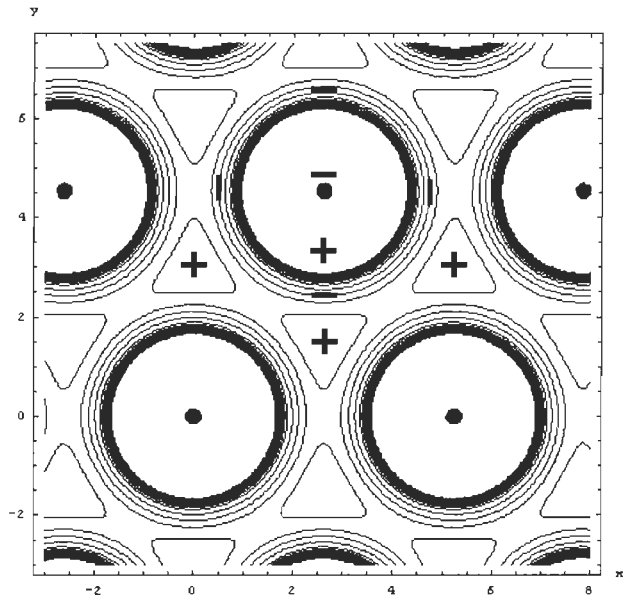
d* = 6.25, Δ* = 0.198 **d* = 7.00, Δ* = 0.220**

Figure 2.16. Équipotentielles du potentiel d'adsorption réduit d'un faisceau de 25 SWNTs de rayon réduit 2.038 pour plusieurs séparations entre leurs centres. Les zones de potentiel plus (moins) élevées sont notées avec des signes + (–) et les valeurs des équipotentielles sont comprises entre –10 et 10.

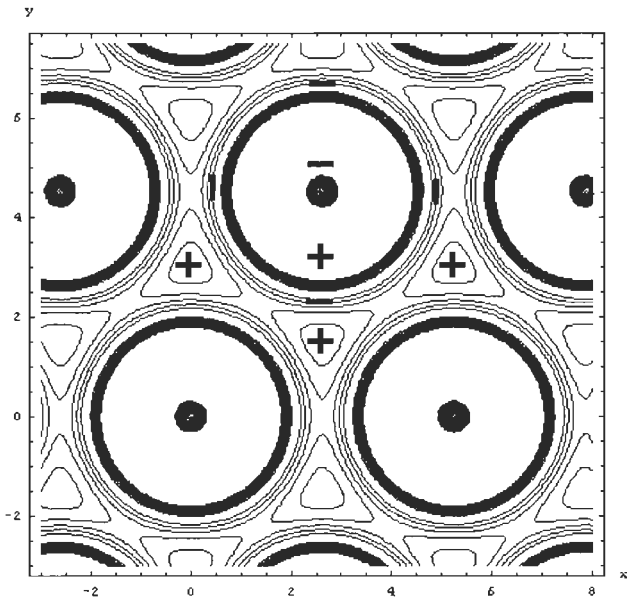
Ci-dessous, nous avons tracé les équipotentielles pour un faisceau de 25 SWNTs ayant leurs centres séparés d'une distance d* = 5.235 pour plusieurs rayons R* des nanotubes tels qu'indiqués sur la figure 5.7.



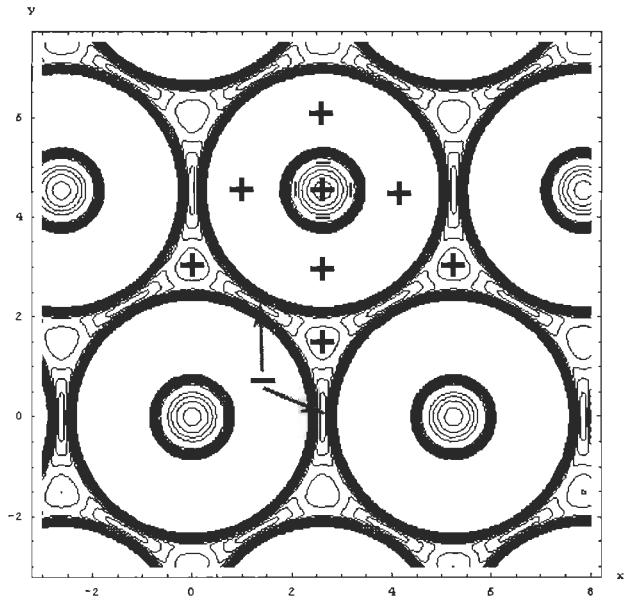
R* = 0.50, Δ* = 0.198



R* = 0.94, Δ* = 0.198



R* = 1.08, Δ* = 0.132



R* = 1.61, Δ* = 0.198

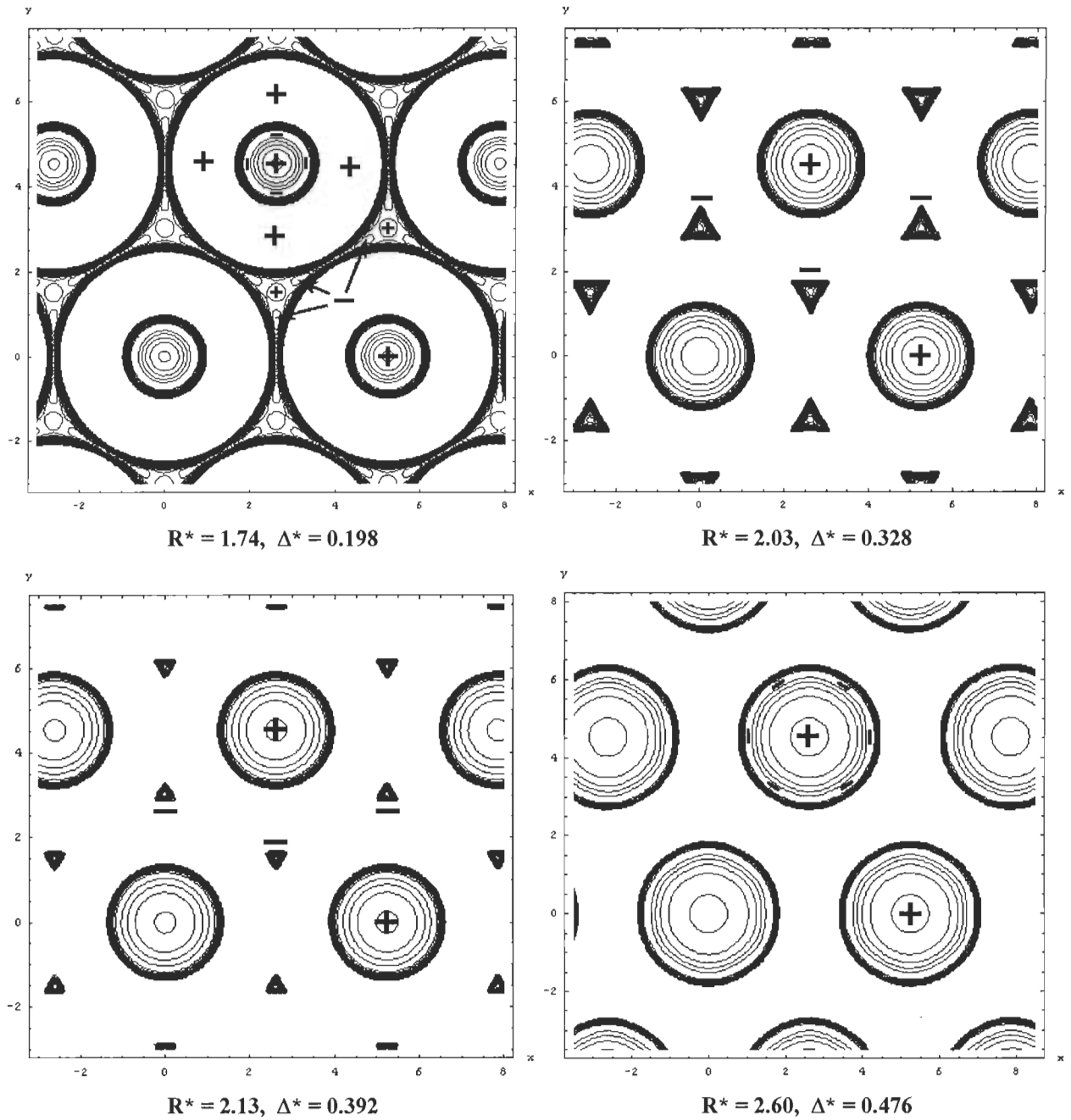


Figure 2.17. Équipotentielles pour un faisceau de 25 SWNTs ayant leurs centres séparés d'une distance réduite de 5.235 pour plusieurs rayons des nanotubes. Les zones de potentiel plus (moins) élevées sont notées avec des signes + (-) et les valeurs des équipotentielles sont comprises entre -10 et 10.

Les deux figures précédentes sont intéressantes car elles permettent de voir où se situent les différents sites d'adsorption (minimums de potentiel) lorsque les dimensions d et R du faisceau varient à tour de rôle. Comme nous le verrons au chapitre 5, la profondeur des puits de potentiel

dans les sites d'adsorption est intimement liée à l'énergie d'activation du processus d'adsorption. C'est pourquoi nous avons calculé la position et la profondeur de ces minimums au moyen d'une méthode générale qui ne requiert aucune hypothèse quant à la position approximative des minimums. Physiquement, la méthode consistera à laisser tomber sur la surface représentant le potentiel des petites billes régulièrement espacées et disposées en une grille carrée. Lorsqu'elles seront lâchées, chaque bille du grillage initial va suivre à chaque instant la direction opposée au gradient de la surface jusqu'à ce qu'elle finisse par se loger dans un minimum local du potentiel. Évidemment, on pourra avoir plus d'une bille par minimum lorsque celles-ci auront atteint l'état de repos. De manière plus détaillée, le calcul des minimums est fait de la manière suivante :

Les points initiaux sont arrangés sur une grille carrée de côté `Lgrille` ayant `Ngrille` points sur chacun de ses côtés. La grille est centrée au point `(Xgrille, Ygrille)` et ses points sont stockés dans une matrice `MatriceGrille` :

```
PtsInitiauxGrille[Xgrille_, Ygrille_, Lgrille_, Ngrille_] :=
 Module[{MatriceGrille, i, j, delta, Xstart, Ystart},
  delta = Lgrille / (Ngrille - 1);
  MatriceGrille = {};
  Xstart = Xgrille - Lgrille / 2;
  Ystart = Ygrille - Lgrille / 2;
  For[i = 1, i ≤ Ngrille, i++,
   (* Pour chaque ligne de la grille, à partir du bas *)
   For[j = 1, j ≤ Ngrille, j++,
     (* Pour chaque élément de la ligne "i" *)
     MatriceGrille = Append[MatriceGrille,
        {Xstart + (j - 1) * delta, Ystart + (i - 1) * delta}];
    ];
  ];
  Return[MatriceGrille];
 ]
```

(2.38)

Une fonction de `Migration[ ]` (donnée ci-après) est utilisée pour calculer la position finale des points de la grille initiale et la valeur correspondante des minimums. Ces résultats sont ensuite retournés dans une matrice `PointsFinaux`. La fonction de `Migration[ ]` fait "migrer" chaque point de la grille un après l'autre au moyen de la fonction `FindMinimum[ ]` implémentée dans Mathematica. La fonction `FindMinimum[V[x, y], {x, x0}, {y, y0}]` utilise un algorithme de minimisation qui consiste essentiellement en une méthode "de la plus forte pente" ("steepest

descent") sophistiquée. La méthode "de la plus forte pente" consiste à suivre la direction opposée au gradient à partir d'un point de départ $\{x_0, y_0\}$ donné jusqu'à ce qu'un minimum soit rencontré dans cette direction. Ce nouvel endroit $\{x_1, y_1\}$ peut être alors utilisé pour débuter une nouvelle itération de descente (après y avoir réévalué le gradient). Éventuellement, ce processus va finir par mener à un minimum local du potentiel `V[x, y]`.

```
Migration[GrilleInitiale_, R_, d_, Ntubes_, θ_, ϵ_, σ_] :=
 Module[{PointsFinaux, LesPtsDuReseau, i},
  LesPtsDuReseau = PtsDuReseau[d, Ntubes];
  PointsFinaux = {};
  For[i = 1, i ≤ Length[GrilleInitiale], i++,
   PointsFinaux = Append[PointsFinaux,
      FindMinimum[Vbundle[x, y, LesPtsDuReseau, R, Ntubes, θ, ϵ, σ],
       {x, GrilleInitiale[[i, 1]]}, {y, GrilleInitiale[[i, 2]]}]];
  ];
  Return[Sort[PointsFinaux]];
 ]
```

(2.39)

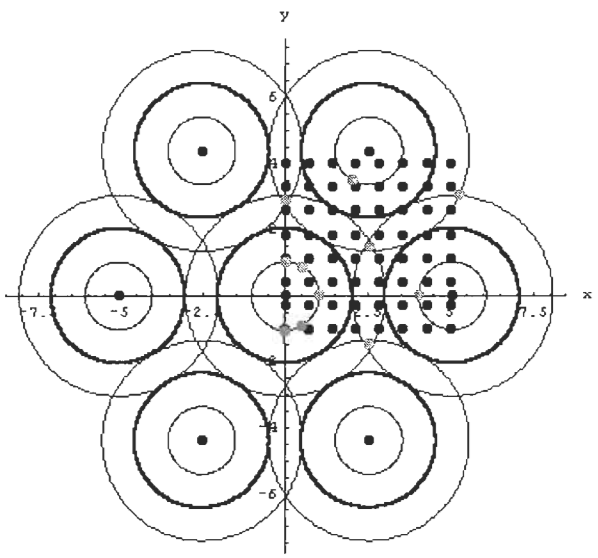
La fonction suivante permet de tracer sur un graphique la grille des points initiaux et la position des minimums trouvés au voisinage de ces points. La fonction trace également les contours (cercles épais) des nanotubes du faisceau avec leurs centres disposés sur un réseau triangulaire de `Ntubes`. Les minimums à l'intérieur et à l'extérieur de chaque nanotube considéré individuellement y sont également tracés en utilisant des cercles minces. Pour finir, la fonction imprime la matrice des `PointsFinaux` sous le graphique. La figure suivante montre le résultat de cette fonction pour un faisceau ayant 7 tubes de 6.5 Å espacés de 3.15 Å pour l'hydrogène (les valeurs réduites y sont utilisées).

Dans cette fonction, la sous-routine `PlotCerclesReseau[R, d, Ntubes, epaisseur]` est utilisée pour tracer tous les cercles représentant les nanotubes et leurs minimums de potentiel. `ListPlot[{{x1, y1}, {x2, y2}, …}]`, `PlotGrillePtsInitiaux[Xgrille, Ygrille, Lgrille, Ngrille, grosseur]` et `PlotPointsMigrers[PointsFinaux, grosseur]` permettent de tracer les centres des nanotubes, les points initiaux et les minimums trouvés, respectivement.

```
CalculerEtTracerMinsVbundle[R_, d_, Ntubes_, Xgrille_, Ygrille_, Lgrille_, Ngrille_, θ_, ϵ_, σ_] :=
 Module[{MatriceMigrer},
  MatriceMigrer = Migration[PtsInitiauxGrille[Xgrille, Ygrille, Lgrille, Ngrille], R, d, Ntubes, θ, ϵ, σ];
  Print[Show[
    ListPlot[PtsDuReseau[d, Ntubes], AxesLabel → {x, y}, Prolog -> AbsolutePointSize[6],
     AspectRatio → Automatic, DisplayFunction → Identity],
    PlotCerclesReseau[R, d, Ntubes, 2],
    PlotCerclesReseau[MinInterieur[R, θ, ϵ, σ], d, Ntubes, 1],
    PlotCerclesReseau[MinExterieur[R, θ, ϵ, σ], d, Ntubes, 1],
    PlotGrillePtsInitiaux[Xgrille, Ygrille, Lgrille, Ngrille, 6],
    PlotPointsMigrers[MatriceMigrer, 6],
    DisplayFunction → $DisplayFunction
   ],
   "\n", MatrixForm[MatriceMigrer]
  ]
 ]
```

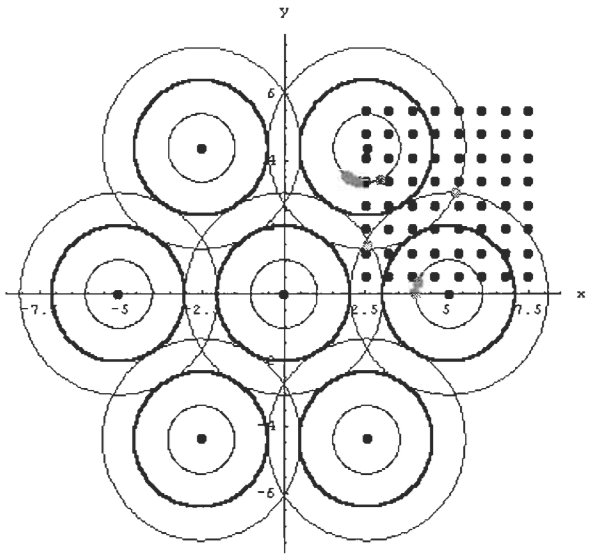
(2.40)

a)



-2.00627	{x→ 5.24424, y→ 3.02776}
effacer	effacer
-2.00627	{x→ 5.24424, y→ 3.02776}
-1.94117	{x→ 0.514139, y→ -0.890509}
effacer	effacer
-1.94117	{x→ 0.517398, y→ -0.888658}
-1.93759	{x→ 0., y→ 1.02785}
effacer	effacer
-1.93759	{x→ 0., y→ 1.02785}
effacer	effacer
-1.92375	{x→ 2.01069, y→ 3.49766}
-1.44518	{x→ 2.53148, y→ 1.46155}
effacer	effacer
-1.44518	{x→ 2.51484×10^-7, y→ 2.9231}

b)



-2.00627	{x→ 5.24424, y→ 3.02776}
effacer	effacer
-2.00627	{x→ 5.24424, y→ 3.02776}
-1.92375	{x→ 2.0172, y→ 3.49391}
effacer	effacer
-1.92372	{x→ 1.99386, y→ 3.50779}
-1.92353	{x→ 4.03547, y→ 0.0649438}
-1.92303	{x→ 1.91319, y→ 3.5638}
-1.92253	{x→ 2.1595, y→ 3.42826}
-1.91971	{x→ 2.27961, y→ 3.3822}
-1.91744	{x→ 4.10686, y→ 0.39178}
-1.90451	{x→ 2.95279, y→ 3.44114}
-1.44518	{x→ 2.53148, y→ 1.46155}
effacer	effacer
-1.44518	{x→ 2.53148, y→ 1.46155}

Figure 2.18. Position des minimums du potentiel d'adsorption (points gris) pour un faisceau de 7 SWNTs ayant d* = 5.063 et R* = 2.038 (16.15 Å et 6.50 Å pour H_2). Les parties a) et b) sont utilisées pour rechercher des minimums interstitiels et périphériques respectivement. Les matrices montrent la valeur et la position de quelques minimums trouvés.

Nous avons ainsi calculé les minimums du potentiel d'adsorption pour d'autres faisceaux formés de 7 nanotubes mais possédant des dimensions d et R différentes. La valeur de ces minimums situés dans les différents sites d'adsorption du faisceau sont montrées dans le tableau suivant. Le fait de prendre des petits faisceaux de seulement 7 nanotubes n'influencera que très peu la valeur des minimums trouvés (pour un faisceau de 25 nanotubes ayant $d^* = 5.235$ et $R^* = 2.038$, les minimums de V^* trouvés sont en bon accord avec les valeurs du tableau suivant: -1.901, -3.012 et -2.008 dans les tubes, les interstices et les sites périphériques respectivement).

Tableau 2.2 Valeur des minimums du potentiel d'adsorption (en kJ/mol) dans les différents sites d'adsorption d'un faisceau de 7 SWNTs pour l'hydrogène. Les valeurs réduites correspondantes (par σ ou ε'_s) sont données entre parenthèses. R_{min} est le rayon du nanotube isolé ayant l'intérieur le plus attractif.

d (Å) (d*)	**Site d'adsorption**	**R = 6.501 Å (R* = 2.038)**	**R_{min} = 3.464 Å (R_{min}* = 1.086)**
	SWNT intérieur	-5.450 (-1.769)	-12.52 (-4.065)
	SWNT extérieur	-3.080 (-1.000)	-2.793 (-0.9067)
16.70 (5.235)	Intérieur SWNTs	-5.843 (-1.897)	
	Sites intersticiels	-9.249 (-3.002)	
	Sites périphériques	-6.180 (-2.006)	
16.15 (5.063)	Intérieur SWNTs	-5.980 (-1.941)	
	Sites intersticiels	-4.452 (-1.445)	
	Sites périphériques	-6.181 (-2.006)	
10.63 (3.332)	Intérieur SWNTs		-13.42 (-4.355)
	Sites intersticiels		3.810 (1.237)
	Sites périphériques		-5.650 (-1.834)
10.08 (3.159)	Intérieur SWNTs		-13.78 (-4.473)
	Sites intersticiels		70.58 (22.91)
	Sites périphériques		-5.657 (-1.836)

Du tableau précédent, on voit que le site d'adsorption le plus attractif est situé à l'intérieur des nanotubes d'un faisceau formé de tubes de rayon R_{min} espacés de 3.15 Å (cf. [24]). Il est intéressant de constater l'avantage que procure le regroupement des SWNTs en faisceau en comparant ce minimum avec ceux des autres nanostructures calculés précédemment :

$$\frac{V_{min,\,bundle}}{V_{min,\,SWNT}} = \frac{4.473\varepsilon_s'}{4.065\varepsilon_s'} = 1.10\,,\ \frac{V_{min,\,bundle}}{V_{min,\,slit}} = \frac{4.473\varepsilon_s'}{2.4\varepsilon_s'} = 1.86 \text{ et } \frac{V_{min,\,bundle}}{V_{min,\,flat}} = \frac{4.473\varepsilon_s'}{1.2\varepsilon_s'} = 3.73 \qquad (2.41)$$

Ainsi, le gain (de 10 %) sur la profondeur du puits de potentiel par rapport au SWNT isolé est assez modeste pour l'hydrogène. Cependant, puisque l'interaction de Van der Waals (totale) entre les nanotubes voisins du faisceau impose un espacement de 3.15 Å entre leurs parois (mesuré à la réf. [24] et déduit à la réf. [19]), il est possible d'obtenir un gain plus élevé pour un gaz possédant des molécules plus grosses : R_{min}* demeure inchangé à 1.086, mais l'espacement réduit $g^* = 3.15/\sigma = d^* - 2R^*$ correspondant est alors diminué, ce qui revient à rapprocher les nanotubes du faisceau entre eux (dans les dimensions réduites par σ) et permet ainsi de creuser davantage le potentiel réduit à l'intérieur des tubes.

2.5. Le potentiel d'adsorption effectif.

Nous allons présenter dans cette section les résultats d'une tentative (suggérée dans [36]) ayant pour but d'inclure une correction quantique au second coefficient du viriel B_{AS} (classique) puisque nous savons que l'hydrogène obéit en fait aux lois de la mécanique quantique. Malgré l'échec de cette tentative, nous aurons quand même une idée de la température à partir de laquelle les effets quantiques deviennent négligeables. Nous obtiendrons dans cette section un "potentiel effectif" qui permettra d'inclure une correction quantique au B_{AS} en remplaçant simplement le potentiel d'adsorption des différentes nanostructures par le potentiel effectif. Étant donné l'ampleur des programmes calculant les B_{AS} (cf. annexes 4 à 7) l'avantage du potentiel effectif est manifeste!

En physique statistique, les fonctions de partition canonique ou grand canonique permettent de déduire toutes les propriétés d'un système thermodynamique. La fonction de partition canonique est donnée par une somme sur tous les états quantiques discrets du système :

$$Q_N(V,T) = Q(N,V,T) = \sum_n e^{-E_n(N,V)/kT} = e^{-\beta F} \qquad (2.42)$$

Où $\beta = 1/kT$ et où F est l'énergie libre de Helmholtz. Lorsque les niveaux d'énergie sont très rapprochés ($\Delta E/kT << 1$, i.e. $T \to \infty$) nous pourrons remplacer cette somme discrète par une

intégrale sur tous les états classiques (p, q) du système dans l'espace des phases. Cette intégrale est plus simple que l'Équ. (2.42) car elle ne requiert pas la connaissance des E_n.

$$Q_N(V,T) = \frac{1}{N!h^{3N}} \int d\vec{r}_1 \cdots d\vec{r}_N d\vec{p}_1 \cdots d\vec{p}_N \; e^{-H_{class}(p,q)/kT} \quad \text{où } T \to \infty \tag{2.43}$$

L'expression précédente est la fonction de partition canonique classique pour N particules indiscernables et sera utilisée pour calculer les B_{AS} classiques (cf. chap. 3). Dans le formalisme de la physique statistique quantique, l'opérateur densité peut être utilisé pour décrire le système qui a une certaine probabilité w_n de se retrouver dans l'état propre normé $|E_n\rangle$ de l'hamiltonien H du système :

$$\begin{aligned} \rho &= \sum_n w_n |E_n\rangle\langle E_n| = \sum_n \left(\frac{1}{Q} e^{-\beta E_n} \right) |E_n\rangle\langle E_n| \\ &= \frac{1}{Q} \sum_n e^{-\beta H} |E_n\rangle\langle E_n| = \frac{e^{-\beta H}}{Q} \end{aligned} \tag{2.44}$$

Où $Q = \sum_n e^{-\beta E_n} = \sum_n \langle E_n | e^{-\beta H} | E_n \rangle = Tr\, e^{-\beta H}$. Dorénavant, nous allons négliger le Q au dénominateur de l'Équ. (2.44) de sorte que ρ soit non normalisé et que nous ayons dans la représentation des coordonnées :

$$Q = e^{-\beta F} = Tr\; \rho = \int dx \, \langle x|\rho|x\rangle = \int dx \; \rho(x,x) \tag{2.45}$$

L'obtention du potentiel effectif pour une particule se déplaçant dans un potentiel V(x) étant montrée en détails dans la référence [38], nous allons nous contenter de donner les grandes lignes de ce calcul ici. Le calcul est basé sur la théorie des intégrales de parcours de Feynman utilisée également en mécanique quantique (e.g. chap. 16, réf. [39]). L'opérateur densité peut être exprimé en terme d'un "temps" $u = \beta\hbar = n\varepsilon$:

$$\rho(u) = e^{-Hu/\hbar} = e^{-H\varepsilon/\hbar} e^{-H\varepsilon/\hbar} \cdots e^{-H\varepsilon/\hbar} = \rho_\varepsilon \rho_\varepsilon \cdots \rho_\varepsilon \quad \text{(n facteurs)} \tag{2.46}$$

Qui devient dans la représentation des coordonnées :

$$\rho(x,x';u) = \int \cdots \int \rho(x,x_{n-1};\varepsilon)\rho(x_{n-1},x_{n-2};\varepsilon) \cdots \rho(x_1,x';\varepsilon)\, dx_1 \cdots dx_{n-1} \tag{2.47}$$

Cette dernière équation se laisse interpréter comme étant une intégrale (dans la limite $n \to \infty$) sur toutes les trajectoires (x', x_1, …, x_{n-1}, x) reliant x' = x(0) et x = x(u) (cf. Fig. 2.19) et peut s'écrire symboliquement :

$$\rho(x,x';U) = \int \Phi[x(u)]\, \mathcal{D}x(u) \tag{2.48}$$

Où :

$$\Phi[x(u)] = \lim_{\substack{\varepsilon \to 0 \\ n\varepsilon = u}} \rho(x, x_{n-1};\varepsilon)\rho(x_{n-1}, x_{n-2};\varepsilon)\cdots\rho(x_1, x';\varepsilon)\left(\frac{m}{2\pi\hbar\varepsilon}\right)^{-n/2} \tag{2.49}$$

$$\mathcal{D}x(u) = \lim_{n\to\infty} dx_1\, dx_2 \cdots dx_{n-1}\left(\frac{m}{2\pi\hbar\varepsilon}\right)^{n/2} \tag{2.50}$$

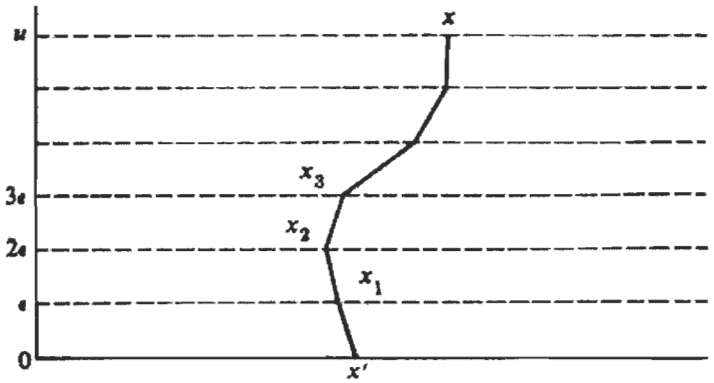


Figure 2.19. Une particule à une dimension se déplaçant de x' à x le long d'un parcours x_1, x_2, etc.

Dans le cas d'une particule soumise à un potentiel V(x), l'Équ. (2.48) devient :

$$\rho(x,x';U) = \int e^{-S[x(u)]}\, \mathcal{D}x(u) = \int \exp\left\{-\frac{1}{\hbar}\int_0^U \left[\frac{m\dot{x}(u)^2}{2} + V(x(u))\right] du\right\} \mathcal{D}x(u) \tag{2.51}$$

Qui permet d'obtenir la fonction de partition (où x' = x est remplacé par x(0)) :

$$e^{-\beta F} = \int dx(0) \int_{\substack{x(0)=x(0) \\ x(U)=x(0)}} e^{-S[x(u)]}\mathcal{D}x(u) \tag{2.52}$$

$$e^{-\beta F} = \frac{\int dx(0) \int_{x(0)=x(U)} \mathcal{D}x\, e^{-(S-S_0)} e^{-S_0}}{\int dx(0) \int_{x(0)=x(U)} \mathcal{D}x\, e^{-S_0}} e^{-\beta F_0} \tag{2.53}$$

Où :

$$e^{-\beta F_0} = \int dx(0) \int_{x(0)=x(U)} \mathcal{D}x\, e^{-S_0} \tag{2.54}$$

$S_0[x(u)]$ est une autre fonctionnelle de la trajectoire qui peut être choisie arbitrairement. Elle est choisie de telle sorte qu'elle soit manipulable plus facilement que la fonctionnelle exacte S. On remarque que le premier facteur de l'Équ. (2.53) est une moyenne ayant e^{-S_0} comme poids :

$$e^{-\beta F} = \left\langle e^{-(S-S_0)} \right\rangle_{S_0} e^{-\beta F_0} \tag{2.55}$$

Utilisant l'inégalité suivante, dont l'interprétation est donnée à la figure ci-dessous :

$$\left\langle e^{-f} \right\rangle \geq e^{-\langle f \rangle} \tag{2.56}$$

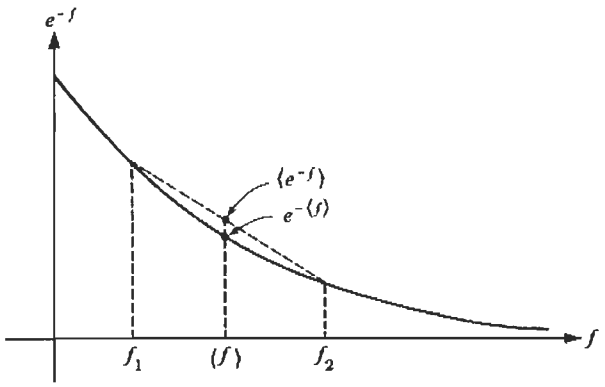


Figure 2.20. Interprétation géométrique de $\langle e^{-f} \rangle \geq e^{-\langle f \rangle}$.

l'Équ. (2.55) permet finalement d'obtenir une inégalité reliant F et les quantités "d'essai" toutes dénotées par l'indice 0 :

$$F \leq F_0 + \frac{1}{\beta} \left\langle S - S_0 \right\rangle_{S_0} \tag{2.57}$$

Cette dernière équation représente en fait un critère de minimisation de la fonctionnelle d'essai S_0 dans le but d'estimer au mieux possible la fonctionnelle exacte S. F_0 et $e^{-\beta F_0}$ seront alors des

approximations pour l'énergie libre et la fonction de partition. Dans le cas d'une particule soumise à un potentiel V(x), la fonctionnelle exacte et la fonctionnelle d'essai auront la forme suivante (où nous avons posé $\hbar = 1$ pour alléger l'écriture des équations qui vont suivre) :

$$S = \int_0^U \left[\frac{m\dot{x}(u)^2}{2} + V\big(x(u)\big) \right] du \tag{2.58}$$

$$S_0 = \int_0^U \frac{m\dot{x}(u)^2}{2} du + Uw(\bar{x}) \tag{2.59}$$

Où $\bar{x}$ est la position moyenne sur la trajectoire x(u) et où $w(\bar{x})$ est un potentiel constant sur la trajectoire considérée et dont la forme fonctionnelle est à déterminer. Nous allons obtenir sa forme en utilisant la condition de minimisation contenue dans l'Équ. (2.57).

$$\bar{x} = \frac{1}{U} \int_0^U x(u)\, du \tag{2.60}$$

Dans le cas où $T \to \infty$, $U \to 0$ de sorte que $\bar{x} \approx x(0)$. De plus, si nous faisons le choix trivial $w(\bar{x}) = V(\bar{x})$, nous obtenons après quelques calculs la limite classique de la fonction de partition :

$$Q_{class} = \sqrt{\frac{mkT}{2\pi\hbar^2}} \int e^{-V(x)/kT} dx = \frac{1}{\Lambda} \int e^{-V(x)/kT} dx \tag{2.61}$$

Où :

$$\Lambda = \frac{h}{\sqrt{2\pi mkT}} \tag{2.62}$$

est la longueur d'onde thermique de la particule. Cette longueur d'onde a le même ordre de grandeur que la longueur d'onde de Broglie d'une particule ayant une énergie cinétique kT ($\lambda = h/p = h/\sqrt{2mK} = h/\sqrt{2mkT}$). Revenant au calcul de $w(\bar{x})$ et de $e^{-\beta F_0}$, nous obtenons après des calculs ardus les résultats suivants :

$$e^{-\beta F_0} = \sqrt{\frac{m}{2\pi\beta\hbar^2}} \int \exp\big[-\beta\, w(y)\big] dy \tag{2.63}$$

$$\frac{1}{\beta} \big\langle S - S_0 \big\rangle_{S_0} = \frac{\int \big(K(y) - w(y)\big) e^{-\beta w(y)} dy}{\int e^{-\beta w(y)} dy} \tag{2.64}$$

Où :

$$K(y)=\sqrt{\frac{6mkT}{\pi\hbar^2}}\int dz\,V(z)\exp\left[-\frac{6mkT}{\hbar^2}(y-z)^2\right] \tag{2.65}$$

Maintenant que nous sommes rendus à ce point-ci, il ne nous reste plus qu'à appliquer le principe variationnel donné à l'Équ. (2.57) afin de trouver la forme du potentiel w(y). Effectuant une petite variation sur w(y), nous obtenons :

$$w(y)\rightarrow w(y)+\eta(y) \tag{2.66}$$

$$e^{-\beta\delta F_0}=\frac{\sqrt{\frac{m}{2\pi\beta\hbar^2}}\int\exp[-\beta(w+\eta)]\,dy}{\sqrt{\frac{m}{2\pi\beta\hbar^2}}\int\exp[-\beta w]\,dy}\approx\frac{\int(1-\beta\eta)\exp[-\beta w]\,dy}{\int\exp[-\beta w]\,dy}$$

$$\approx\exp\left[-\frac{\int\beta\eta\exp[-\beta w]\,dy}{\int\exp[-\beta w]\,dy}\right]$$

D'où :

$$\delta F_0=\frac{\int\eta e^{-\beta w}dy}{\int e^{-\beta w}dy} \tag{2.67}$$

Aussi puisque $e^{-\beta\eta}\approx 1-\beta\eta$ et $1/(A+\delta)\approx 1/A-\delta/A^2$ $(\delta\rightarrow 0)$:

$$\delta\frac{1}{\beta}\langle S-S_0\rangle_{S_0}\approx\frac{\int(K-w-\eta)(1-\beta\eta)e^{-\beta w}\,dy}{\int e^{-\beta w}(1-\beta\eta)dy}-\frac{\int(K-w)e^{-\beta w}\,dy}{\int e^{-\beta w}\,dy}$$

$$\approx\frac{\int e^{-\beta w}\left[(K-w-\eta)(1-\beta\eta)-(K-w)\right]dy}{\int e^{-\beta w}\,dy}$$

$$+\frac{\int\beta\eta\,e^{-\beta w}\,dy}{\left(\int e^{-\beta w}\,dy\right)^2}\int e^{-\beta w}(K-w-\eta)(1-\beta\eta)dy$$

Les termes d'ordre η^2 pourront être négligés dans le premier terme de l'équation précédente. Puisque le deuxième terme est petit devant le premier, tous les termes d'ordre η qu'il contient pourront être négligés, ce qui donne :

$$\delta \frac{1}{\beta}\langle S - S_0 \rangle_{S_0} \approx \frac{\int e^{-\beta w}\left[-\beta\eta(K-w)-\eta\right]dy}{\int e^{-\beta w}\,dy} + \frac{\int e^{-\beta w}(K-w)dy \int \beta\eta\, e^{-\beta w}dy}{\left(\int e^{-\beta w}\,dy\right)^2} \tag{2.68}$$

De sorte que :

$$0 = \delta\left(F_0 + \frac{1}{\beta}\langle S - S_0 \rangle_{S_0}\right) = \frac{\int e^{-\beta w}\left[-\beta\eta(K-w)\right]dy}{\int e^{-\beta w}\,dy} + \frac{\int e^{-\beta w}(K-w)dy \int \beta\eta\, e^{-\beta w}dy}{\left(\int e^{-\beta w}\,dy\right)^2}$$

$$\iint dy\,dy'\, e^{-\beta(w(y')+w(y))}\left[K(y)-w(y)\right]\left[\eta(y')-\eta(y)\right] = 0$$

Qui permet finalement d'obtenir la forme finale du potentiel w(y) :

$$w(y) = K(y) \tag{2.69}$$

La fonction de partition semi-classique est alors donnée par (à comparer avec l'expression classique Équ. (2.61) :

$$Q_{scl} = \sqrt{\frac{mkT}{2\pi\hbar^2}} \int e^{-K(y)/kT}dy = \frac{1}{\Lambda}\int e^{-K(y)/kT}dy \tag{2.70}$$

Donc, tout ce que l'on devra faire sera de remplacer V(y) par K(y) (le potentiel effectif) pour obtenir des expressions semi-classiques et ainsi inclure une correction pour les effets quantiques.

On remarque que l'Équ. (2.65) est en fait une moyenne du potentiel classique V(z) pondérée par une gaussienne d'écart type σ_T centrée sur la position y :

$$\sigma_T = \hbar/\sqrt{12mkT} = \Lambda/\sqrt{24\pi} \tag{2.71}$$

Donc, lorsque $T \to \infty$, $\sigma_T \to 0$ et $K(y) \to V(y)$ comme on s'y attend lorsque les effets quantiques deviennent négligeables. De façon plus précise, les effets quantiques deviendront négligeables lorsque le potentiel est pratiquement constant sur un rayon de $3\sigma_T$ autour de y. Puisque le potentiel d'adsorption varie très rapidement près des parois, c'est là que les effets quantiques seront le plus significatif. De plus, puisque la largeur du puits de potentiel d'adsorption a une largeur typique de σ (cf. Fig. 2.4 pour d* = 4 et $|z^*| > d^*/2$), la profondeur du puits de potentiel effectif coïncidera essentiellement avec celle du potentiel classique lorsque $2\cdot 3\sigma_T << \sigma$, i.e. lorsque T >> 7.10 K pour l'adsorption d'hydrogène sur une feuille de graphène.

Afin d'avoir une meilleure idée de la température à partir de laquelle les effets quantiques deviennent négligeables et dans le but de préparer le terrain pour le calcul du potentiel effectif d'un SWNT et d'un faisceau de SWNTs, nous avons calculé numériquement le potentiel effectif d'une feuille de graphène. Pour ce cas, le potentiel effectif réduit (sans dimensions et indépendant de la nature de l'adsorbat) évalué à une distance d* du plan est donné par :

$$V^*_{eff,\,flat}(d^*,\alpha^*) = \sqrt{\frac{\alpha^*}{\pi}} \int_{-\infty}^{\infty} dx^*\, V^*_{flat}(x^*)\, e^{-\alpha^*\left(x^*-d^*\right)^2} \tag{2.72}$$

Où :

$$V^*_{flat}(x^*) = 2\left[\frac{2}{5}\left(\frac{1}{x^*}\right)^{10} - \left(\frac{1}{x^*}\right)^4\right] \tag{2.73}$$

$$\alpha^* = \frac{1}{2\sigma_T^{*2}} = \frac{6mT^*}{\hbar^2}\varepsilon_s{}'\sigma^2 = \frac{12\pi}{\Lambda^{*2}} \quad \left(= 2.5357\ K^{-1}\cdot T \text{ pour } H_2\right) \tag{2.74}$$

$$T^* = kT/\varepsilon_s{}' \tag{2.75}$$

Cependant l'Équ. (2.72) donnant le potentiel effectif d'une feuille de graphène possède un problème majeur. Bien que le terme gaussien ait pour effet d'atténuer le potentiel $V^*_{flat}(x^*)$ loin du point d'évaluation d*, cela n'est pas suffisant pour empêcher la divergence de V^*_{flat} à x* = 0 de faire surface, de sorte que l'Équ. (2.72) diverge pour tout d*. En effet, posant $x^* = u^{-1}$:

$$V^*_{eff,\,flat}(d^*,\alpha^*) = 2\sqrt{\frac{\alpha^*}{\pi}}\left[\frac{2}{5}I_{10} - I_4\right] = 2\sqrt{\frac{\alpha^*}{\pi}}\int_{-\infty}^{\infty} du\left[\frac{2}{5}u^8 - u^2\right]e^{-\alpha^*\left(u^{-1}-d^*\right)^2} \tag{2.76}$$

Où :

$$I_n = \int_{-\infty}^{\infty} x^{*-n} e^{-\alpha^*\left(x^*-d^*\right)^2} dx^* = \int_{-\infty}^{\infty} u^{n-2} e^{-\alpha^*\left(u^{-1}-d^*\right)^2} du \tag{2.77}$$

Il est aisé de voir que l'intégrand de l'Équ. (2.76) tend vers l'infini lorsque $|u| \to \infty$ mais qu'il demeure borné au centre de sorte que le potentiel effectif divergera peu importe l'endroit d* choisi pour l'évaluer. Ce problème subsistera également pour le potentiel effectif des autres nanostructures (de forme similaire à l'Équ. (2.65)) puisque le terme gaussien n'arrive pas à atténuer la divergence près des parois. Afin d'avoir tout de même une idée du comportement du

potentiel effectif en fonction de la température, nous avons utilisé un potentiel tronqué près de l'origine pour éviter la divergence du potentiel $V^*_{flat}(x^*)$ près du plan :

$$V^*_{\substack{flat,\\ tronq}}(x^*,\Delta^*) = \begin{cases} 2\left[\frac{2}{5}x^{*-10} - x^{*-4}\right], & |x^*| \geq \Delta^* \\ 2\left[\frac{2}{5}\Delta^{*-10} - \Delta^{*-4}\right], & |x^*| < \Delta^* \end{cases} \tag{2.78}$$

Posant u = x* – d* et limitant l'intégration dans l'Équ. (2.72) à $N_s\sigma^*_T$ autour du point d'évaluation d* (puisque l'intégrand est négligeable en dehors de cette plage) nous avons après avoir remplacé le facteur de normalisation $\sqrt{\alpha^*/\pi}$ par le facteur correspondant pour cette plage de valeurs de x* :

$$V^*_{\substack{eff,\,flat\\ tronq}}(d^*,\alpha^*;\Delta^*,N_s) \approx \frac{\int_{N_s/\sqrt{2\alpha^*}}^{-N_s/\sqrt{2\alpha^*}} V^*_{\substack{flat,\\ tronq}}(u+d^*,\Delta^*)\,e^{-\alpha^* u^2}\,du}{\int_{N_s/\sqrt{2\alpha^*}}^{-N_s/\sqrt{2\alpha^*}} e^{-\alpha^* u^2}\,du} \tag{2.79}$$

Choisissant une distance de $\Delta^* = 0.3\,\sigma$ de la surface pour tronquer le potentiel (car à cette distance, nous sommes bien entrés dans les coeurs durs) et un $N_s = 4.0$ suffisamment grand pour inclure pratiquement toute la gaussienne, nous obtenons finalement le comportement du potentiel effectif réduit pour plusieurs températures (données entre parenthèses pour l'hydrogène) :

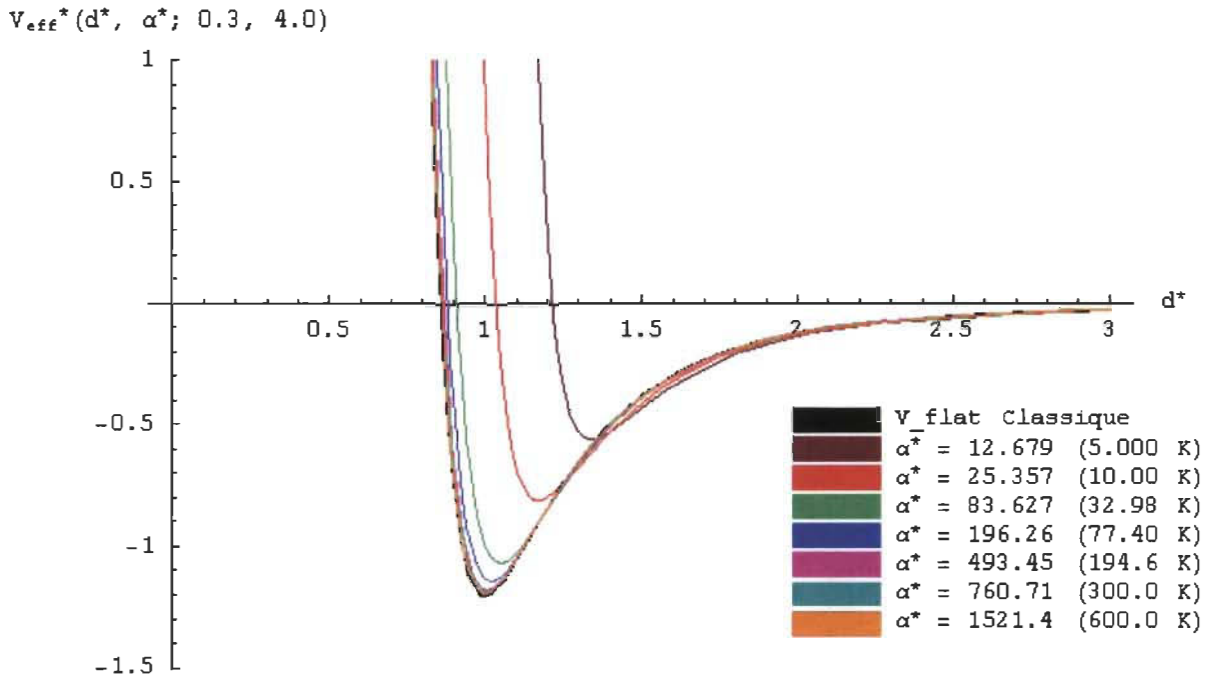


Figure 2.21. Potentiel effectif réduit d'adsorption sur une feuille de graphène pour plusieurs températures (les valeurs entre parenthèses sont pour l'hydrogène).

Sur ce graphique, nous pouvons voir finalement que les effets quantiques seront négligeables pour les températures d'intérêt ($T \geq 77.4$ K) dans le cas de l'hydrogène. Pour d'autres gaz plus massifs, les effets quantiques seront encore plus négligeables car leur σ_T seront plus petits.

Le problème qu'il y a lorsqu'on utilise un potentiel $V^*_{\substack{flat, \\ tronq}}(x^*, \Delta^*)$ tronqué pour calculer le potentiel effectif est que le potentiel résultant est mal défini près de la surface et que cela empire à mesure que la température diminue. (Sous ces conditions, l'intégrand de l'Équ. (2.79) possède un pic très étroit près de $u = -d^*$, ce qui cause des problèmes d'intégration numérique). Nous ne pouvons donc pas faire une sous-routine fonctionnant à toutes les températures et qui pourrait s'insérer facilement dans le programme calculant le second coefficient du viriel B_{AS} d'une feuille de graphène (cf. annexe 4). Néanmoins, nous allons maintenant généraliser l'Équ. (2.65) du potentiel effectif d'une particule à une dimension au cas du faisceau de SWNTs et du SWNT isolé. Prenant trois degrés de liberté indépendants x, y et z et puisque $V(\vec{r})$ est indépendant de z à cause de la symétrie axiale, nous obtenons :

$$V(\vec{r}) = V(\vec{r}_{xy}) = V(x) + V(y)$$
$$V_{eff}(\vec{r}) = V_{eff}(\vec{r}_{xy}) = V_{eff}(x) + V_{eff}(y) \tag{2.80}$$

Avec (où C_T et D_T sont des constantes de normalisation) :

$$V_{eff}(x) = C_T \int dx'\, V(x')\, e^{-\alpha(x'-x)^2} \quad ; \quad \alpha = \frac{6mkT}{\hbar^2} \tag{2.81}$$

$$V_{eff}(\vec{r}_{xy}) = D_T \int dx'\, V(x')\, e^{-\alpha(x'-x)^2} \int dy'\, e^{-\alpha(y'-y)^2} + D_T \int dy'\, V(y')\, e^{-\alpha(y'-y)^2} \int dx'\, e^{-\alpha(x'-x)^2} \tag{2.82}$$

Dans le cas d'un SWNT et d'un faisceau de SWNTs, l'intégration sera faite sur la projection dans le plan XY de la région de l'espace accessible aux molécules d'adsorbat A_{acc}, de sorte que l'équation précédente puisse s'écrire sous la forme :

$$V_{eff}(\vec{r}_{xy}) = \frac{\int_{A_{acc}} dA'\, V(\vec{r}_{xy}') \exp\left[-\dfrac{(\vec{r}_{xy}' - \vec{r}_{xy})^2}{\sigma_T^{\,2}}\right]}{\int_{A_{acc}} dA' \exp\left[-\dfrac{(\vec{r}_{xy}' - \vec{r}_{xy})^2}{\sigma_T^{\,2}}\right]} \tag{2.83}$$

Où :

$$\sigma_T = \left[\frac{\int_{plan} dA\, r^2\, e^{-\alpha r^2}}{\int_{plan} dA\, e^{-\alpha r^2}}\right]^{1/2} = 1/\sqrt{\alpha} = \frac{\hbar}{\sqrt{6mkT}} = \frac{\Lambda}{\sqrt{12\pi}} \tag{2.84}$$

L'écart type ainsi calculé est différent de celui obtenu pour une particule à une seule dimension (Équ. (2.71)). Un rayon de 2.15 σ_T autour du centre de la gaussienne sera suffisant pour inclure 99 % de sa surface.

L'Équ. (2.83) servant au calcul du potentiel effectif dans un plan XY peut être simplifiée dans le cas d'un SWNT ouvert (cf. Fig. 2.22) :

$$V_{eff}(\vec{r}_{xy}) = V_{eff}(r) = \left(\sqrt{\frac{\alpha}{\pi}}\right)^2 \int_{plan} (r'\, d\phi\, dr')\, V(r', \phi) \exp\left[-\frac{(\vec{r}_{xy}' - \vec{r}_{xy})^2}{\sigma_T^{\,2}}\right] \tag{2.85}$$

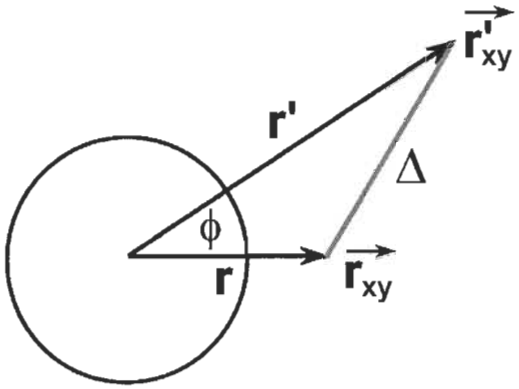


Figure 2.22. Calcul du potentiel effectif d'un SWNT.

$$V_{eff}(r) = \frac{\alpha}{\pi} \int_0^{2\pi} d\phi \int_0^{\infty} dr'\, r'\, V(r') \exp\left[-\frac{r^2 + r'^2 - 2rr'\cos\phi}{{\sigma_T}^2} \right] \tag{2.86}$$

Renommant r' par ρ, posant $\tilde{x} = -\dfrac{2i\,\rho r}{\sigma_T^2}$ et utilisant la représentation intégrale de la fonction de Bessel (cf. réf. [40]) $J_0(\tilde{x}) = \dfrac{1}{2\pi} \int_0^{2\pi} e^{i\tilde{x}\cos\phi}\, d\phi$:

$$V_{eff}(r) = \frac{2}{{\sigma_T}^2} \int_0^{\infty} d\rho\; \rho\; J_0(\tilde{x})\, V(\rho) \exp\left[-\frac{\rho^2 + r^2}{{\sigma_T}^2} \right] \tag{2.87}$$

Utilisant la représentation en série de la fonction $J_0(\tilde{x})$ [40] :

$$J_0(\tilde{x}) = \sum_{s=0}^{\infty} \frac{(-1)^s}{s!(0+s)!} \left(\frac{\tilde{x}}{2} \right)^{0+2s} = J_0(-\tilde{x}) \tag{2.88}$$

et définissant une fonction de Bessel modifiée :

$$I_0(\tilde{x}) = i^{-0} J_0(i\tilde{x}) = J_0(i\tilde{x}) \approx 1 + \frac{\tilde{x}^2}{4} + \frac{\tilde{x}^4}{64} + \frac{\tilde{x}^6}{2304} + \ldots \tag{2.89}$$

le potentiel d'un SWNT ouvert aux extrémités devient finalement :

$$V_{eff}(r) = \frac{2}{{\sigma_T}^2} \int_0^{\infty} d\rho\; \rho\; I_0\left(\frac{2\rho r}{{\sigma_T}^2} \right) V(\rho) \exp\left[-\frac{\rho^2 + r^2}{{\sigma_T}^2} \right] \tag{2.90}$$

Dont l'allure est donnée à la figure ci-dessous [36] pour un nanotube (13, 0) (continu) ayant un rayon de 5.09 Å. Cette figure montre également la corrugation du potentiel du nanotube (13, 0)

discret en traçant le potentiel le long d'une ligne radiale passant par le centre d'un hexagone (points S) ou passant par un atome de carbone (points A) du réseau d'hexagones formant sa surface.

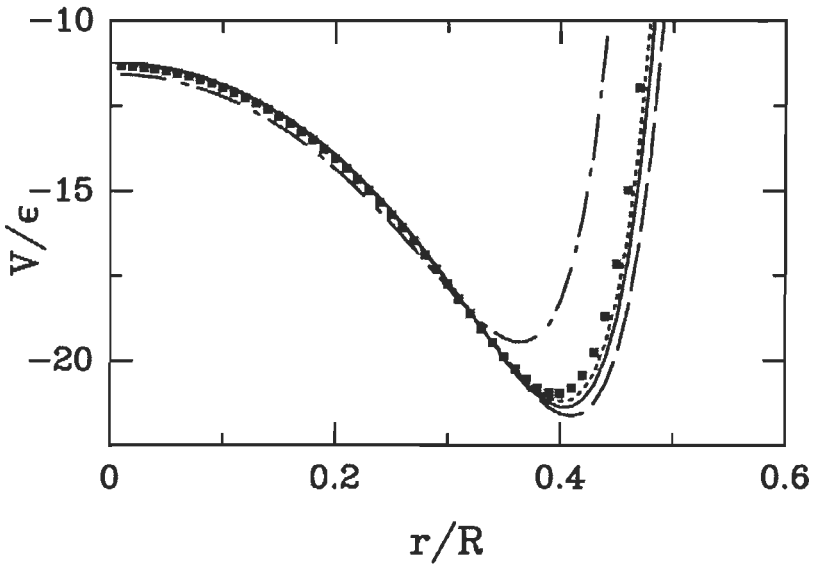


Figure 2.23. Potentiel d'adsorption (réduit par ε_{C-H_2}) d'une molécule d'hydrogène sur un SWNT (13, 0) discret le long d'une ligne radiale passant par les points S (courbe tiretée) ou par les points A (courbe pointillée). La courbe continue est le potentiel classique V(r, R) obtenu avec un nanotube continu. Le potentiel d'adsorption effectif à 10 K et à 50 K est représenté par la courbe tiret-point-tiret et par les carrés respectivement.

De la figure précédente, nous voyons que les effets quantiques sont assez faibles pour l'adsorption d'hydrogène à 50 K sur un nanotube de 5.09 Å de rayon. Donc, comme pour le cas de l'adsorption sur une feuille de graphène, les effets quantiques seront négligeables aux températures d'intérêt expérimental (T ≥ 77.4 K).

CHAPITRE 3. LE SECOND COEFFICIENT DU VIRIEL B_{AS}.

3.1. Équations générales des coefficients du viriel pour l'adsorption.

Nous allons maintenant développer la quantité de molécules de gaz adsorbée en excès sur la surface de l'adsorbant en série du viriel. Les molécules de gaz seront considérées comme des particules (sans degrés de liberté internes) indiscernables. Nous présenterons ici l'essentiel de la dérivation menant à l'obtention des second et troisième coefficients du viriel (B_{AS} et C_{AAS}), les détails étant présentés dans de nombreuses références [11-13, 41].

Les fonctions de partition canonique classique du gaz en présence et en absence (dont les quantités seront indicées d'un 0) de la surface sont données par (après avoir intégré l'Équ. (2.43) sur les $\vec{p}_i$) :

$$Q_N^0(V,T) = \frac{Z_N^0}{N!\Lambda^{3N}} \quad ; \quad Q_N(V,T) = \frac{Z_N}{N!\Lambda^{3N}} \tag{3.1}$$

où les intégrales de configurations classiques sont données par :

$$Z_N^0 = \int_{V_{accessible}} e^{-U_N^0/kT} d\vec{r}_1 \cdots d\vec{r}_N \quad ; \quad Z_N = \int_{V_{accessible}} e^{-U_N/kT} d\vec{r}_1 \cdots d\vec{r}_N \tag{3.2}$$

et où U_N et U_N^0 sont l'énergie potentielle totale du gaz en présence et en absence de la surface. En supposant que le potentiel intermoléculaire u(r) est additif par paires, nous obtenons :

$$U_N^0 = \sum_{i<j} u(r_{ij}) \quad ; \quad U_N = U_N^0 + \sum_{i=1}^{N} u_s(\vec{r}_i) \tag{3.3}$$

où $u_s(\vec{r})$ est le potentiel d'interaction d'une molécule d'adsorbat avec la surface de l'adsorbant qui a été étudié au chapitre précédent pour différentes nanostructures.

Les fonctions de partition grand canonique classiques deviennent donc, en définissant l'activité $z = Q_1^0 \lambda / V = \lambda / \Lambda^3$ (où $\lambda = e^{\beta\mu}$, μ = potentiel chimique de l'adsorbat) :

$$\Xi^0 = \sum_{N\geq 0} Q_N^0 \lambda^N = \sum_{N\geq 0} \frac{Z_N^0}{N!} z^N \quad ; \quad \Xi = \sum_{N\geq 0} Q_N \lambda^N = \sum_{N\geq 0} \frac{Z_N}{N!} z^N \tag{3.4}$$

Qui donne, en développant formellement le logarithme au moyen de la série $\ln(1+x) = x - x^2/2 + x^3/3 - \ldots$:

$$\ln \Xi^0 = \sum_{j\geq 1} V b_j^0 z^j \quad ; \quad \ln \Xi = \sum_{j\geq 1} V b_j z^j \tag{3.5}$$

Où les coefficients b_j sont donnés par :

$$\begin{aligned} 1!Vb_1 &= Z_1; \qquad b_1^0 = 1 \\ 2!Vb_2 &= Z_2 - Z_1^{\,2} \\ 3!Vb_3 &= Z_3 - 3Z_1Z_2 + 2Z_1^{\,3} \end{aligned} \tag{3.6}$$

avec des équations similaires pour les b_j^0. Les nombres moyens de particules en absence et en présence de la surface s'obtiennent des fonctions de partition grand canonique :

$$\begin{aligned} \overline{N} &= z\left(\frac{\partial \ln \Xi}{\partial z}\right)_{V,T} = \sum_{j\geq 1} V j b_j z^j \\ \overline{N}^0 &= z\left(\frac{\partial \ln \Xi^0}{\partial z}\right)_{V,T} = \sum_{j\geq 1} V j b_j^0 z^j \end{aligned} \tag{3.7}$$

De sorte que la quantité de molécules adsorbées en excès sera simplement donnée par la différence du nombre de molécules en présence et en absence de l'adsorbant :

$$N_a = \overline{N} - \overline{N}^0 = V\sum_{j\geq 1} j(b_j - b_j^0) z^j = (Z_1 - V)z + \left(Z_2 - Z_1^{\,2} - Z_2^0 + Z_1^{0\,2}\right)z^2 + \ldots \tag{3.8}$$

Loin de la surface, nous pouvons supposer que le gaz est idéal ($U_N^0 = 0$), de sorte que $Z_2^0 = V^2$. Substituant les Z_N dans l'équation précédente et puisque $z \to p/kT$ quand $p \to 0$, nous obtenons finalement la série du viriel pour l'adsorption en excès :

$$N_a = \overline{N} - \overline{N}^0 = B_{AS}\left(\frac{p}{kT}\right) + C_{AAS}\left(\frac{p}{kT}\right)^2 + D_{AAAS}\left(\frac{p}{kT}\right)^3 + \ldots \tag{3.9}$$

Où les second et troisième coefficient du viriel sont donnés par :

$$B_{AS} = \int_V \left[e^{-\beta u_s(\vec{r}_1)} - 1\right] d\vec{r}_1 \tag{3.10}$$

$$C_{AAS} = \int_V e^{-\beta(u_s(\vec{r}_1)+u_s(\vec{r}_2))} f_{12}\, d\vec{r}_1\, d\vec{r}_2 \quad ; \quad f_{12} = e^{-\beta u(r_{12})} - 1 \tag{3.11}$$

Dans les équations précédentes pour B_{AS} et C_{AAS}, V est le volume accessible aux molécules d'adsorbat. On remarque que le coefficient B_{AS} fait intervenir l'interaction d'une molécule d'adsorbat en présence de la surface et ignore toutes les autres molécules présentes dans le gaz. Donc l'approximation de la loi de Henry (qui consiste à garder simplement le terme linéaire en pression) valide aux très faibles recouvrements néglige l'interaction adsorbat-adsorbat. Pour des pressions plus élevées, nous devrons tenir compte de cette interaction en incluant C_{AAS}. Et pour des recouvrements encore plus élevés, l'interaction à 3 corps devra aussi être incluse au moyen du coefficient D_{AAAS}. Au cours de ce travail, nous nous limiterons à l'étude du coefficient B_{AS}.

3.2. Le second coefficient du viriel B_{AS} pour les différentes nanostructures.

Lors de cette section, nous allons donner la forme que prend l'équation (3.10) dans le cas des différentes nanostructures vues au chapitre précédent. Nous obtiendrons des équations normalisées, écrites en terme du potentiel d'adsorption réduit. Ces coefficients B^*_{AS} réduits auront l'avantage d'avoir un comportement universel, indépendant de la nature de l'adsorbant et de l'adsorbat.

3.2.1. Le second coefficient du viriel d'une feuille de graphène.

Le B_{AS} non normalisé, lorsque l'adsorption a lieu des deux côtés du plan d'aire A, est donné par :

$$B_{AS}(T) = 2A\int_0^\infty \left\{ \exp\left(-\frac{V_{flat}(z)}{kT} \right) - 1 \right\} dz \tag{3.12}$$

Où :

$$V_{flat}(z) = 2\varepsilon_s' \left[\frac{2}{5}\left(\frac{\sigma}{z}\right)^{10} - \left(\frac{\sigma}{z}\right)^{4} \right] \quad ; \quad \varepsilon_s' = \pi\theta\varepsilon\sigma^2 \quad (\varepsilon_s'/\mathrm{k} = 371\,\mathrm{K\ pour\ H_2}) \tag{3.13}$$

Le B_{AS} normalisé, indépendant de la nature de l'adsorbant et de l'adsorbat (i.e. indépendant de ε, σ, θ), est donné par :

$$B^*_{AS}(T^*) = \frac{B_{AS}}{A\sigma} = 2\int_0^{\infty}\left\{\exp\left(-\frac{V^*_{flat}(z^*)}{T^*}\right)-1\right\}dz^* \tag{3.14}$$

Où :

$$V^*_{flat}(z^*) = 2\left[\frac{2}{5}\left(\frac{1}{z^*}\right)^{10} - \left(\frac{1}{z^*}\right)^{4}\right] \quad \text{(le potentiel d'adsorption réduit)} \tag{3.15}$$

$$T^* = kT/\varepsilon_s' \quad \text{(la température réduite)}$$
$$z^* = z/\sigma \tag{3.16}$$

On remarque des équations précédentes, que les B^*_{AS} et B_{AS} coïncideront lorsque l'on aura $A = 1, \sigma = 1$ et $\varepsilon = k/\pi\theta$ (= 0.8377 lorsque les énergies sont exprimées en Kelvins) :

$$B_{AS} \rightarrow B^*_{AS} = B_{AS}/A\sigma$$
$$T \rightarrow T^* = kT/\varepsilon_s' \tag{3.17}$$

Alors, il suffira d'entrer ces valeurs dans un programme calculant le B_{AS} (donné à l'annexe 4) pour qu'il puisse aussi calculer le B^*_{AS} réduit sans avoir à être modifié.

3.2.2. Le second coefficient du viriel d'un pore en fente.

Le B_{AS} non normalisé, lorsque l'adsorption a lieu entre deux feuilles de graphène d'aire A séparées d'une distance d, est donné par :

$$B_{AS}(T, d) = 2A\int_0^{d/2}\left\{\exp\left(-\frac{V_{slit}(z, d)}{kT}\right)-1\right\}dz \tag{3.18}$$

Où :

$$V_{slit}(z, d) = V_{flat}(z + d/2) + V_{flat}(z - d/2) \tag{3.19}$$

Le B_{AS} normalisé, indépendant de la nature de l'adsorbant et de l'adsorbat (i.e. indépendant de ε, σ, θ), est donné par :

$$B^*_{AS}(T^*, d^*) = \frac{B_{AS}}{A\sigma} = 2\int_0^{d^*/2}\left\{\exp\left(-\frac{V^*_{slit}(z^*, d^*)}{T^*}\right)-1\right\}dz^* \tag{3.20}$$

Où :

$$V^*_{slit}(z^*, d^*) = V^*_{flat}(z^* + d^*/2) + V^*_{flat}(z^* - d^*/2) \tag{3.21}$$

$$d^* = d/\sigma \qquad \text{(la largeur réduite du pore)} \tag{3.22}$$

On remarque des équations précédentes, que les B^*_{AS} et B_{AS} coïncideront lorsque l'on aura $A = 1$, $\sigma = 1$ et $\varepsilon = k/\pi\theta$ (= 0.8377 lorsque les énergies sont exprimées en Kelvins) :

$$\begin{aligned} B_{AS} &\rightarrow B^*_{AS} = B_{AS}/A\sigma \\ T &\rightarrow T^* = kT/\varepsilon_s' \\ d &\rightarrow d^* = d/\sigma \end{aligned} \tag{3.23}$$

Alors, il suffira d'entrer ces valeurs dans un programme calculant le B_{AS} (donné à l'annexe 5) pour qu'il puisse aussi calculer le B^*_{AS} réduit sans avoir à être modifié.

3.2.3. Le second coefficient du viriel d'un nanotube (SWNT et MWNT).

Le B_{AS} non normalisé, lorsque l'adsorption a lieu sur un nanotube de longueur L à parois multiples (MWNT) ayant des rayons R_i, est donné par :

$$B_{AS}(T, \{R_i\}) = 2\pi L \int_{r_{min}}^{\infty} \left\{ \exp\left(-\frac{1}{kT} \sum_{i=1}^{Nbre\ parois} V(r, R_i) \right) - 1 \right\} r\, dr \tag{3.24}$$

Où :

$$V(r, R_i) = 3\varepsilon_s' \left[\frac{21}{32} \left(\frac{\sigma}{R_i} \right)^{10} M_{11}(r/R_i) - \left(\frac{\sigma}{R_i} \right)^{4} M_5(r/R_i) \right] \; ; \; M_n(x) = \int_0^{\pi} \frac{d\phi}{\left(1 + x^2 - 2x\cos\phi\right)^{n/2}} \tag{3.25}$$

$$r_{\min} = \begin{cases} 0 & \text{pour un MWNT ouvert} \\ \text{Max}\{R_i\} & \text{pour un MWNT fermé} \end{cases} \tag{3.26}$$

Dans l'équation (3.26), il est sous-entendu que l'adsorbat ne peut pénétrer entre les parois d'un MWNT fermé aux bouts car les parois sont supposées continues. Le B_{AS} normalisé, indépendant de la nature de l'adsorbant et de l'adsorbat (i.e. indépendant de ε, σ, θ), est donné par :

$$B^*_{AS}\left(T^*, \{R^*_i\}\right) = \frac{B_{AS}}{L\sigma^2} = 2\pi \int_{r^*_{min}}^{\infty} \left\{ \exp\left(-\frac{1}{T^*} \sum_{i=1}^{Nbre\ parois} V^*(r^*, R^*_i) \right) - 1 \right\} r^*\, dr^* \tag{3.27}$$

Où :

$$V^*(r^*, R^*_i) = 3\left[\frac{21}{32}\left(\frac{1}{R^*_i}\right)^{10} M_{11}(r^*/R^*_i) - \left(\frac{1}{R^*_i}\right)^{4} M_5(r^*/R^*_i) \right] \quad \text{(le potentiel réduit)} \tag{3.28}$$

$$\begin{aligned} R^*_i &= R_i/\sigma \qquad \text{(les rayons réduits du MWNT)} \\ r^*_{\min} &= r_{\min}/\sigma \\ r^* &= r/\sigma \end{aligned} \tag{3.29}$$

On remarque des équations précédentes, que les B^*_{AS} et B_{AS} coïncideront lorsque l'on aura $L = 1, \sigma = 1$ et $\varepsilon = k/\pi\theta$ (= 0.8377 lorsque les énergies sont exprimées en Kelvins) :

$$\begin{aligned} B_{AS} &\rightarrow B^*_{AS} = B_{AS}/L\sigma^2 \\ T &\rightarrow T^* = kT/\varepsilon_s{}' \\ R_i &\rightarrow R^*_i = R_i/\sigma \end{aligned} \tag{3.30}$$

Alors, il suffira d'entrer ces valeurs dans un programme calculant le B_{AS} (donné à l'annexe 6) pour qu'il puisse aussi calculer le B^*_{AS} réduit sans avoir à être modifié.

3.2.4. Le second coefficient du viriel d'un faisceau de nanotubes à parois simples.

Le B_{AS} non normalisé, lorsque l'adsorption a lieu sur un faisceau de SWNTs de longueur L et de rayon R, ayant leurs centres séparés d'une distance d (cf. Fig. 2.14) est donné par :

$$B_{AS}(T, d, R) = L \int_{A_{acc}} \left\{ \exp\left[-\frac{1}{kT} \sum_{\substack{noeuds \\ (p,q)}}^{N_{tubes}} V\left(\left|\vec{r} - \vec{R}_{pq}\right|, R\right) \right] - 1 \right\} dA \tag{3.31}$$

Où la position des centres des SWNTs est donnée par :

$$\vec{R}_{pq} = p\vec{a} + q\vec{b} = \frac{d}{2}\left[(2p+q)\hat{x} + \sqrt{3}\, q\, \hat{y} \right] \tag{3.32}$$

Dans l'équation (3.31), A_{acc} représente la projection du volume accessible aux molécules d'adsorbat dans un plan perpendiculaire au faisceau. Le B_{AS} normalisé, indépendant de la nature de l'adsorbant et de l'adsorbat (i.e. indépendant de ε, σ, θ), est donné par :

$$B^*_{AS}(T^*, d^*, R^*) = \frac{B_{AS}}{L\sigma^2} = \int_{A^*_{acc}} \left\{ \exp\left[-\frac{1}{T^*} \sum_{\substack{noeuds \\ (p,q)}}^{N_{tubes}} V^*\left(\left| \vec{r}^* - \vec{R}^*_{pq} \right|, R^* \right) \right] - 1 \right\} dA^* \tag{3.33}$$

Où :

$$\begin{aligned} &d^* = d/\sigma \qquad \text{(le pas réduit du réseau)} \\ &\vec{R}^*_{pq} = \vec{R}_{pq}/\sigma \\ &A^*_{acc} = A_{acc}/\sigma^2 \text{ (la région ayant toutes ses dimensions réduites par } \sigma) \\ &\vec{r}^* = \vec{r}/\sigma \end{aligned} \tag{3.34}$$

On remarque des équations précédentes, que les B^*_{AS} et B_{AS} coïncideront lorsque l'on aura $L = 1, \sigma = 1$ et $\varepsilon = k/\pi\theta$ (= 0.8377 lorsque les énergies sont exprimées en Kelvins) :

$$\begin{aligned} &B_{AS} \rightarrow B^*_{AS} = B_{AS}/L\sigma^2 \\ &T \rightarrow T^* = kT/\varepsilon_s{}' \\ &R \rightarrow R^* = R/\sigma \\ &d \rightarrow d^* = d/\sigma \end{aligned} \tag{3.35}$$

Alors, il suffira d'entrer ces valeurs dans un programme calculant le B_{AS} (donné à l'annexe 7) pour qu'il puisse aussi calculer le B^*_{AS} réduit sans avoir à être modifié.

Lorsque nous calculerons le B_{AS} d'un faisceau de SWNTs fermés au moyen de l'équation (3.31), il sera plus facile de définir l'intégrand f de la manière suivante :

$$f = \begin{cases} \exp\left[-\dfrac{1}{kT} \displaystyle\sum_{\substack{noeuds \\ (p,q)}}^{N_{tubes}} V\left(\left| \vec{r} - \vec{R}_{pq} \right|, R \right) \right] - 1 & \text{, si } \vec{r} \text{ est dans la région trouée } A_{acc} \text{ entre les SWNTs} \\ 0 & \text{, si } \vec{r} \text{ est dans un des SWNTs} \end{cases} \tag{3.36}$$

et d'intégrer f sans aucune restriction dans tout le plan XY perpendiculaire aux axes des nanotubes du faisceau. (En pratique, on intègre numériquement l'Équ. (3.36) dans une région

circulaire englobant le faisceau. La région circulaire est environ 14 σ plus large que le faisceau.) L'Équ. (3.36) est équivalente à poser :

$$V_{bundle}(\vec{r}, d, R) = \begin{cases} \sum_{\substack{noeuds \\ (p,q)}}^{N_{tubes}} V\left(\left|\vec{r} - \vec{R}_{pq}\right|, R\right), & \text{si } \vec{r} \text{ est dans la région trouée } A_{acc} \\ 0 & \text{, si } \vec{r} \text{ est dans un des SWNTs fermés du faisceau} \end{cases} \tag{3.37}$$

De plus, si l'on suppose que les nanotubes sont des tiges dures ($V = \infty$ à l'intérieur et $V = 0$ à l'extérieur), l'équation (3.37) donne $V_{bundle} = 0$ dans tout le plan, d'où $B_{AS} = 0$ et $N_a = 0$. I.e. qu'il n'y a pas d'adsorption sur des tiges dures comme on s'y attendrait.

Au lieu de prendre comme volume accessible la région trouée comprise entre les SWNTs fermés du faisceau (ce qui revient à poser $V_{bundle} = 0$ à l'intérieur des SWNTs fermés), il est intéressant de regarder ce qui arriverait si l'on supposait que la totalité de l'espace était accessible aux molécules d'adsorbat. Dans ce cas, on aurait pour des tiges dures que $V_{bundle} = \infty$ à l'intérieur des tiges et que $V_{bundle} = 0$ entre les tiges. La quantité adsorbée en excès deviendrait alors :

$$\begin{aligned} N_a = N - N_0 &= \frac{p}{kT}\left\{\int_{V_{troué}}\left[e^{-0} - 1\right]d\vec{r} + \int_{V_{tubes}}\left[e^{-\infty} - 1\right]d\vec{r}\right\} \\ &= -\frac{pV_{tubes}}{kT} = -N_{0_{tubes}} < 0 \end{aligned} \tag{3.38}$$

ce qui signifie que le gaz est simplement chassé de V_{tubes} après l'ajout des tiges dures. Mais l'adsorption, qui est en fait due à l'attraction des molécules d'adsorbat sur la surface, devrait être nulle dans ce cas. Cet exemple illustre donc qu'il faut considérer seulement le volume accessible aux molécules d'adsorbat lorsqu'on désire calculer l'adsorption en excès N_a.

CHAPITRE 4. ASPECTS NUMÉRIQUES DES CALCULS DES SECONDS COEFFICIENTS DU VIRIEL B_{AS}.

Le calcul du second coefficient du viriel des différentes nanostructures vues précédemment est essentiellement un problème d'intégration numérique. Nous allons donc discuter des algorithmes d'intégration numérique utilisés. Nous verrons également l'importance de trouver une manière efficace pour calculer les fonctions $M_n(x)$ afin que le calcul du B_{AS} d'un faisceau ne soit pas excessivement lent. Nous discuterons également des complications dues à la divergence du potentiel d'adsorption près des parois et de la procédure utilisée pour générer le réseau formé des centres des nanotubes du faisceau.

Les programmes calculant les B_{AS} étant explicitement donnés aux annexes 4 à 7, nous expliquerons ici simplement le fonctionnement général des différents algorithmes.

4.1. Discussion des algorithmes pour les nanostructures ayant un potentiel d'adsorption dépendant d'une seule coordonnée (feuille de graphène, pore en fente, MWNT).

Dans le cas où le potentiel d'adsorption dépend d'une seule variable, l'intégration numérique calculant B_{AS} est faite au moyen de la méthode de Simpson dont les détails sont donnés dans les références [42, 43]. La figure suivante introduit les notations utilisées lorsqu'on intègre une fonction f(x) sur un intervalle [a, b] :

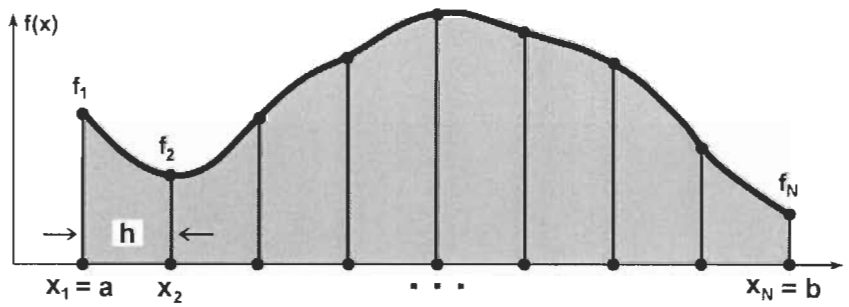


Figure 4.1. Notations utilisées dans la règle du trapèze et de Simpson.

La règle du trapèze, obtenue en faisant passer une droite entre (x_1, f_1) et (x_2, f_2), suivie de l'intégration sur $[x_1, x_2]$ est donnée par :

$$\int_{x_1}^{x_2} f(x)\,dx = h\left[\frac{1}{2}f_1 + \frac{1}{2}f_2\right] + O(h^3 f'') \tag{4.1}$$

La règle de Simpson est obtenue comme l'équation précédente, sauf qu'on fait passer une parabole entre les points (x_1, f_1), (x_2, f_2), (x_3, f_3) :

$$\int_{x_1}^{x_3} f(x)\,dx = h\left[\frac{1}{3}f_1 + \frac{4}{3}f_2 + \frac{1}{3}f_3\right] + O(h^5 f^{(4)}) \tag{4.2}$$

Les deux règles précédentes peuvent être "étendues" en les appliquant sur des sous-intervalles consécutifs :

$$\int_{x_1}^{x_N} f(x)\,dx = h\left[\frac{1}{2}f_1 + f_2 + f_3 + \ldots + f_{N-1} + \frac{1}{2}f_N\right] + O\left(\frac{1}{N^2}\right) \quad \text{(où N est quelconque)} \tag{4.3}$$

$$\int_{x_1}^{x_N} f(x)\,dx = h\left[\frac{1}{3}f_1 + \frac{4}{3}f_2 + \frac{2}{3}f_3 + \frac{4}{3}f_4 + \ldots + \frac{2}{3}f_{N-2} + \frac{4}{3}f_{N-1} + \frac{1}{3}f_N\right] + O\left(\frac{1}{N^4}\right) \tag{4.4}$$

(où N est impair)

L'avantage de la règle du trapèze étendue est qu'il est possible de doubler le nombre de sous-intervalles utilisés pour l'évaluer sans devoir recalculer les f_i à tous les points. Seule l'évaluation des nouveaux f_i (situés entre les anciens points (x_i, f_i)) sera nécessaire, ce qui diminuera de moitié le travail requis pour l'ordinateur. En effet, dénotant par I_{i1} la règle du trapèze obtenue par i bissections consécutives des sous-intervalles, on obtient aisément les résultats suivants :

$$I_{01} = \frac{b-a}{2}\left[f_a + f_b\right] \tag{4.5}$$

$$\begin{aligned} I_{i1} &= \frac{1}{2}\left[I_{i-1,1} + \frac{b-a}{2^{i-1}}\sum_{k=1,3,\ldots}^{2^i-1} f\left(a + \frac{b-a}{2^i}k\right)\right] \quad ; \quad i = 1, 2, \ldots \\ &= \frac{1}{2}I_{\text{précédente}} + h\cdot(\text{somme des nouveaux } f_i) \end{aligned} \tag{4.6}$$

Alors, on peut évaluer successivement la série I_{01}, I_{11}, I_{21} des intégrales "trapèzes" jusqu'à ce qu'une certaine précision ERREUR_RELATIVE soit atteinte sur la valeur de l'intégrale recherchée $\int_a^b f(x)\,dx$:

$$\left|\frac{I_{\text{courante}} - I_{\text{précédente}}}{I_{\text{précédente}}}\right| = \left|\frac{I_{i,1} - I_{i-1,1}}{I_{i-1,1}}\right| < \text{ERREUR\_RELATIVE} \tag{4.7}$$

La méthode de Simpson donnée à l'équation (4.4) a été utilisée pour faire l'intégration nécessaire au calcul des B_{AS} et des fonctions $M_n(x)$ (lorsque les développements en série n'ont pu être utilisés) dans ce travail. Ce qu'il y a de très intéressant, c'est que deux règles du trapèze successives I_{i1} et $I_{i+1,1}$ peuvent être utilisées pour la calculer. En effet, dénotant par I_{i2} la $(i+1)^{ème}$ règle de Simpson, on vérifie aisément que :

$$I_{i,2} = \frac{4I_{i+1,1} - I_{i,1}}{3} \quad ; \quad i = 0, 1, \ldots \tag{4.8}$$

Qui devient dans le cas particulier i = 1 :

$$I_{12} = \frac{4}{3}\left(h\left[\frac{1}{2}f_1 + f_2 + f_3 + f_4 + \frac{1}{2}f_5\right]\right) - \frac{1}{3}\left(2h\left[\frac{1}{2}f_1 + f_3 + \frac{1}{2}f_5\right]\right)$$
$$= h\left[\frac{1}{3}f_1 + \frac{4}{3}f_2 + \frac{2}{3}f_3 + \frac{4}{3}f_4 + \frac{1}{3}f_5\right]$$

L'équation (4.8) permet de calculer l'intégrale au moyen de la règle de Simpson étendue. Cette règle est exacte lorsque f(x) est un polynôme d'ordre 3 ou moins. Puisque dans ce travail les fonctions à intégrer (pour les $M_n(x)$ et les B_{AS}) ne sont pas très lisses, seules les règles du trapèze et de Simpson ont été utilisées pour effectuer les intégrations numériques.

Maintenant que nous savons comment effectuer l'intégration pour calculer les B_{AS}, nous devons aborder le problème dû au fait que le potentiel d'adsorption diverge à l'infini lorsqu'on se rapproche de la surface de l'adsorbant. Dans ce cas, nous savons que près des parois $f = \exp(-\beta V) - 1 \to -1$ de sorte que nous définirons dans les programmes une DISTANCE_REDUITE_EXCLUSION telle que la fonction f y prenne la valeur -1 à l'intérieur de cette distance (mesurée à partir de la paroi et réduite par σ). Dans le cas où V dépend d'une seule coordonnée, nous avons fixé cette distance à 0.01 de sorte que le potentiel y soit très répulsif (cette très petite distance évite des complications éventuelles lorsque la température choisie pour le calcul du B_{AS} est très élevée).

4.2. Discussion des algorithmes pour le faisceau de SWNTs.

Comme nous avons vu au chapitre 3 (cf. Équ. (3.31)), le calcul du B_{AS} dans le cas d'un faisceau de SWNTs consiste essentiellement à effectuer une intégration numérique d'une fonction

f(x, y) sur une région R du plan XY. La région R enveloppe le faisceau et est suffisamment grande (un cercle de diamètre 14 σ plus grand que le faisceau) pour inclure pratiquement toute l'adsorption qu'on peut avoir sur le faisceau. Nous avons étudié deux méthodes numériques pour faire cette intégrale: la méthode de Monte Carlo et celle du trapèze imbriquée [42].

Après avoir fait des tests sur une fonction possédant des pics étroits (comme ça sera le cas pour l'intégrand du B_{AS} aux faibles températures), nous nous sommes aperçus que la méthode du trapèze imbriquée est beaucoup plus efficace que celle de Monte Carlo. Cette méthode consiste à effectuer une intégrale bidimensionnelle au moyen de deux intégrales "trapèze" imbriquées l'une dans l'autre. La méthode du trapèze imbriquée est obtenue ci-dessous et est donnée dans les fichiers trapeze_2D.h, trapeze_2D.cpp et trapeze_2D_interne.cpp (cf. programme à l'annexe 7 calculant le B_{AS} d'un faisceau de SWNTs) :

$$I = \iint_R f(x, y)\, dS = \int_{y=y_{min}}^{y_{max}} dy \left[\int_{x=\text{frontiere\_gauche}(y)}^{\text{frontiere\_droite}(y)} f(x, y)\, dx \right]$$
$$= \int_{y=y_{min}}^{y_{max}} F(y)\, dy \quad \text{où} \quad F(y) = \int_{\text{frontiere\_gauche}(y)}^{\text{frontiere\_droite}(y)} f(x, y)\, dx \tag{4.9}$$

La méthode du trapèze imbriquée possédera donc deux critères de convergence : ERREUR_RELATIVE_TRAPEZE_INTERNE et ERREUR_RELATIVE_TRAPEZE_EXTERNE. Les deux intégrales imbriquées de l'équation (4.9) se feront simplement au moyen des équations (4.5) et (4.6). L'intégration de la fonction $f = \exp(-\beta V) - 1$ est illustrée à la figure suivante pour un faisceau de SWNTs :

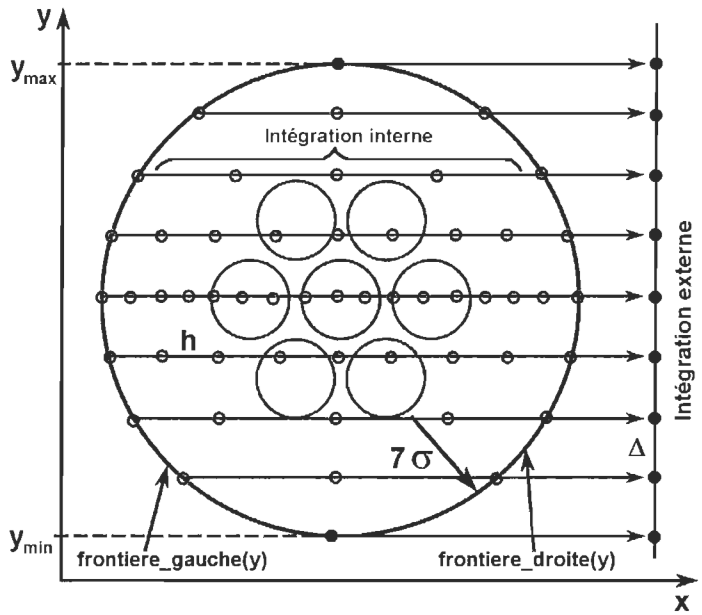


Figure 4.2. Calcul du B_{AS} d'un faisceau de SWNTs avec deux règles du trapèze imbriquées.

Après plusieurs ajustements minutieux, l'erreur relative sur les fonctions $M_n(x)$ apparaissant dans le potentiel V(r, R) des SWNTs a été fixée à 10^{-4} %. L'erreur relative tolérée sur les intégrales internes et externes ont été fixées à 0.1 % et à 1 % respectivement et la région circulaire R a un diamètre 14 σ plus grand que celui du faisceau.

Maintenant que nous avons un algorithme permettant d'intégrer une fonction f(x, y), nous devons résoudre un dernier problème. La fonction *f*, en plus de dépendre sur (x, y) dépendra également d'autres variables (comme la température) qui sont gardées fixes lors de l'intégration. Il suffira alors de passer ces paramètres dans la fonction au moyen de variables globales (mais demeurant locales au fichier source). Nous pourrons alors utiliser l'algorithme conçu pour intégrer f(x, y) dans le cas où nous avons f(x, y, constantes). Cette idée est illustrée ci-dessous pour l'intégrand du B_{AS} :

```
/************************* integrand_Bas.cpp ***************************/

        /* Variables externes, mais qui restent définies seulement à
        l'intérieur de ce fichier source.  Ces variables apparaissent
        dans l'integrand_Bas comme des constantes lors de l'intégration
        sur le plan XY. */

static double T, R, theta, eps, sigma;
static int N_reseau, flag_fermer;
static double** MatriceReseau;


double integrand_Bas(double x, double y) {

        /*  ... Instructions dépendant des constantes. ...  */

}


void Fixer_ctes_integrand_Bas(double T_, double R_,
        double** MatriceReseau_, int N_reseau_,
        int flag_fermer_, double theta_,
        double eps_, double sigma_) {

                /* Fonction servant à fixer les constantes. */

        T = T_;
        R = R_;
        MatriceReseau = MatriceReseau_;
        N_reseau = N_reseau_;
        flag_fermer = flag_fermer_;
        theta = theta_;
        eps = eps_;
        sigma = sigma_;
}
```

Les centres des nanotubes du faisceau sont disposés sur un réseau hexagonal. L'algorithme donné ci-dessous (dans sa version Mathematica) enroule les nanotubes en couches concentriques en suivant l'ordre des flèches (cf. Fig. 4.3) :

```
PtsDuReseau[d_, Ntubes_] := Module[{MatricePtsReseau, j, i, c, k1, θ, NbrePtsSurc},
  j = 1;                                (* Compte le nombre de pts du réseau placés *)
  MatricePtsReseau = {{0, 0}};          (* Placer le point central pour commencer *)
  If[j == Ntubes, Return[MatricePtsReseau]];
      (* Remplir les couches "i" du réseau *)
  For[i = 1, i < Infinity, i++,
   j++;
   MatricePtsReseau = Append[MatricePtsReseau, {i * d, 0}];
   If[j == Ntubes, Return[MatricePtsReseau]];
       (* Placer i pts sur les 5 premiers côtés "c" de la couche i et i-1 pts sur le 6ème *)
   For[c = 1, c ≤ 6, c++,
    θ = π / 3 + c * (π / 3);
    If[c ≤ 5, NbrePtsSurc = i, NbrePtsSurc = i - 1];
    For[k1 = 1, k1 ≤ NbrePtsSurc, k1++,
     j++;
     MatricePtsReseau = Append[MatricePtsReseau, {MatricePtsReseau[[j - 1, 1]] + d * Cos[θ],
        MatricePtsReseau[[j - 1, 2]] + d * Sin[θ]}];
     If[j == Ntubes, Return[MatricePtsReseau]];
    ];
   ];
  ];
 ]
```

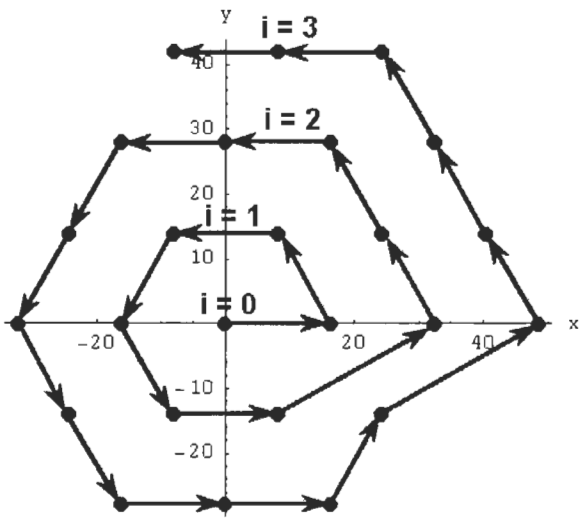


Figure 4.3. Enroulement de 25 SWNTs en faisceau de couches "i" concentriques.

Lorsqu'on regarde les équations (4.9), (3.31) et (3.25), on constate qu'il y aura en fait trois "couches" d'intégration imbriquées (sans compter la somme de V(r, R) sur tous les tubes du faisceau…). Il est donc primordial d'optimiser au maximum la vitesse d'évaluation du potentiel

V(r, R) en tentant d'éliminer le plus possible les intégrations calculant les fonctions $M_n(x)$ au moyen de développements en séries. Nous utiliserons le développement en série de l'intégrale suivante [44] :

$$\int_0^{\pi} \frac{d\phi}{\left(1+b^2-2b\cos\phi\right)^p} = \pi\, F(p,\, p,\, 1;\, b^2) \quad ; \quad \left(\text{qui converge pour } |b|<1\right) \tag{4.10}$$

où $p = n/2$ et où F est la fonction hypergéométrique [40] :

$$F(a,b,c;u) = \sum_{m=0}^{\infty} \frac{(a)_m\,(b)_m}{(c)_m}\frac{u^m}{m!} \quad ; \quad (a)_m = \frac{(a+m-1)!}{(a-1)!} \quad ; \quad \left(\text{convergente pour } |u|<1\right) \tag{4.11}$$

Puisque :

$$M_n(x) = \int_0^{\pi} \frac{d\phi}{\left(1+x^2-2x\cos\phi\right)^{n/2}} = \frac{1}{x^n}\int_0^{\pi} \frac{d\phi}{\left(1+\left(x^{-1}\right)^2-2\left(x^{-1}\right)\cos\phi\right)^{n/2}} \tag{4.12}$$

nous obtenons les développements en séries suivants :

$$\begin{aligned} M_n(x) &= \pi\, F\left(n/2,\, n/2,\, 1;\, x^2\right); \quad \text{qui converge pour } 0 \le x < 1 \\ M_n(x) &= \frac{\pi}{x^n} F\left(n/2,\, n/2,\, 1;\, x^{-2}\right); \quad \text{qui converge pour } x > 1 \end{aligned} \tag{4.13}$$

Ces séries peuvent être écrites sous une forme plus explicite au moyen de l'équation (4.11):

$$\begin{aligned} M_5(x) &= \pi + \frac{25}{4}\pi x^2 + \frac{1225}{64}\pi x^4 + \ldots \\ M_{11}(x) &= \pi + \frac{121}{4}\pi x^2 + \frac{20449}{64}\pi x^4 + \ldots \end{aligned} \quad ; \text{ centrées à } x = 0 \text{ et convergentes pour } 0 \le x < 1 \tag{4.14}$$

$$\begin{aligned} M_5(x) &= \frac{\pi}{x^5} + \frac{\frac{25}{4}\pi}{x^7} + \frac{\frac{1225}{64}\pi}{x^9} + \ldots \\ M_{11}(x) &= \frac{\pi}{x^{11}} + \frac{\frac{121}{4}\pi}{x^{13}} + \frac{\frac{20449}{64}\pi}{x^{15}} + \ldots \end{aligned} \quad ; \text{ centrées à } x = 0 \text{ et convergentes pour } x > 1 \tag{4.15}$$

La convergence de ces séries de Laurent, ayant pour centre x = 0, peut être grandement améliorée en utilisant la transformation [44] :

$$F(p,\, p,\, 1;\, b^2) = \frac{1}{\left(1-b^2\right)^{2p-1}} F(1-p,\, 1-p,\, 1;\, b^2) \quad ; \text{ (valide pour } |b| < 1) \tag{4.16}$$

qui donne les développements en séries :

$$M_n(x) = \frac{\pi}{\left(1-x^2\right)^{n-1}} F\left(1-n/2,\, 1-n/2,\, 1;\, x^2\right) \quad ; \text{ (convergente pour } 0 \le x < 1) \tag{4.17}$$

$$M_n(x) = \frac{\pi\, x^{n-2}}{\left(x^2-1\right)^{n-1}} F\left(1-n/2,\, 1-n/2,\, 1;\, x^{-2}\right) \quad ; \text{ (convergente pour } x > 1) \tag{4.18}$$

Ne gardant que les huit premiers termes, ces séries permettent de calculer les fonctions $M_n(x)$ avec une erreur relative toujours moindre que 8.60×10^{-6} % (cf. tableau 4.2) :

$$M_n(x) = \frac{\pi}{\left(1-x^2\right)^{n-1}} \sum_{i=0}^{7} a_{ni} x^{2i} \quad ; \quad n = 5 \text{ ou } 11 \quad ; \quad 0 \le x < 1 \tag{4.19}$$

$$M_n(x) = \frac{\pi\, x^{n-2}}{\left(1-x^2\right)^{n-1}} \sum_{i=0}^{7} a_{ni} x^{-2i} \quad ; \quad n = 5 \text{ ou } 11 \quad ; \quad x > 1 \tag{4.20}$$

où les coefficients a_{ni} sont donnés dans le tableau ci-dessous :

Tableau 4.1 Coefficients des séries utilisées pour calculer les fonctions $M_n(x)$.

n	a_{n0}	a_{n1}	a_{n2}	a_{n3}	a_{n4}	a_{n5}	a_{n6}	a_{n7}
5	1	9/4	9/64	1/256	9/16384	9/65536	49/1048576	81/4194304
11	1	81/4	3969/64	11025/256	99225/16384	3969/65536	441/1048576	81/4194304

L'erreur maximale obtenue sur les fonctions $M_n(x)$ est donnée ci-après pour les différentes plages de valeurs de x :

Tableau 4.2 Précision et méthode de calcul des fonctions $M_n(x)$ pour les différentes valeurs de x.

n	Valeurs de x	Erreur relative maximale (%)	Équations utilisées pour évaluer $M_n(x)$
5	$0 \le x < 0.24$	Précision d'un "double" ($< 10^{-13}$)	(4.19)
	$0.24 \le x < 0.77$	8.60×10^{-6}	(4.19)
	$0.77 \le x \le 1.30$	10^{-4}	(4.12 a), (4.8), (4.6), (4.5)
	$1.30 < x \le 4.5$	8.47×10^{-6}	(4.20)
	$4.5 < x$	Précision d'un "double" ($< 10^{-13}$)	(4.20)
11	$0 \le x < 0.28$	Précision d'un "double" ($< 10^{-13}$)	(4.19)
	$0.28 \le x < 0.9999$	1.70×10^{-6}	(4.19)
	$0.9999 \le x \le 1.0001$	10^{-4}	(4.12 a), (4.8), (4.6), (4.5)
	$1.0001 < x \le 4.0$	1.70×10^{-6}	(4.20)
	$4.0 < x$	Précision d'un "double" ($< 10^{-13}$)	(4.20)

Le fait que les séries (4.19) et (4.20) aient leur terme dominant différent de ceux des séries données plus tôt à l'équation (2.26) n'a en fait rien de surprenant. Les séries (4.19) et (4.20) découlent en fait des séries de Laurent (4.14) et (4.15) qui ont pour centre x = 0, tandis que les séries données à l'équation (2.26) sont centrées autour de x = 1. Comme cela est expliqué dans la référence [45], il est possible qu'une même fonction (ici $M_n(x)$) ait des développements en séries de Laurent différents pour des centres différents ou des régions de convergence différentes.

En regardant de plus près les équations (4.19) et (4.20), on voit que ce sont essentiellement des polynômes. La manière simple de les évaluer est d'additionner ses différentes puissances au moyen de l'instruction x^{2i} = pow(x, 2i) par ordre décroissant du degré 2i afin d'éviter les erreurs d'arrondis. Cependant, puisque les fonctions pow( ) sont lentes à évaluer, cette manière de calculer les polynômes est à éviter [42]. Nous les avons donc calculé avec le plus petit nombre d'opérations élémentaires (+, −, ×, ÷) possible. Voici donc la manière très efficace pour évaluer $M_5(x < 1)$ avec 19 opérations élémentaires (7+, 1−, 10×, 1÷). (Cf. module Mn.cpp, programme calculant le B_{AS} d'un faisceau de SWNTs, à l'annexe 7) :

$$X2 = x * x,\ \overline{X} = 1 - X2,\ \overline{X}2 = \overline{X} * \overline{X},\ \overline{X}4 = \overline{X}2 * \overline{X}2$$
$$M5 = \frac{((((((\tilde{a}_{57} * X2 + \tilde{a}_{56}) * X2 + \tilde{a}_{55}) * X2 + \tilde{a}_{54}) * X2 + \tilde{a}_{53}) * X2 + \tilde{a}_{52}) * X2 + \tilde{a}_{51}) * X2 + \tilde{a}_{50}}{\overline{X}4} \quad (4.21)$$

Où le facteur π a été préalablement inclus dans les coefficients $\tilde{a}_{ni}$.

Lorsque nous calculons le potentiel V(r, R) au moyen des fonctions $M_n(x)$ données au tableau 4.2, nous devrons encore définir une "zone d'exclusion" près des parois des nanotubes. À l'intérieur de cette zone d'exclusion, nous poserons que $f = \exp(-\beta V) - 1 \to -1$ puisque l'intégrale numérique calculant $M_n(x)$ n'arrive pas à converger lorsque $|x - 1| < 10^{-4}$. Cette zone d'exclusion devra être également suffisamment étroite pour que f y demeure très près de -1 même lorsque la température devient très élevée. La distance d'exclusion autour d'un SWNT du faisceau est choisie telle que :

$$\left| \frac{f(r) - (-1)}{f(r)} \right| < 10^{-7} \quad (4.22)$$

Qui donne la condition suivante, lorsqu'on est situé dans la zone d'exclusion :

$$V(r, R) > (7 \ln 10)\, kT \quad (4.23)$$

L'équation précédente montre que la DISTANCE_REDUITE_EXCLUSION dépendra du rayon du nanotube en plus de la température. La figure suivante montre que la distance d'exclusion diminue lorsque la température augmente ou lorsque le rayon du nanotube diminue :

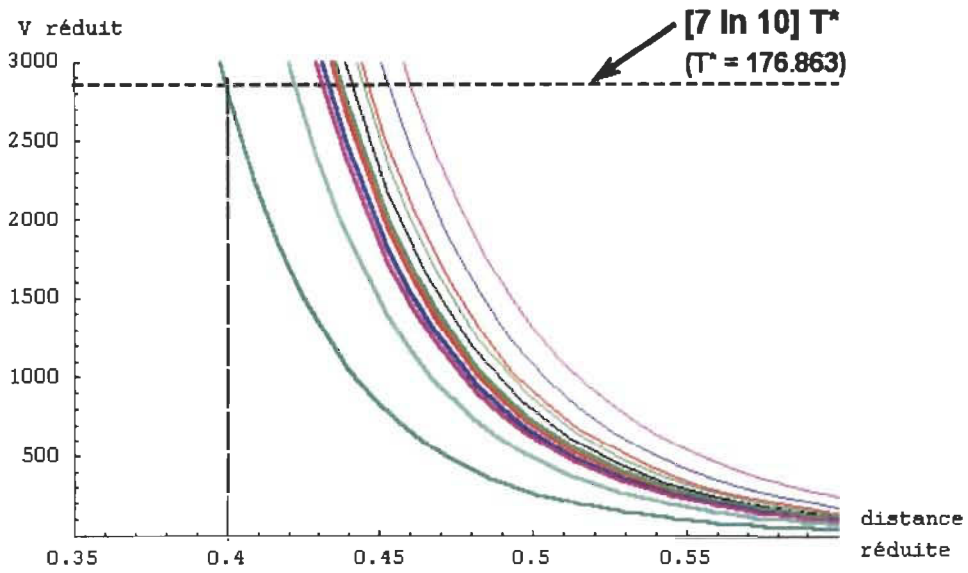


Figure 4.4. Détermination de la distance d'exclusion autour des parois d'un SWNT où $f \to -1$. Les courbes épaisses (minces) donnent le potentiel réduit à l'extérieur (intérieur) du SWNT. De gauche à droite, le rayon réduit du nanotube vaut 0.05, 0.3, 0.8, 1.086, 2.038, 3.0, ∞, 3.0, 2.038, 1.086, 0.8.

De la figure précédente, on voit qu'une DISTANCE_REDUITE_EXCLUSION de 0.40 est bonne tant que la température réduite est inférieure à 176.863 (65531 K pour H_2) et tant que le rayon réduit du tube est supérieur à 0.05 (0.1595 Å pour H_2). Cependant, le rayon réduit du nanotube devra être alors inférieur à 4000 (12760 Å pour H_2) afin que la zone d'exclusion soit assez large pour que la condition $|x-1| > 10^{-4}$ sur la convergence des $M_n(x)$ soit toujours respectée.

CHAPITRE 5. RÉSULTATS.

5.1. Comportement du second coefficient du viriel pour les différentes nanostructures étudiées.

Dans cette section, nous allons montrer le comportement du B_{AS} des nanostructures vues précédemment en fonction de leurs dimensions et de la température.

5.1.1. Comportement du B_{AS} d'une feuille de graphène.

Dans ce cas, le second coefficient du viriel réduit dépendra uniquement de la température réduite (cf. Équ. (3.14)). Son comportement, indépendant de la nature de l'adsorbant et de l'adsorbat, est montré sur le graphique ci-dessous. On voit que dans la limite des faibles températures, B_{AS} aura la forme $B_{AS} \propto e^{D/kT}$:

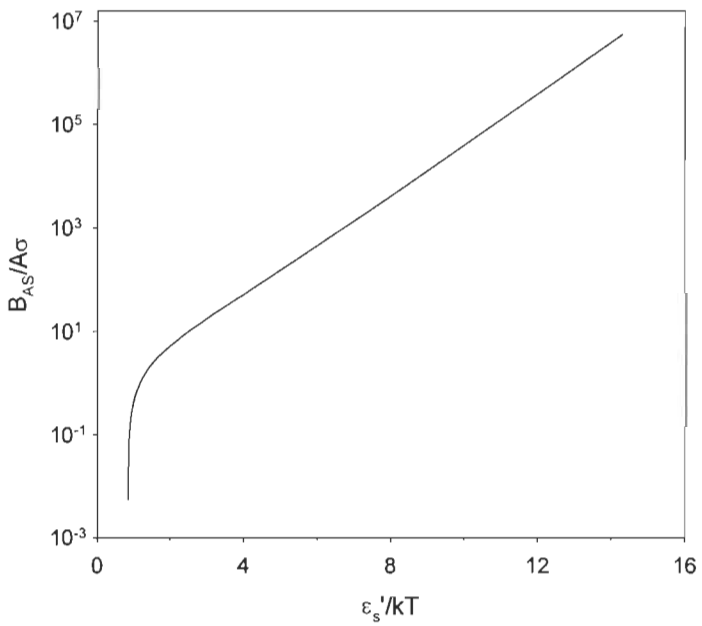


Figure 5.1. Comportement du B_{AS} d'une feuille de graphène en fonction de la température.

5.1.2. Comportement du B_{AS} d'un pore en fente (du charbon activé).

Dans ce cas, le second coefficient du viriel réduit dépendra de la largeur réduite d* du pore en plus de la température réduite (cf. Équ. (3.20)). Son comportement en fonction de la

température (cf. Fig. 5.2 a) est le même que celui d'une feuille de graphène ($B_{AS} \propto e^{D/kT}$ quand $T \to 0$) sauf que la pente de la partie linéaire est plus élevée.

La figure 5.2 b montre la dépendance de B_{AS} en fonction de la largeur réduite du pore. On constate que dans la limite des faibles densités, le gain sur la quantité de gaz adsorbé par rapport à la feuille de graphène est de 5.4×10^4 fois lorsque la température réduite est de 0.1 (37 K pour H_2) mais qu'il diminue rapidement lorsque celle-ci augmente (99 fois pour T* = 0.208, la pression dans le contenant étant la même dans les deux cas). Le graphique montre également qu'aux basses températures, la largeur optimale correspond sensiblement au puits de potentiel le plus profond (d* = 2) mais qu'elle augmente avec la température (d^*_{max} = 2.20 pour T* = 0.810).

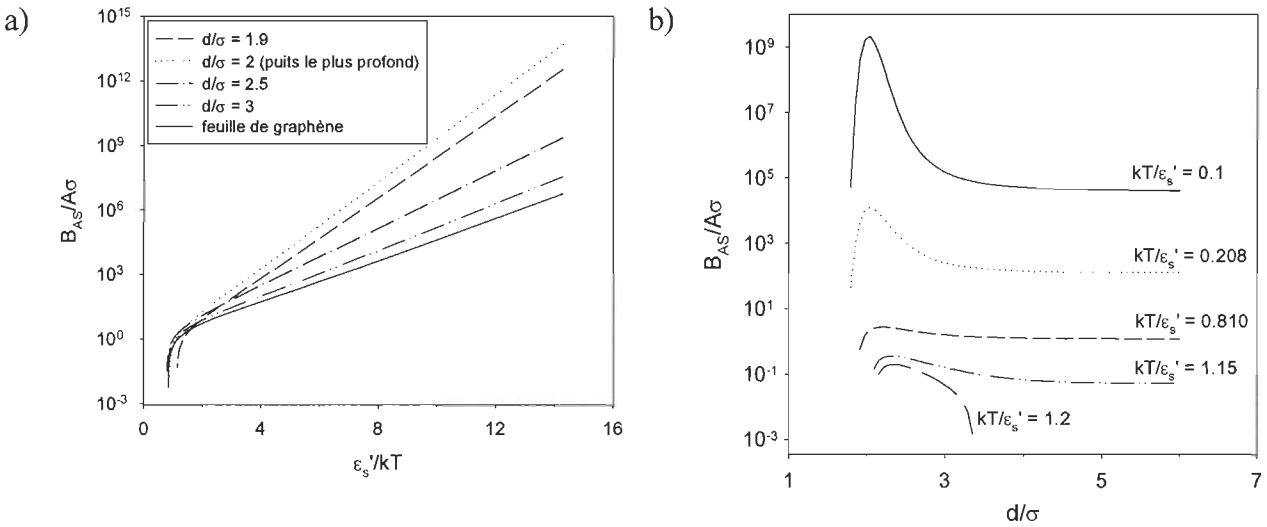


Figure 5.2. Comportement du B_{AS} d'un pore en fente en fonction de la température et de sa largeur.

5.1.3. Comportement du B_{AS} d'un nanotube à paroi simple.

Le second coefficient du viriel réduit dépendra du rayon réduit R* du nanotube et de la température réduite (cf. Équ. (3.27)). Son comportement en fonction de la température a le même aspect que celui d'une feuille de graphène et d'un pore en fente ($B_{AS} \propto e^{D/kT}$ quand $T \to 0$).

La figure 5.3 montre qu'aux faibles températures (T* = 0.208; 77 K pour H_2), le nanotube ouvert à paroi simple (o−SWNT) optimal a un rayon correspondant pratiquement au puits de

potentiel le plus profond à l'intérieur du tube (situé à 1.086σ) et que le B_{AS} y est alors environ 1700 fois plus élevé que pour une feuille de graphène équivalente de même aire, ce qui représente une amélioration considérable. Lorsque le rayon devient grand, les courbes pour le tube ouvert et fermé se rapprochent, indiquant qu'ils se comportent essentiellement comme une surface plane de graphène. Leur légère pente positive (e.g. T* = 0.810) provient alors de l'augmentation de la surface du nanotube lorsque son rayon augmente.

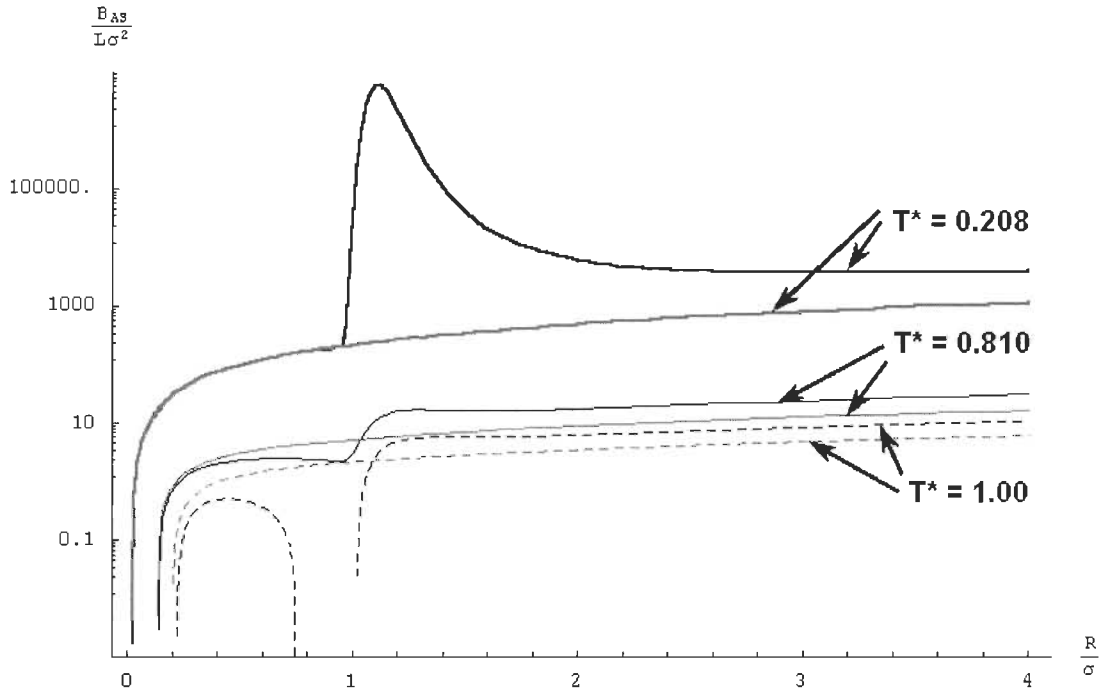


Figure 5.3. Comportement du B_{AS} d'un SWNT en fonction de son rayon. Les courbes noires (grises) sont pour le nanotube ouvert (fermé).

5.1.4. Comportement du B_{AS} d'un faisceau de nanotubes à parois simples.

Le second coefficient du viriel réduit dépendra du nombre de nanotubes formant le faisceau, du pas d* du réseau formé de leurs centres, de leur rayon réduit R* et de la température réduite (cf. Équ. (3.33)). Dans cette section, nous ferons donc varier un seul de ces paramètres à la fois pour obtenir des courbes du comportement de B_{AS} pour quelques cas types. La présentation complète des courbes de B_{AS} obtenues lors de ce travail a été reportée à l'annexe 3. Dans tous les graphiques qui seront présentés dans cette sous-section, les courbes noires (grises) seront utilisées lorsque les SWNTs sont ouverts (fermés) aux extrémités. Les courbes continues (en tirets) seront pour N_{tubes} SWNTs regroupés en faisceau (isolés).

Les principales températures étudiées sont celles-ci : 32.98 K (température critique de H_2), 77.40 K (ébullition de N_2), 194.6 K (sublimation du CO_2) et 300 K (température ambiante). Les rayons de nanotubes étudiés sont : 3.464 Å (puits de potentiel le plus creux pour H_2) et 6.5 Å (rayon typique). L'espacement entre les centres des nanotubes a été le plus souvent fixé à 16.7 Å. Le nombre de nanotubes formant le faisceau a été fixé à 61 (4 couches pleines concentriques), 91 (5 couches pleines) ou 25 ($3^{ème}$ couche incomplète).

5.1.4.1. Comportement du B_{AS} en fonction du pas du réseau.

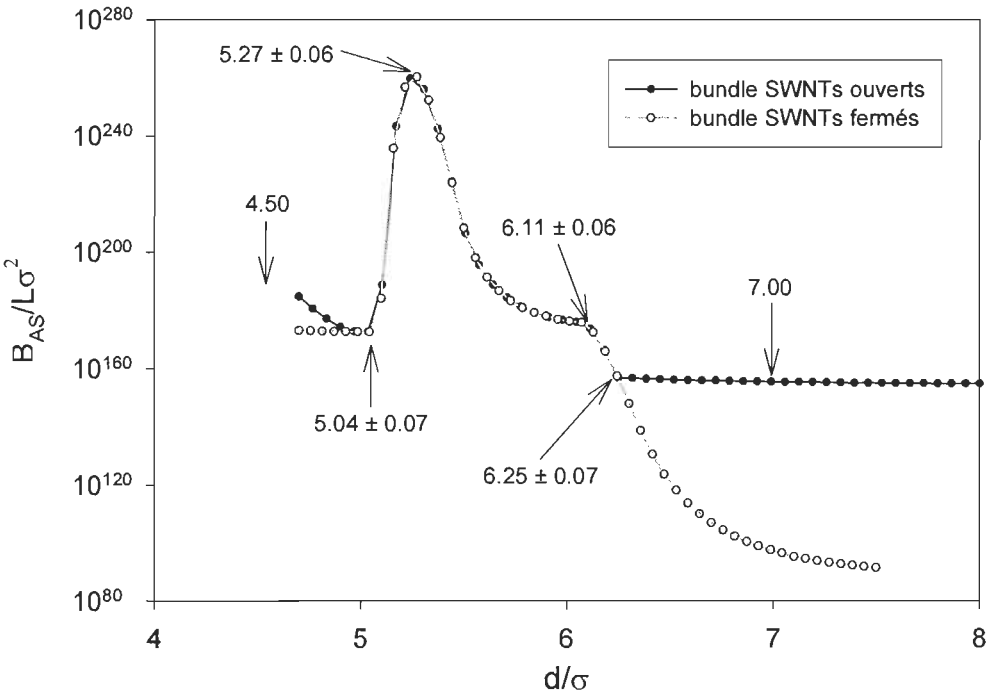


Figure 5.4. Comportement du B_{AS} en fonction du pas du réseau. $N_{tubes} = 25$, $R^* = 2.038$ (H_2 : 6.5 Å), $T^* = 0.005$ (H_2 : 1.85 K).

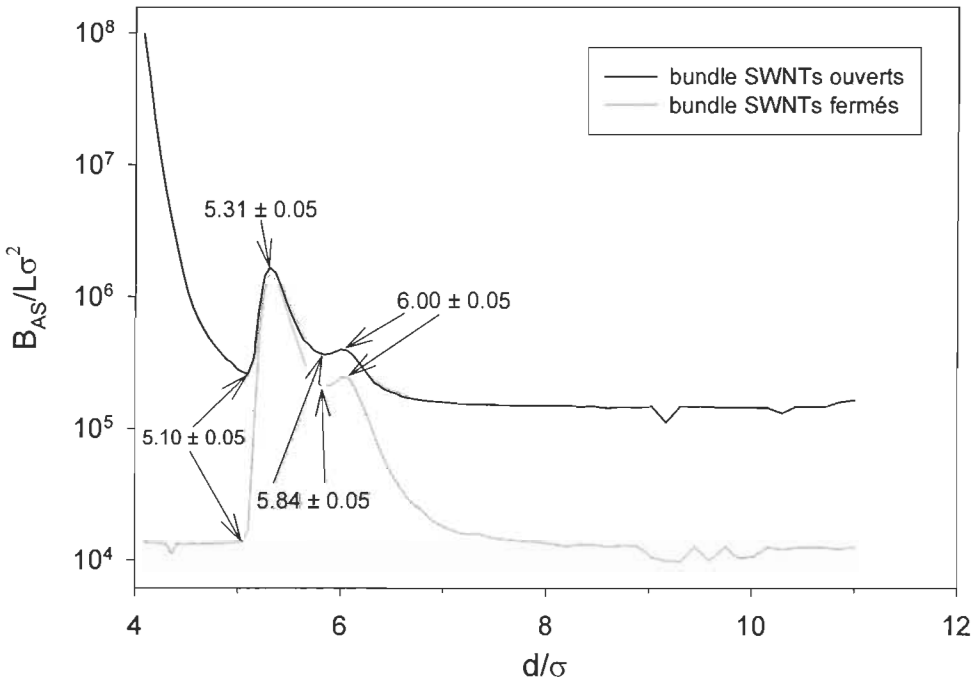


Figure 5.5. Comportement du B_{AS} en fonction du pas du réseau. $N_{tubes} = 25$, $R^* = 2.038$ (H_2 : 6.5 Å), $T^* = 0.2089$ (H_2 : 77.40 K).

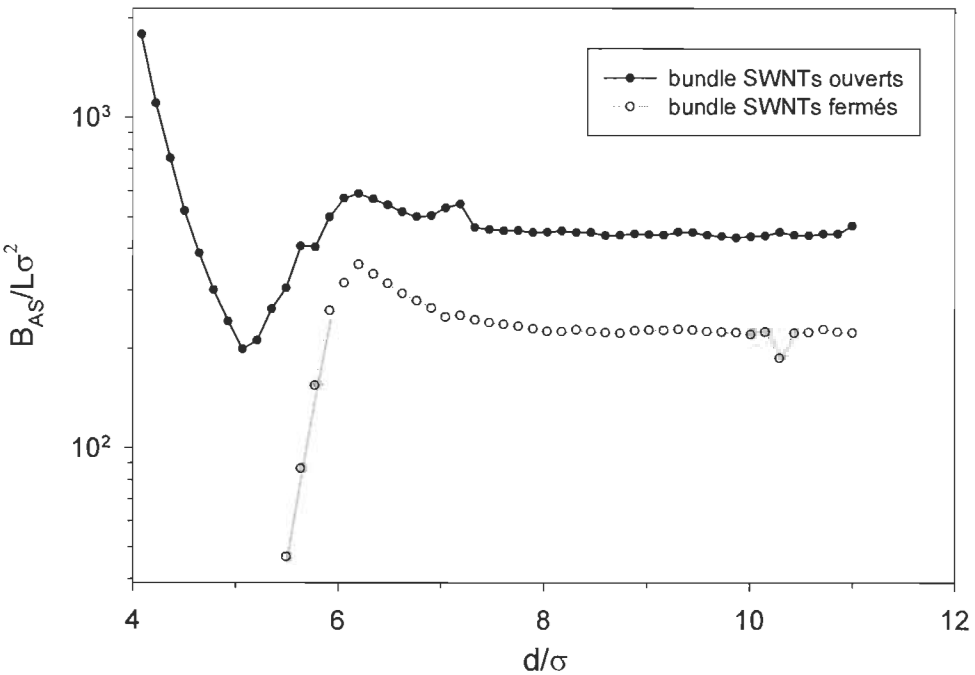


Figure 5.6. Comportement du B_{AS} en fonction du pas du réseau. $N_{tubes} = 25$, $R^* = 2.038$ (H_2 : 6.5 Å), $T^* = 0.8097$ (H_2 : 300.0 K).

5.1.4.2. Comportement du B_{AS} en fonction du rayon des nanotubes.

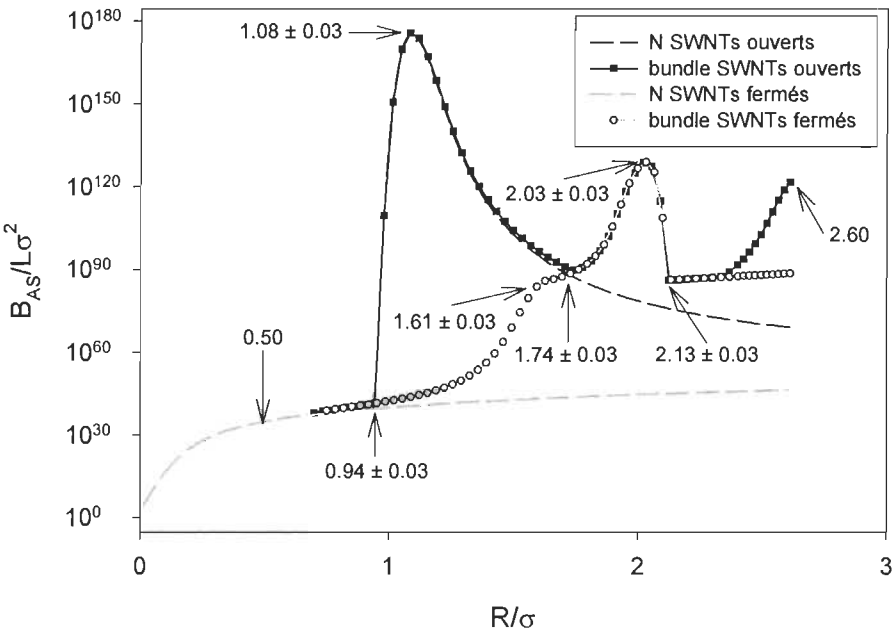


Figure 5.7. Comportement du B_{AS} en fonction du rayon des nanotubes. $N_{tubes} = 25$, $d^* = 5.235$ (H_2 : 16.7 Å), $T^* = 0.01$ (H_2 : 3.71 K).

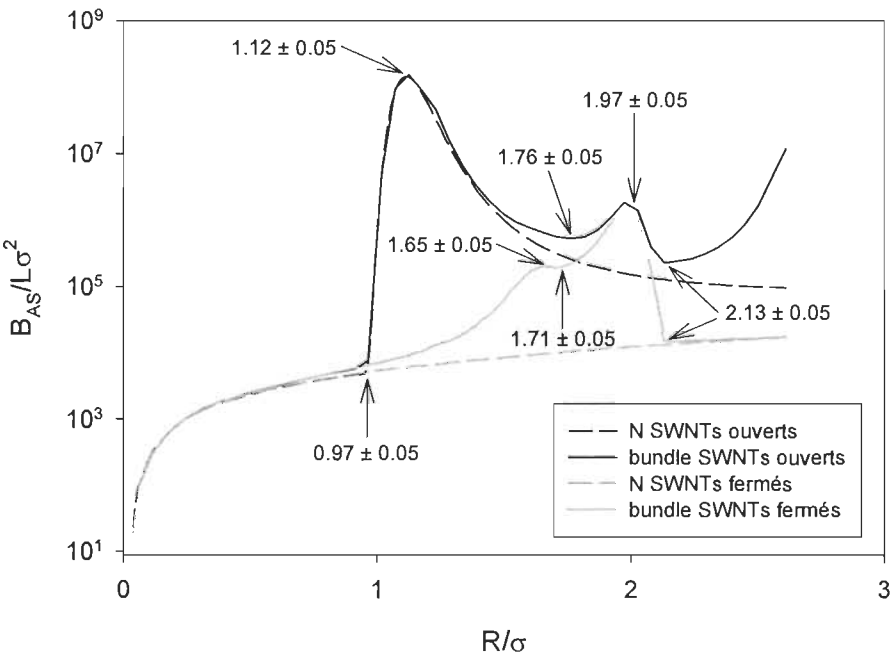


Figure 5.8. Comportement du B_{AS} en fonction du rayon des nanotubes. $N_{tubes} = 25$, $d^* = 5.235$ (H_2 : 16.7 Å), $T^* = 0.2089$ (H_2 : 77.40 K).

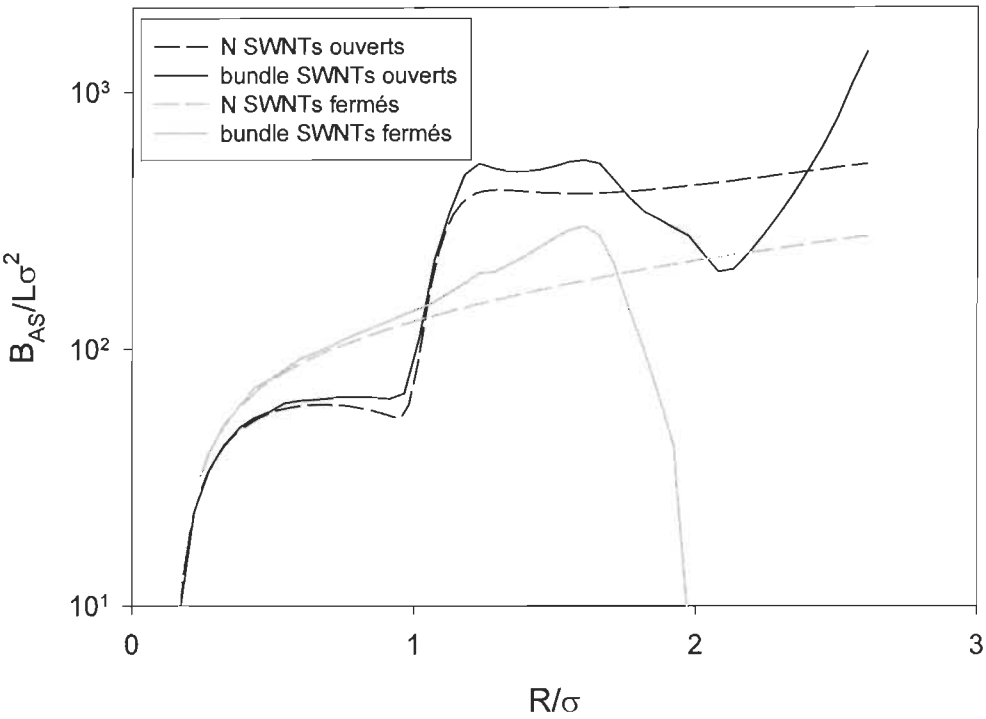


Figure 5.9. Comportement du B_{AS} en fonction du rayon des nanotubes. $N_{tubes} = 25$, $d^* = 5.235$ (H_2 : 16.7 Å), $T^* = 0.8097$ (H_2 : 300.0 K).

5.1.4.3. Comportement du B_{AS} en fonction de la température.

Nous verrons à la section 5.2 que la pente $(-E^*_A)$ de la partie linéaire des graphiques de $\ln B^*_{AS}$ en fonction de $1/T^*$ coïncide avec la profondeur du puits de potentiel dans les différents sites d'adsorption ($B^*_{AS} \propto e^{D^*/T^*}$ quand $T^* \to 0$). Les graphiques ci-dessous montrent donc l'écart relatif Δ entre E^*_A et la profondeur du puits de potentiel correspondant (cf. Fig. 2.15 et tableau 2.2) : Δ_{1i} (intérieur d'un seul SWNT), Δ_{1e} (extérieur d'un seul SWNT), Δ_T (intérieur d'un SWNT du faisceau), Δ_I (interstices entre les SWNTs du faisceau), Δ_P (site périphérique).

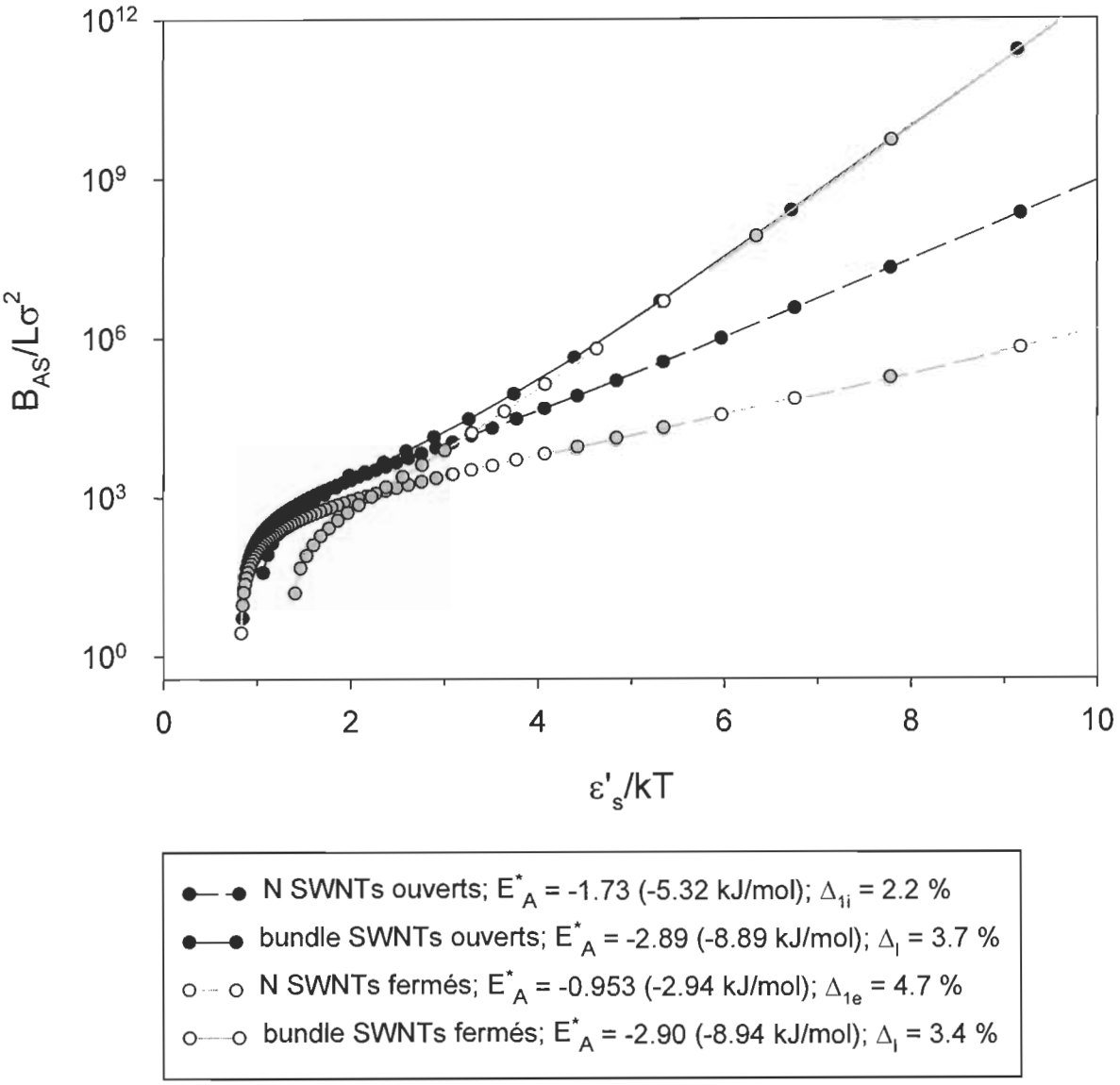


Figure 5.10. Comportement du B_{AS} en fonction de la température. $N_{tubes} = 25$, $d^* = 5.235$ (H_2 : 16.7 Å), $R^* = 2.038$ (H_2 : 6.5 Å).

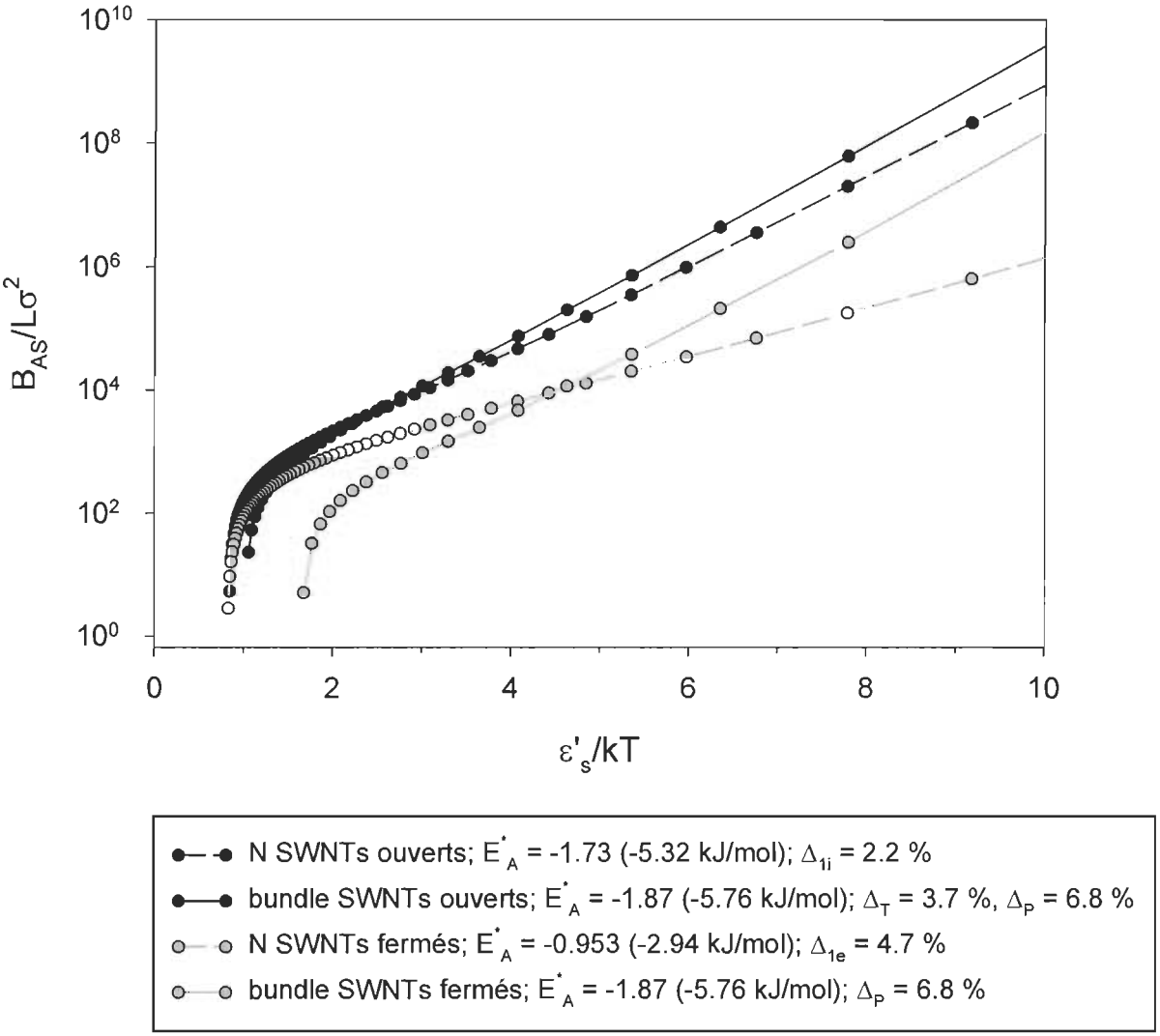


Figure 5.11. Comportement du B_{AS} en fonction de la température. $N_{tubes} = 25$, $d^* = 5.063$ (H_2 : 16.15 Å), $R^* = 2.038$ (H_2 : 6.5 Å).

5.1.4.4. Comportement du B_{AS} en fonction du nombre de nanotubes.

Afin de vérifier l'influence de la taille du faisceau sur le B_{AS} et de voir quelle sera l'ampleur des "effets de bords" lorsque le faisceau ne possède que quelques couches de nanotubes enroulées, nous avons étudié le comportement du B_{AS} renormalisé par le nombre de nanotubes (B_{AS} / N_{tubes}). Il est intéressant de constater que l'étendue du faisceau n'a que peu d'influences sur le comportement de B_{AS}. Puisque le comportement est similaire pour différents N_{tubes}, nous

avons là une bonne manière de vérifier le bon fonctionnement du programme décrit au chapitre précédent et donné à l'annexe 7.

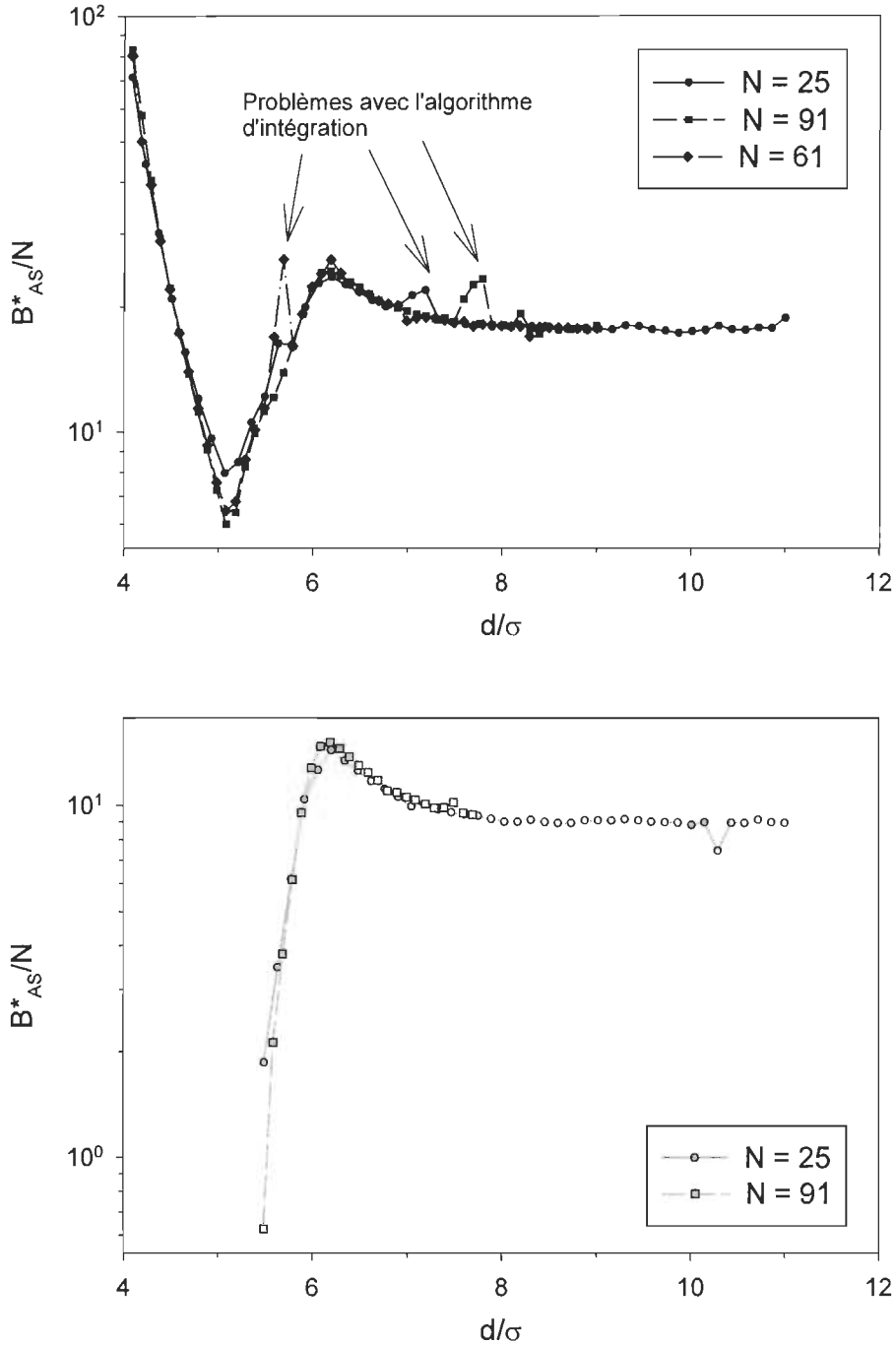


Figure 5.12. Influence du nombre de nanotubes sur le comportement du B_{AS} d'un faisceau en fonction du pas du réseau. Les courbes noires (grises) sont pour les nanotubes ouverts (fermés) aux bouts. $R^* = 2.038$ (H_2 : 6.5 Å), $T^* = 0.8097$ (H_2 : 300.0 K).

5.2. Interprétation du comportement du second coefficient du viriel.

Lors de cette section, nous allons expliquer physiquement l'origine des courbes obtenues à la section précédente. Nous avons vu aux sous-sections 5.1.2. et 5.1.3. que le B_{AS} devient maximal aux faibles températures lorsque la dimension du pore est telle que la profondeur du puits de potentiel à l'intérieur de celui-ci devient maximale. En effet, dans le cas du pore en fente, nous avons obtenu à l'aide du programme donné à l'annexe 5 :

Tableau 5.1 Largeur maximisant le B_{AS} d'un pore en fente en fonction de la température.

Température réduite (–)	**Largeur optimale réduite d^*_{max} (–)**
0.01	2.0018
0.1	2.018
0.2089	2.040
0.8097	2.198
1.15	2.318

Aux faibles températures, l'intégrand de B_{AS} $f = \exp\left(-\frac{V^*_{slit}(z^*, d^*)}{T^*}\right) - 1$ est fortement localisé autour des minimums de V^*_{slit}, de sorte que nous pouvons développer le potentiel d'adsorption réduit du pore en fente $V^*_{slit}(x^*, d^*)$ en série autour de x* = 0 lorsque le pore est étroit ($d < d_c$) :

$$V^*_{slit}(x^*, d^*) = -D^* + K^* x^{*2} \tag{5.1}$$

Où la profondeur du puits de potentiel et la constante de force (réduites) sont données par :

$$\begin{aligned} D^* &= D/\varepsilon_s' = -4\left(\frac{2048}{5\,d^{*10}} - \frac{16}{d^{*4}}\right) \\ K^* &= K\left(\sigma^2/\varepsilon_s'\right) = 4\left(\frac{90112}{d^{*12}} - \frac{640}{d^{*6}}\right) \end{aligned} \tag{5.2}$$

Alors (cf. Équ. (3.20)) :

$$B^*_{AS}(T^*, d^*) = \frac{B_{AS}}{A\sigma} = 2\, e^{D^*/T^*} \int_0^\infty e^{-\frac{K^*}{T^*}x^{*2}}\, dx^* - d^* \tag{5.3}$$

Où nous avons envoyé la borne d'intégration $d^*/2$ à l'infini puisque l'intégrand est une gaussienne fortement localisée autour de x* = 0. Utilisant l'intégrale

$\int_0^\infty e^{-\alpha u^2}\, du = \frac{1}{2}\sqrt{\frac{\pi}{\alpha}} \quad (\alpha > 0)$, nous obtenons :

$$B^*_{AS} = \sqrt{\pi}\, e^{D^*/T^*} \sqrt{\frac{T^*}{K^*}} - d^* \tag{5.4}$$

Et puisque le premier terme est très grand quand $T^* \to 0$, nous obtenons finalement une approximation de B^*_{AS} pour des pores étroits aux faibles températures :

$$B^*_{AS}\left(T^* \approx 0, d^* \approx 2\right) \approx \sqrt{\pi}\, e^{D^*/T^*} \sqrt{\frac{T^*}{K^*}} \quad ; \quad d^* < 2.281 \tag{5.5}$$

Puisque ln ( ) est une fonction monotoniquement croissante de son argument et puisque $B^*_{AS} > 0$ autour de d* = 2, nous pouvons trouver la largeur optimale du pore au moyen de la condition suivante :

$$0 = \frac{\partial}{\partial d^*} \ln B^*_{AS}(T^*, d^*) = -\frac{1}{2} K^{*-1} \frac{\partial K^*}{\partial d^*} + \frac{1}{T^*} \frac{\partial D^*}{\partial d^*} \tag{5.6}$$

Qui devient, après avoir substitué l'équation (5.2) :

$$0 = 256\left(-45056 + 1024\, d^{*6} - 5\, d^{*12}\right) + 3T^*\left(-1408\, d^{*10} + 5\, d^{*16}\right) \quad ; \quad d^* < 2.281,\ T^* \to 0 \tag{5.7}$$

Cette équation redonne bien les trois premières lignes du tableau 5.1. Elle montre également que $d^*_{\max B^*_{AS}}(T^* \to 0) = 2$ comme on s'y attendait. Ce résultat peut être généralisé pour toutes les nanostructures étudiées : le maximum de B_{AS} a lieu lorsque le puits du potentiel d'adsorption est le plus profond dans la limite $T \to 0$.

Il est possible de trouver une expression similaire à l'équation (5.5) dans le cas où $d^* > d^*_c$. Utilisant l'approximation harmonique du potentiel $V^*_{slit}(x^*, d^*)$ autour de son minimum $x^*_m > 0$:

$$B^*_{AS}\left(T^* \approx 0, d^* > d^*_c\right) \approx 2\sqrt{\pi}\; e^{D^*/T^*}\sqrt{\frac{T^*}{K^*}} \tag{5.8}$$

Une expression similaire existe également lorsque l'adsorption a lieu dans un SWNT [14] :

$$B^*_{AS}\left(T^* \approx 0, R^*\right) = \frac{B_{AS}}{L\sigma^2} \approx (2\pi)^{3/2} R^*\; e^{D^*/T^*}\sqrt{\frac{T^*}{K^*}} \tag{5.9}$$

Les équations (5.5), (5.8) et (5.9) ont toutes un comportement s'approchant de $B^*_{AS} \propto e^{D^*/T^*}$ quand $T^* \to 0$ (cf. Fig. 5.13) :

$$\lim_{T^*\to 0} \frac{\Delta_{droite}}{\left(\ln B^*_{AS}\right)_{droite}} = \lim_{\beta^*\to\infty} \frac{\left\{C^* - \frac{1}{2}\ln\beta^* + D^*\beta^*\right\} - \left\{C^* + D^*\beta^*\right\}}{C^* + D^*\beta^*} = \lim_{\beta^*\to\infty} \frac{-\frac{1}{2}\cdot\frac{1}{\beta^*}}{D^*} = 0 \tag{5.10}$$

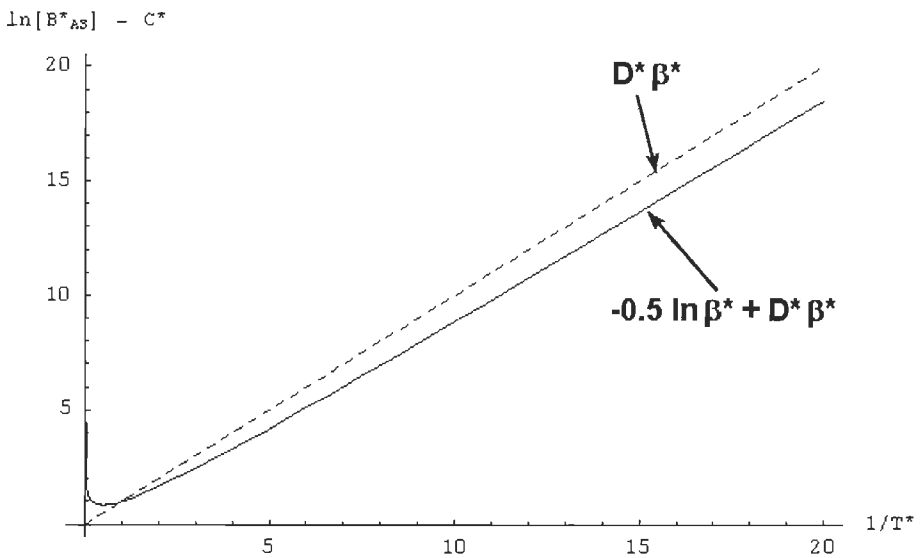


Figure 5.13. Comportement de ln B^*_{AS} quand $T^* \to 0$. Ici, D* = 1.

Nous pouvons donc conclure que "l'énergie d'activation" E_A, que nous avons définie par l'expression $B_{AS} = A\, e^{-E_A/kT}$, est égale à la valeur minimum −D du potentiel d'adsorption dans le pore.

Lorsqu'on regarde les courbes de B^*_{AS} en fonction de 1/T* montrées à la section précédente, on voit que peu importe la nanostructure, B^*_{AS} devient négatif aux hautes températures. Cela veut donc dire, en vertu de la définition de l'adsorption en excès (cf. passage

au dessus de Équ. (3.8)), qu'il y a moins d'adsorbat en présence de l'adsorbant qu'en son absence. En effet, si l'on trace l'intégrand $f = \exp\left(-\frac{V^*_{slit}(z^*, d^*)}{T^*}\right) - 1$ du B^*_{AS} d'un pore en fente pour des températures croissantes (cf. Fig. 5.14), on voit que lorsque la température augmente, les parties contribuant positivement au B^*_{AS} (où $V^*_{slit} < 0$) sont davantage atténuées que les parties contribuant négativement (où $V^*_{slit} > 0$). Lorsque la température devient élevée, l'adsorbat se comporte donc comme un gaz comprimé interagissant faiblement avec le pore, sauf pour l'interaction coeur dur qui l'empêche de s'approcher à moins d'une distance $\approx \sigma$ de la surface. Dans ce cas là, il est donc préférable, pour des applications de stockage, de retirer l'adsorbant du contenant car il ne fait alors qu'introduire un volume inaccessible à l'adsorbat.

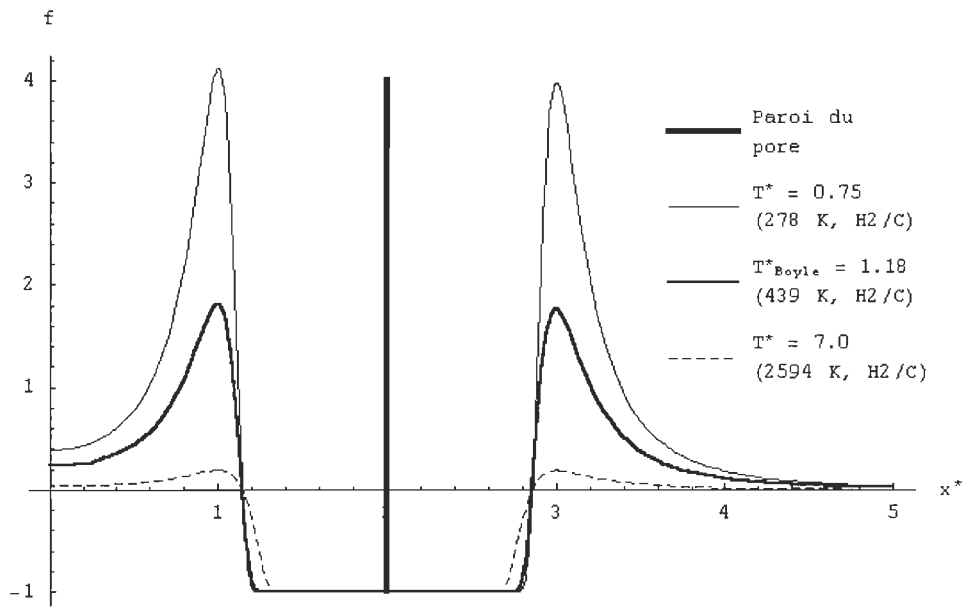


Figure 5.14. Comportement de l'intégrand f(d* = 4.0, T*, x*) du B^*_{AS} d'un pore en fente selon la température.

La température où B^*_{AS} devient négatif est appelée température de Boyle et sa dépendance sur la largeur du pore en fente est illustrée ci-dessous :

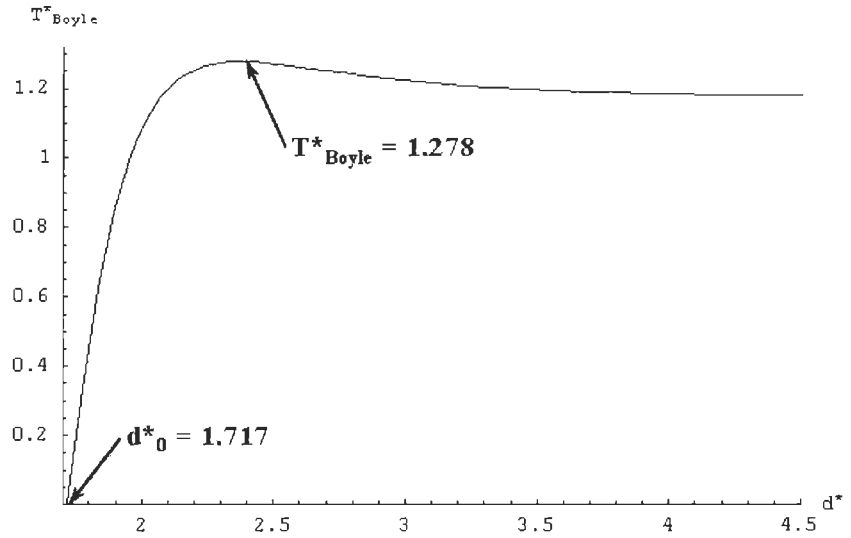


Figure 5.15. Température de Boyle d'un pore en fente en fonction de sa largeur.

Le comportement de la température de Boyle (pour laquelle $B^*_{AS} = 0$) en fonction du rayon d'un SWNT ouvert ou fermé est donné à la figure suivante :

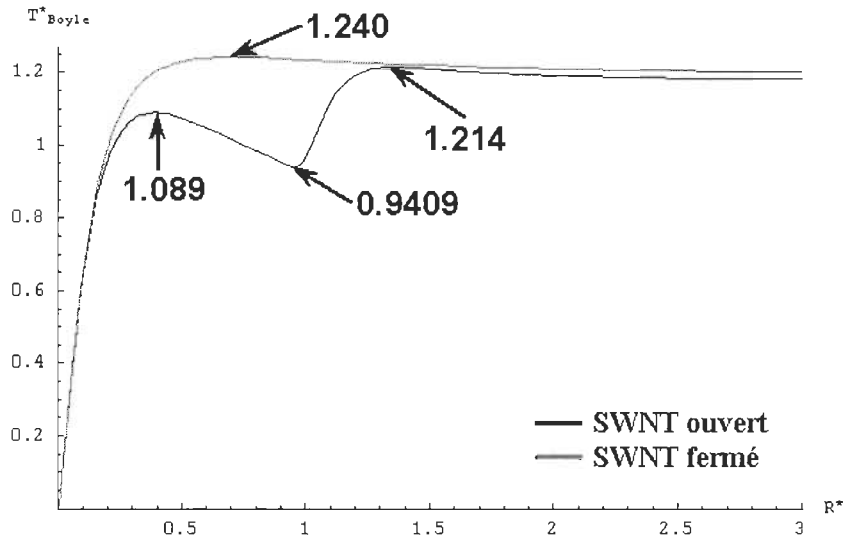


Figure 5.16. Température de Boyle d'un SWNT en fonction de son rayon.

Lorsqu'on regarde le pic situé à d* = 5.30 sur les graphiques de B^*_{AS} en fonction du pas du réseau (cf. Figs. 5.4 et 5.5), on voit que les courbes pour le faisceau de nanotubes ouverts et fermés coïncident exactement entre elles lorsque la température est très faible (H_2 : 1.85 K) mais que la coïncidence devient moins bonne à une température plus élevée (H_2 : 77.40 K). Nous verrons bientôt que ce pic est dû à la formation d'un minimum de potentiel au centre des

interstices situés entre les nanotubes. C'est donc dire que même lorsque les sites d'adsorption présents à l'intérieur des nanotubes contribuent à l'adsorption, seuls les sites d'adsorption interstitiels (plus attractifs) contribuent de manière significative à l'adsorption lorsque la température est faible. Le même phénomène est également visible au pic situé à R* = 2.00 sur les figures 5.7 et 5.8. Cela se vérifie aisément de la manière suivante :

On considère deux sortes de sites d'adsorption ayant des énergies d'interaction $-D_1$ et $-D_2$ (où $D_1 > D_2$) avec les molécules d'adsorbat. Nous noterons par N_1 et N_2 le nombre de molécules d'adsorbat (indiscernables) liées respectivement à ces deux sites (discernables) et nous supposerons que nous avons seulement deux sites d'adsorption pour simplifier. Alors, la fonction de partition de ce système sera donnée par :

$$Q_{N_1,N_2} = \frac{q_1^{N_1}}{N_1!}\frac{q_2^{N_2}}{N_2!} \tag{5.11}$$

Où $q_i = e^{D_i/kT}$ est la fonction de partition d'une molécule d'adsorbat liée au site i. Alors :

$$Q_{N_1,N_2} = \frac{e^{(N_1D_1+N_2D_2)/kT}}{N_1!\,N_2!} \tag{5.12}$$

Le potentiel chimique des molécules d'adsorbat liées au site 1 sera donné par [12] :

$$\mu_1 = \left(\frac{\partial A}{\partial N_1}\right)_{T,N_2} = -kT\left(\frac{\partial \ln Q_{N_1,N_2}}{\partial N_1}\right)_{T,N_2} \tag{5.13}$$

Qui devient, après avoir utilisé l'approximation de Stirling pour $\ln N_i!$ [12] :

$$\begin{aligned}\mu_1 = \left(\frac{\partial A}{\partial N_1}\right)_{T,N_2} &= -kT\,\frac{\partial}{\partial N_1}\left(\beta N_1D_1 + \beta N_2D_2 + N_1 + N_2 - N_1\ln N_1 - N_2\ln N_2\right)_{T,N_2} \\ &= -D_1 + kT\ln N_1\end{aligned} \tag{5.14}$$

Similairement, le potentiel chimique des molécules liées au site 2 est donné par :

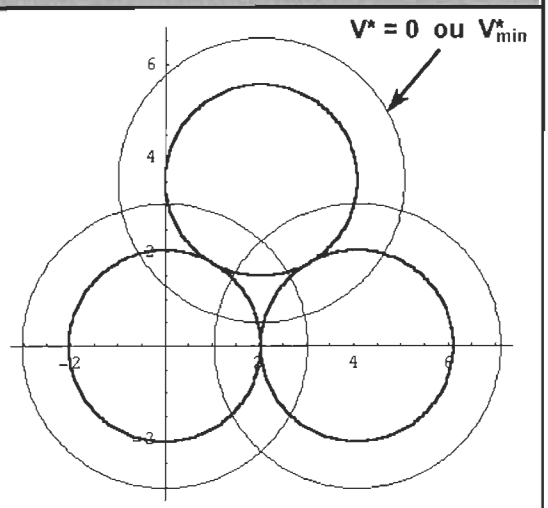
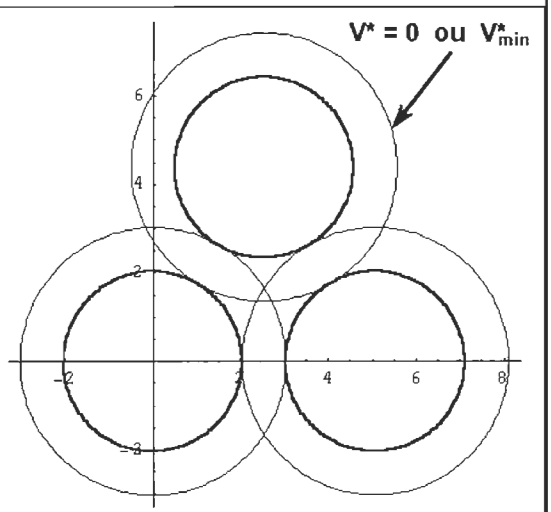
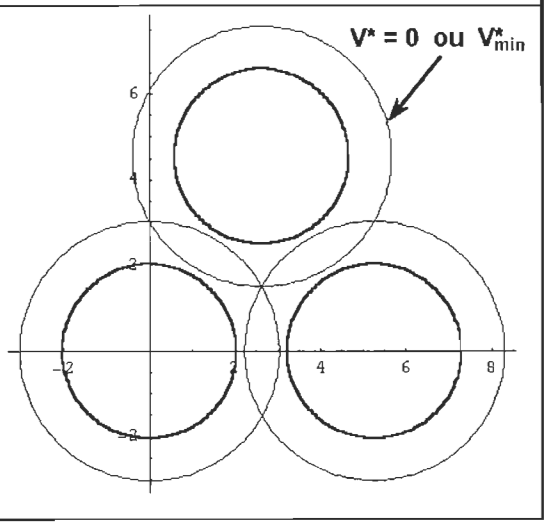
$$\mu_2 = -D_2 + kT\ln N_2 \tag{5.15}$$

À l'équilibre, ces deux potentiels chimiques devront être égaux, de sorte que nous obtenions finalement le rapport du nombre de molécules adsorbées sur les deux sites :

$$\frac{N_2}{N_1} = \exp\left(-\frac{D_1 - D_2}{kT}\right) \underset{T\to 0}{\to} 0 \qquad (5.16)$$

Ainsi donc, aux faibles températures, seulement les sites d'adsorption les plus attractifs contribuent de manière significative à l'adsorption.

Il est possible d'interpréter les principales caractéristiques des courbes de B^*_{AS} en fonction de d* et de B^*_{AS} en fonction de R* (Figs. 5.4 et 5.7) en terme du potentiel d'adsorption du faisceau et des nanotubes, lorsque la température est faible. Rappelons qu'au chapitre 2, nous avons obtenu que le potentiel devient minimum à l'intérieur du nanotube isolé lorsque $R^*_{\min} = 1.086$ et qu'il devient nul lorsque $R^*_0 = 0.9322$. La figure suivante montre le comportement des équipotentielles individuelles minimales et nulles à l'extérieur de trois nanotubes regroupés en faisceau lorsqu'on fait varier d* ou R* (l'autre dimension étant gardée fixe).

R* = 2.038	**d* = 5.235**	
$d^* = 4.076$	$R^* = 2.618$	
$d^*_{\min} = 5.071$ $d^*_0 = 4.933$	$R^*_{\min} = 2.120$ $R^*_0 = 2.189$	
$d^*_{\min} = 5.253$ $d^*_0 = 5.014$	$R^*_{\min} = 2.027$ $R^*_0 = 2.165$	

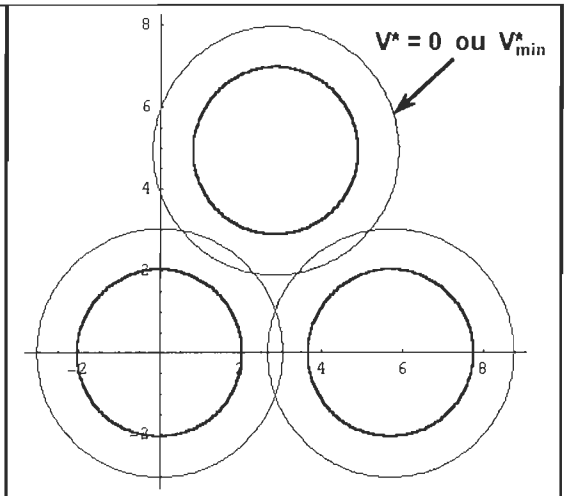
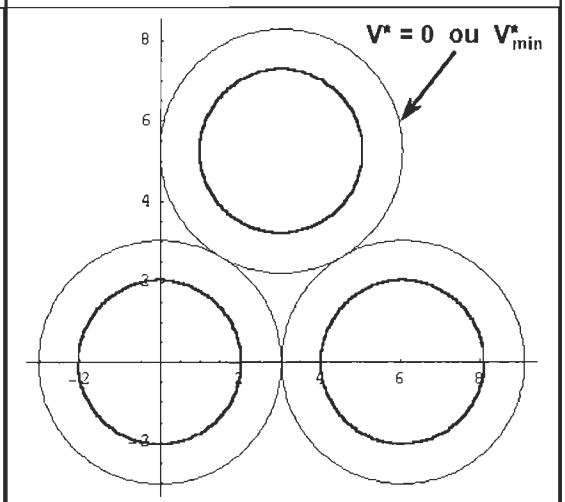
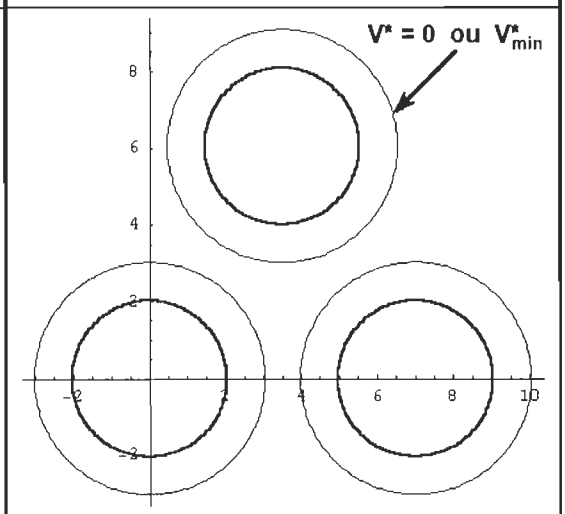
$5.253 < d^*_{min} < 6.066$ $5.014 < d^*_0 < 5.790$	$2.027 > R^*_{min} > 1.623$ $2.165 > R^*_0 > 1.760$	
$d^*_{min} = 6.066$ $d^*_0 = 5.790$	$R^*_{min} = 1.623$ $R^*_0 = 1.760$	
$d^* \to \infty$	$R^* \to 0$	

Figure 5.17. Comportement des équipotentielles minimales et nulles des nanotubes d'un faisceau lorsque ses dimensions réduites d* et R* varient.

La figure 5.4 montre le comportement du B^*_{AS} en fonction de la distance séparant les centres des nanotubes formant le faisceau. Le B^*_{AS} du faisceau de nanotubes ouverts augmente près de l'origine puisque les puits de potentiels extérieurs des nanotubes voisins creusent le potentiel à l'intérieur des nanotubes. Le pic à $d^* = 5.27$ est dû à la fusion des minimums de potentiel au centre des interstices entre les tubes. Lorsque les tubes continuent de s'éloigner les uns des autres, les minimums secondaires alors présents dans les interstices finissent par disparaître à $d^* = 6.11$ créant ainsi un coude dans la courbe du B^*_{AS}. Pour des distances encore plus grandes, le B^*_{AS} fini par se stabiliser à sa valeur limite pour 25 nanotubes indépendants. Cette valeur est plus grande pour les tubes ouverts que pour les tubes fermés puisque les sites d'adsorption à l'intérieur des tubes sont plus attractifs que ceux situés à l'extérieur des tubes (cf. tableau 2.2).

La figure 5.7 compare le B^*_{AS} des nanotubes indépendants (courbes en tirets) avec celui des faisceaux (courbes continues). Lorsque les nanotubes sont trop étroits ($R^* < 0.94$) l'intérieur des tubes est répulsif et seulement les surfaces extérieures contribuent à l'adsorption. Le pic à $R^* = 1.08$ se produit lorsque le potentiel dans les nanotubes ouverts atteint sa profondeur maximale. Le deuxième pic à $R^* = 2.03$ est dû à la fusion des minimums de potentiel à l'intérieur des interstices et l'augmentation ($R^* > 2.13$) se produisant pour le faisceau de tubes ouverts est causée par le fait que les nanotubes voisins creusent le potentiel à l'intérieur d'un nanotube donné. Le coude présent à $R^* = 1.61$ se forme lorsque le rayon est assez grand pour que des minimums secondaires puissent se former dans les interstices.

5.3. Identification des sites d'adsorption dominants d'un faisceau de nanotubes.

Nous avons vu à la section précédente que l'énergie d'activation E_A est égale au minimum du potentiel d'adsorption dans les pores. Il suffira donc de comparer l'énergie d'activation avec les minimums de potentiel (donnés au tableau 2.2) dans les différents sites d'adsorption d'un faisceau pour trouver quelle sorte de site contribue davantage à l'adsorption (e.g. Figs. 5.10 et 5.11). Le tableau suivant montre les sites d'adsorption dominants pour différents faisceaux de SWNTs :

Tableau 5.2 Sites d'adsorption dominants d'un nanotube et d'un faisceau, pour l'hydrogène à de faibles températures.

d (Å) (d*)	**Nanostructure**	**R = 6.501 Å (R* = 2.038)**	**R_{min} = 3.464 Å (R_{min}* = 1.086)**
	SWNT ouvert	Intérieur tube	Intérieur tube
	SWNT fermé	Extérieur tube	Extérieur tube
16.70 (5.235)	Faisceau o-SWNTs	Interstices	
	Faisceau c-SWNTs	Interstices	
16.15 (5.063)	Faisceau o-SWNTs	Intérieur tubes Sites périphériques	
	Faisceau c-SWNTs	Sites périphériques	
10.63 (3.332)	Faisceau o-SWNTs		Intérieur tubes
	Faisceau c-SWNTs		Sites périphériques
10.08 (3.159)	Faisceau o-SWNTs		Intérieur tubes
	Faisceau c-SWNTs		Sites périphériques

Du tableau 2.2, on voit que l'énergie d'activation pour un faisceau de nanotubes de rayon R_{min} = 3.464 Å espacés de 3.15 Å est de −13.78 kJ/mol. Ce résultat est de 30 % inférieur à la très forte énergie d'activation de 19.6 kJ/mol trouvée par Dillon et al. [4] pour des nanotubes ayant un rayon non optimal de 6 Å (i.e. > 3.464 Å). L'écart est sans doute dû à la petitesse des échantillons (≈ 1 mg) utilisés dans ces expériences et à leur très grande impureté (les nanotubes représentaient seulement 0.1 à 0.2 % du poids total).

5.4. Calcul des isothermes d'adsorption sur des nanotubes et des faisceaux, pour des faibles pressions, au moyen du B_{AS}.

Il est possible de calculer les isothermes d'adsorption en excès sur des nanotubes et des faisceaux pour de très faibles pressions, au moyen des B_{AS} obtenus avec les programmes donnés aux annexes 6 et 7. Les nanotubes considérés ici ont un rayon de 6.5 Å et leurs centres sont espacés de 16.7 Å. Nous avons choisi un gros faisceau de 91 tubes enroulés sur 5 couches concentriques afin de minimiser les effets de bords. Les isothermes d'adsorption en excès de l'hydrogène ont ainsi été calculés à 77.4 et 300 K. Afin de pouvoir comparer nos résultats avec l'objectif du DOE (6.5 wt%, [3, 4]), nous avons exprimé ces isothermes en pourcentage de la masse totale du système.

$$N_a = B_{AS}\left(\frac{p}{kT}\right) = \frac{B^*_{AS} L \sigma^2}{kT} p \qquad [\text{molécules } H_2] \tag{5.17}$$

Nous noterons par M_C et M_{H_2} la masse molaire du carbone et de l'hydrogène moléculaire respectivement. Puisque N_{tubes} nanotubes à parois simples regroupés en faisceau possèdent $2\pi N_{tubes} R L \theta$ atomes de carbone, nous avons :

$$\begin{aligned} N_a &= \frac{B^*_{AS}\sigma}{2\pi N_{tubes}\theta R^*} \cdot \frac{p}{kT} \qquad [\text{mol } H_2/\text{mol C}] \\ &= 101325 \frac{M_{H_2}}{M_C} \cdot \frac{B^*_{AS}\sigma}{2\pi N_{tubes}\theta R^*} \cdot \frac{P}{kT} \qquad [\text{g } H_2/\text{g C}] \end{aligned} \tag{5.18}$$

Où la pression P est exprimée en atmosphères. Alors, l'adsorption en excès exprimée en pourcentage de la masse totale est donnée par :

$$\%\ \text{masse} = 100\left(\frac{\text{masse } H_2}{\text{masse } H_2 + \text{masse C}}\right) = 100\left(\frac{N_a}{1+N_a}\right) = 100\left[1 + N_a^{-1}\right]^{-1} \tag{5.19}$$

Les isothermes d'adsorption ainsi obtenus sont donnés à la figure ci-dessous pour 77.4 et 300 K :

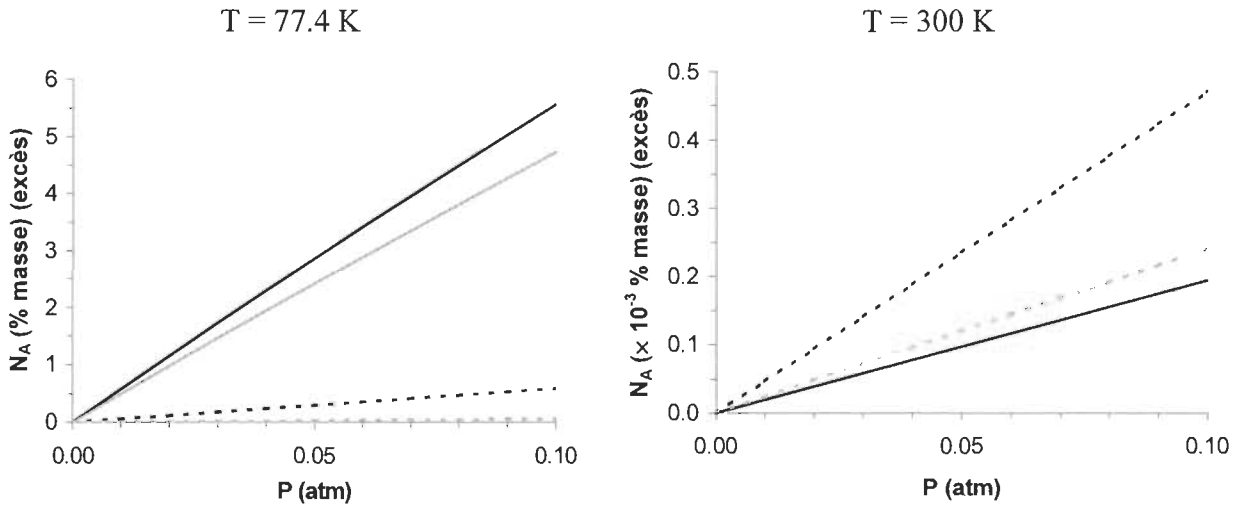


Figure 5.18. Isothermes d'adsorption en excès d'hydrogène sur un nanotube de rayon 6.5 Å et sur un faisceau de 91 tubes espacés de 3.7 Å. Les courbes continues (en tirets) sont pour le faisceau (nanotube isolé) et celles en noir (gris) sont pour des nanotubes ouverts (fermés) aux bouts.

Sur la figure précédente, l'isotherme du faisceau de c-SWNTs à 300 K n'a pas pu être tracé puisque le B^*_{AS} correspondant est négatif. Les valeurs de B^*_{AS} utilisées pour tracer ces isothermes ont été vérifiées indépendamment en intégrant simplement l'équation (3.33) au moyen de la méthode Monte Carlo implémentée dans Mathematica (bien que cela soit beaucoup plus lent).

5.5. Application expérimentale.

En mesurant plusieurs isothermes d'adsorption d'hydrogène sur le charbon activé AX-21 sur une vaste plage de températures, il a été possible de déterminer la surface spécifique de ce charbon de même que la largeur moyenne de ses pores au moyen du modèle du pore en fente. Premièrement, nous avons extrait les B_{AS} expérimentaux en faisant une régression polynomiale sur les isothermes d'adsorption aux faibles pressions. L'équation (3.9) a été utilisée sous la forme :

$$N_a/p = A + Bp + Cp^2 + Dp^3 + Ep^4 \quad \text{où} \quad B_{AS} = A\,kT \tag{5.20}$$

La figure ci-dessous illustre la détermination du B_{AS} à partir de l'isotherme à 113 K :

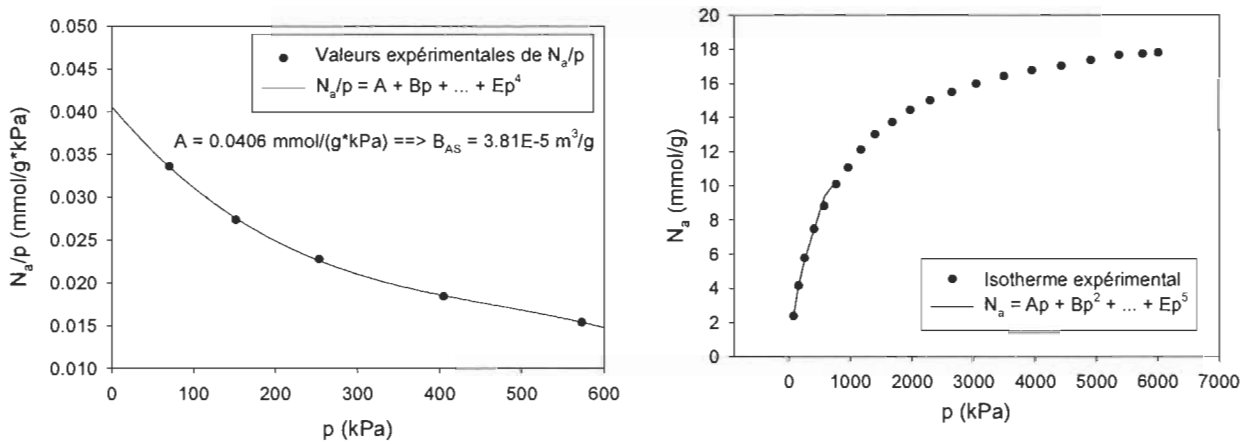


Figure 5.19. Détermination du B_{AS} à partir de l'isotherme d'adsorption d'hydrogène à 113 K sur le charbon activé AX-21.

Après avoir obtenu les B_{AS} expérimentaux des différents isothermes d'adsorption, il suffit de les ajuster aux courbes de B_{AS} théoriques (calculées avec l'équation (3.20) pour plusieurs largeurs du pore) pour obtenir la largeur moyenne d du pore représentant le charbon et sa surface spécifique.

Cet ajustement est montré ci-dessous et est fait sur les paramètres Aσ et d* (A étant l'aire d'une des parois du pore).

Comparaison des B_{AS} calculés selon le modèle "slit pore" avec les B_{AS} expérimentaux et détermination de la distance caractéristique des 2 plans et de la surface spécifique pour l'adsorption d'H_2 sur le charbon activé AX-21.

H_2:

			Calculs avec la valeur théorique de ε_s'				Courbe théorique 1 plan (d = infini)		Courbe théorique 2 plans (d/σ = 2)	Courbe théorique 2 plans (d/σ = ...)
T (K)	A (mmol/g*kPa)	B_{AS} (m^3/g)	ε_s'/k (K)	Aσ (m^3/g)	ε_s'/kT (-)	B_{AS}/Aσ (-)	ε_s'/kT (-)	B_{AS}/Aσ (-)	B_{AS}/Aσ (-)	B_{AS}/Aσ (-)
77	0.7026	0.000449815	370.52	0.00000047	4.811948052	957.0523364	14.2857143	5586600	5.15E+13	.....
93	0.1899	0.000146839			3.984086022	312.4242461	11.5622576	239414	8.26E+10	.....
113	0.0406	3.81451E-05	σ (m)	A (m^2/g)	3.278938053	81.15986094	9.71090632	28707.8	1.05E+09	.....
133	0.01515	1.67532E-05	3.19E-10	**1.5E+03**	2.785864662	35.64520289	8.37065249	6279.15	4.53E+07	.....
153	0.008725	1.10992E-05			2.421699346	23.61531178	7.35548314	2010.51	4.21E+06	.....
173	0.004126	5.93485E-06	meilleur Aσ (m^3/g) pour qques d/σ		2.141734104	12.62734784	6.55991498	831.719	658433	.....
193	0.002722	4.36797E-06			1.919792746	9.293562994	5.91961167	411.882	148504	
213	0.001608	2.84774E-06			1.739530516	6.059021102	5.39321964	232.425	43813.3	
233	0.001468	2.84392E-06	2.5	0.00000045	1.590214592	6.050883534	4.95279982	144.565	15826.3	
253	0.000961	2.02152E-06	2.4	pas bon fit	1.464505929	4.301111746	4.57885941	96.8354	6883.67	
273	0.0007468	1.69512E-06	2.6	0.00000055	1.357216117	3.606648126	4.25743881	68.7026	3192.22	
298	0.000541	1.34044E-06	**2.55**	**0.00000047**	1.243355705	2.852005419	3.97818364	51.002	1682.6	
			1.81	pas bon fit			3.73329351	39.2578	960.845	
			1.80	pas bon fit			3.51681742	31.1164	586.061	
							3.32407026	25.2621	377.593	
			Paramètres caractéristiques du charbon:				3.15135319	20.9201	254.724	
			d (Angstroms) =	8.1			2.9956892	17.6136	178.649	
			$A_{spécitique}$ (m^2/g) =	2.9E+03			2.85468783	15.0379	129.514	
							2.72636311	12.9914	96.5974	
							2.60907227	11.3374	73.8313	
							2.50146336	9.98012	57.6396	
							2.40237932	8.8514	45.8363	
Conclusion:							2.31084572	7.90154	37.0418	
Il est difficile d'obtenir avec le fit des valeurs précises. Pour la distance caractéristique des							2.22602625	7.09367	30.36	
2 plans modélisant l'adsorption d'H_2 sur le charbon activé AX-21, on a:							2.14721753	6.4	25.194	
d = 8.1 ± 0.2 Angstroms							2.07379818	5.79926	21.1366	
Et pour la surface spécifique du charbon activé, on a:							2.00522964	5.27495	17.9045	
A = 2.9E3 m^2/g. Où on a compté les 2 faces des plans opposés servant à modéliser le charbon activé.							1.94105407	4.81411	15.2963	

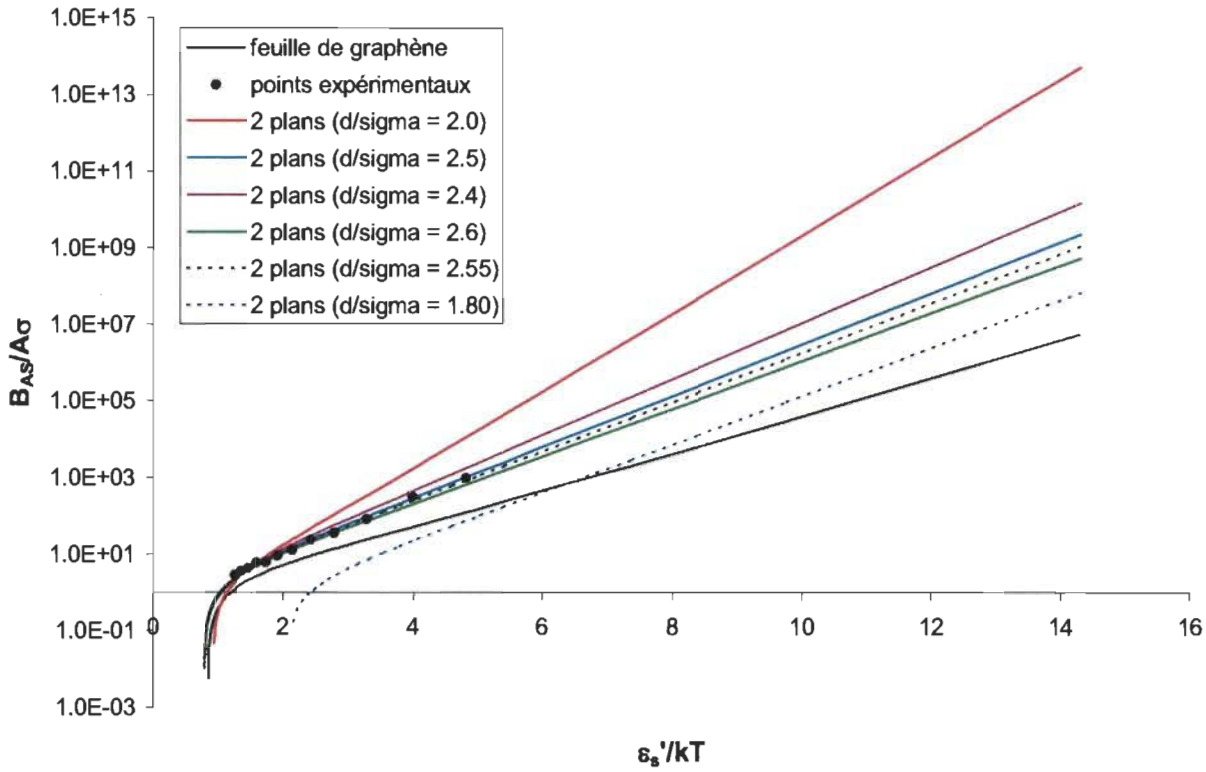


Figure 5.20. Détermination de la surface spécifique et de la largeur moyenne des pores du charbon activé AX-21.

De ces calculs, nous obtenons une surface spécifique de 2900 m^2/g et une largeur moyenne de 8.1 ± 0.2 Å. Cette surface spécifique est en bon accord avec la surface de BET, valant 3000 m^2/g, obtenue à partir d'un isotherme d'adsorption d'azote. La largeur moyenne est de 17 % inférieure à celle obtenue (12 ± 2 Å) du modèle de la densité fonctionnelle avec le logiciel Quantachrome Autosorb.

En procédant de la même manière pour des isothermes d'adsorption de méthane sur le charbon activé CNS-201, nous avons obtenu une largeur moyenne des pores en fente de 7.2 ± 0.4 Å, ce qui est une valeur sensée (près de la largeur 7.7 Å des ultramicropores donnée au chapitre 1). Notre technique pourra également être employée sur un échantillon contenant des faisceaux de SWNTs puisque là aussi, seulement deux paramètres indépendants (R* et $L\sigma^2$) auront besoin d'êtres trouvés ($d = 2R + 3.15$ Å n'est pas indépendant). La surface spécifique pourra alors être calculée simplement de l'équation $A_{sp} = 2\pi N_{tubes} RL$, où N_{tubes} est assez grand (≈ 61).

CONCLUSION

Nous avons présenté une étude numérique de la physisorption d'hydrogène sur différentes nanostructures de carbone dans la limite des faibles pressions au moyen d'une expansion en série du viriel de la quantité adsorbée en excès N_a. Aux faibles températures et aux très faibles pressions, nous avons trouvé que le potentiel d'adsorption joue un rôle déterminant sur le comportement de la physisorption.

Aux températures d'intérêt ($\geq$ 77.4 K), nous avons montré que l'utilisation de nanotubes ayant des parois continues est tout à fait légitime lorsqu'on désire calculer le potentiel d'adsorption (la corrugation du potentiel étant alors négligeable par rapport à l'énergie thermique des molécules d'adsorbat). De plus, pour ces températures, une étude du potentiel d'adsorption effectif V_{eff} développé par R. P. Feynman [38] a montré que les effets quantiques (favorisés par des adsorbats légers et des faibles températures) sont négligeables. Nous avons vu que les nanotubes et les faisceaux ont une forte énergie d'interaction avec l'adsorbat, ce qui favorise la physisorption :

$$\frac{V_{min,\,slit}}{V_{min,\,flat}} = 2,\ \frac{V_{min,\,SWNT}}{V_{min,\,flat}} = 3.39 \text{ et } \frac{V_{min,\,bundle}}{V_{min,\,flat}} = 3.73.$$

Dans la limite des faibles pressions, l'adsorption en excès est proportionnelle à la pression ($N_a = B_{AS}\ p/kT$). Une étude du second coefficient du viriel B_{AS} en fonction des dimensions de la nanostructure (d et/ou R) et de la température a montré sa relation avec le potentiel d'adsorption. Notamment, nous avons montré qu'aux faibles températures B_{AS} est maximal lorsque le puits du potentiel d'adsorption devient le plus creux. De plus, lorsque plusieurs sites d'adsorption sont présents, seul celui ayant la plus forte énergie d'interaction D contribue de manière significative à l'adsorption totale. Aux hautes températures (où $B_{AS} < 0$), il est préférable de retirer l'adsorbant du contenant car l'adsorbat se comporte alors comme un gaz n'ayant aucune interaction avec la surface, sauf une interaction coeur dur qui a pour effet d'introduire un volume inaccessible aux molécules d'adsorbat. Après avoir associé l'énergie d'activation E_A au minimum $-D$ du potentiel d'adsorption dans les différents sites d'adsorption d'un faisceau de nanotubes, il a été possible de trouver quels sites contribuent majoritairement à l'adsorption sur le faisceau. Notamment, pour un faisceau typique ayant des tubes de 6.5 Å de

rayon espacés de 3.15 Å, nous avons obtenu que l'intérieur des tubes et les sites situés en périphérie du faisceau dominent l'adsorption à faible pression. L'énergie d'activation maximale possible (pour des tubes de 3.46 Å de rayon espacés de 3.15 Å) est de −13.78 kJ/mol, soit 30 % inférieure à celle trouvée par Dillon et al. (19.6 kJ/mol) [4] pour des nanotubes ayant un rayon non optimal de 6 Å (i.e. > 3.46 Å). L'écart semble attribuable à la petitesse et à la très grande impureté de leurs échantillons. Après avoir obtenu les B_{AS} expérimentaux à partir d'une régression polynomiale sur les isothermes d'adsorption d'hydrogène sur le charbon activé AX-21 aux faibles pressions, il a été possible d'obtenir sa surface spécifique et la taille moyenne de ses pores en ajustant ces B_{AS} aux courbes théoriques de B_{AS} obtenues avec le programme donné à l'annexe 5 pour le pore en fente. Les valeurs ainsi trouvées (2900 m^2/g et 8.1 ± 0.2 Å) sont en assez bon accord avec la surface de BET et la largeur donnée par le modèle de la densité fonctionnelle (3000 m^2/g et 12 ± 2 Å). Notre méthode pourra également s'appliquer au faisceau de nanotubes puisque là aussi, seulement deux paramètres indépendants (L et R) devront être trouvés.

Notre approche possède cependant quelques défauts dont nous allons maintenant discuter brièvement. Premièrement, le fait de ne retenir que le coefficient B_{AS} dans les calculs revient à négliger les interactions (à deux corps, à trois corps, etc.) présentes entre les molécules d'adsorbat aux pressions plus élevées. C'est donc dire que puisque les molécules d'adsorbat ne se voient pas entre elles, aucune saturation de l'isotherme ne pourrait avoir lieu lorsque les pores deviennent complètement remplis d'adsorbat. Malgré le fait que l'étude du potentiel effectif indique que les effets quantiques devraient être négligeables aux températures d'intérêt expérimental, nous n'avons pas pu inclure les effets quantiques dans le calcul des B_{AS} (à cause d'un problème de divergence causé par la définition même du potentiel effectif). Le potentiel de Lennard-Jones 12−6 que nous avons utilisé pour calculer l'interaction entre un atome de carbone et une molécule d'adsorbat suppose que la molécule d'adsorbat est sphérique. Or ceci n'est pas le cas pour la molécule d'hydrogène. Cependant, des simulations Monte Carlo grand canonique classiques [33] pour l'adsorption d'hydrogène dans des nanofibres de carbone (ayant des pores en fente) ont montré que l'utilisation d'une molécule allongée au lieu d'une molécule sphérique ne fait que peu de différences sur les résultats observés. Étant donné que les très petits pores (ultramicropores de largeur d ≤ 7 Å) ont une forte énergie d'interaction avec l'adsorbat, ceux-ci vont contribuer

davantage à la largeur moyenne trouvée ici (8.1 ± 0.2 Å). Il faudrait donc inclure la distribution de la taille des pores f(H) dans les calculs lorsqu'on travaille avec les charbons activés [20, 23].

Il existe des théories plus sophistiquées que celle employée ici et qui permettent toutes d'inclure les interactions entre les molécules d'adsorbat. Mentionnons notamment les simulations Monte Carlo grand canonique classique [15, 16] et quantique [17, 18], la théorie de la densité fonctionnelle [20-22] (qui permet de trouver f(H)) et la simulation avec dynamique moléculaire [19].

RÉFÉRENCES

1. www.oilcrisis.org; www.dieoff.org

2. C. J. Campbell, J. H. Laherrère, "The End of Cheap Oil", *Scientific American*, 78-83, March 1998.

3. A. C. Dillon, K. E. H. Gilbert et al., "Carbon Nanotube Materials For Hydrogen Storage", *Proceedings of the 2001 U.S. DOE Hydrogen Program Review*. Disponible sur le web: http://www.eren.doe.gov/hydrogen/publications.html

4. A. C. Dillon, K. M. Jones et al., *Nature*, **386**, 377 (1997).

5. B. Simard, S. Dénommée et al., "Recent Advances In Carbon Nanotubes Technologies At The National Research Council", *11th CHA Conference Proceedings*, Victoria, B.C., (2001).

6. Y. Ye, C. C. Ahn, C. Witham et al., *Appl. Phys. Lett.*, **74**, 2307 (1999).

7. C. Liu, Y. Y. Fan, M. Liu et al., *Science*, **286**, 1127 (1999).

8. C. Nutzenadel, A. Zuttel, D. Chartouni et al., *Electrochem. Solid-State Lett.*, **2**, 30 (1999).

9. H. M. Cheng, C. Liu, Y. Y. Fan et al., *Z. Metallkd.*, **91**, 306 (2000).

10. N. Rajalakshmi, K. S. Dhathathreyan, A. Govindaraj et al., *Electrochimica Acta*, **45**, 4511 (2000).

11. W. A. Steele, *The interactions of Gases with Solid Surfaces*, Pergamon, Oxford, (1974).

12. T. L. Hill, *An Introduction to Statistical Thermodynamics*, Dover, New York, (1986).

13. E. A. Flood, *The Solid-Gas Interface (vol. 1)*, Marcel Dekker, New York, (1967).

14. G. Stan, M. W. Cole, *Surf. Sci.*, **395**, 280 (1998).

15. M. Rzepka, P. Lamp, M. A. de la Casa-Lillo, *J. Phys. Chem. B*, **102**, 10894 (1998).

16. F. Darkrim, D. Levesque, *J. Chem. Phys*, **109**, 4981 (1998).

17. Q. Wang, J. K. Johnson, *Molecular Physics*, **95**, 299 (1998).

18. Q. Wang, J. K. Johnson, *J. Chem. Phys.*, **110**, 577 (1999).

19. S. Maruyama, T. Kimura, "Molecular Dynamics Simulation of Hydrogen Storage in Single-Walled Carbon Nanotubes", *Proc. ASME Heat Transfer Division 2000*, Orlando, **2**, 405 (2000). Disponible sur le web: http://www.photon.t.u-tokyo.ac.jp/~maruyama/papers/00/tube_asme.pdf

20. C. Lastoskie, K.E. Gubbins, N. Quirke, *J. Phys. Chem.*, **97**, 4786 (1993).

21. A. V. Neimark, *Langmuir*, **11**, 4183 (1995).

22. C. Lastoskie, K.E. Gubbins, N. Quirke, *Langmuir*, **9**, 2693 (1993).

23. D. H. Everett, J. C. Powl, *Journal of the Chemical Society. Faraday Transactions I*, **72**, 619 (1976).

24. A. Thess, R. Lee, P. Nikolaev et al., *Science*, **273**, 483 (1996).

25. J. W. G. Wildöer, L. C. Venema, A. G. Rinzler et al., *Nature*, **391**, 59 (1998).

26. A. Kuznetsova, D. B. Mawhinney, V. Naumenko et al., *Chem. Phys. Lett.*, **321**, 292 (2000).

27. A. G. Rinzler, J. Liu, H. Dai et al., *Appl. Phys. A*, **67**, 29 (1998).

28. W. E. Carlos, M. W. Cole, *Surface Science*, **91**, 339 (1980).

29. J. O. Hirschfelder, C. F. Curtiss, R. B. Bird, *Molecular Theory of Gases and Liquids*, Wiley, New York, (1954).

30. M. J. Bojan, R. van Slooten, W. A. Steele, *Separation Science and Technology*, **27**, 1837 (1992).

31. G. Stan, M. J. Bojan, S. Curtarolo et al., *Phys. Rev. B*, **62**, 2173 (2000).

32. A. Cheng, M. L. Klein, C. Caccamo, *Phys. Rev. Lett.*, **71**, 1200 (1993).

33. R. F. Cracknell, *Phys. Chem. Chem. Phys.*, **3**, 2091 (2001).

34. D. Cao, W. Wang, *Phys. Chem. Chem. Phys.*, **3**, 3150 (2001).

35. R. C. Reid, J. M. Prausnitz, B. E. Poling, *The properties of gases and liquids*, 4th edition, McGraw-Hill, New York, (1987).

36. G. Stan, M. W. Cole, *J. Low Temp. Phys.*, **110**, 539 (1998).

37. M. Brandbyge, "*Atomic and Electronic Structure of Carbon Nanotubes*", Mikroelectronik Centret (MIC), Danish Technical University.
Notebook Mathematica disponible sur le web:
http://library.wolfram.com/infocenter/MathSource/384/

38. R. P. Feynman, *Statistical Mechanics, A Set of Lectures*, Addison-Wesley, Reading, (1998).

39. L. Marchildon, *Mécanique Quantique*, 1ère éd., De Boeck Université, Bruxelles, (2000).

40. G. B. Arfken, H. J. Weber, *Mathematical methods for physicists*, 4ème éd., Academic Press, San Diego, (1995).

41. T.L. Hill, *Statistical Mechanics*, Dover, New York, (1987).

42. W. H. Press, S. A. Teukolsky, W. T. Vetterling et al., *Numerical Recipes in C, the art of scientific computing*, 2ème éd., Cambridge University Press, Cambridge, (1992).

43. B. M. Ayyub, R. H. McCuen, *Numerical methods for engineers*, Prentice-Hall, Upper Saddle River, (1996).

44. K. Murata, K. Kaneko, W. A. Steele et al., *J. Phys. Chem. B.*, **105**, 10210 (2001).

45. E. Kreysig, *Advanced Engineering Mathematics*, 7ème éd., John Wiley & Sons, New York, (1993).

ANNEXE 1. QUELQUES CARACTÉRISTIQUES DES FULLÉRÈNES.

Nous avons vu, au chapitre 1, que les nanotubes de carbone et les buckyballs appartiennent à une famille de molécules convexes, refermées sur elles-mêmes et ne comportant que des faces hexagonales et pentagonales. Le but de cet annexe est d'obtenir le résultat intéressant que toutes les fullérènes ont nécessairement 12 faces pentagonales et que la plus petite fullérène permise a seulement 20 atomes de carbone arrangés en une balle de 12 pentagones. Mais avant de parvenir à ces conclusions, nous avons besoin d'un théorème :

A1.1. Le théorème d'Euler sur les polyèdres.

Soit un polyèdre quelconque, un solide qui est limité dans l'espace par des faces polygonales. Notons par c son nombre de côtés, par s son nombre de sommets et par f son nombre de faces. Alors le théorème d'Euler dit que $s - c + f = 2$. (Par exemple, pour un cube : $s = 8$; $c = 12$; $f = 6$; $s - c + f = 2$.)

Il est commode de définir deux quantités $D = s - c + f$ et $D' = s - c + F$, où $F = f - 1$. Si nous prenons notre polyèdre et que nous lui retirons une face, nous obtenons une coquille creuse formée de toutes les faces restantes. Nous avons alors que s et c sont inchangés mais que le nombre de faces devient F. Nous allons donc travailler avec la quantité D' lors de cette démonstration. Si l'on aplati de manière continue la coquille sur un plan, nous obtenons une figure polygonale possédant les mêmes s, c, F et D'. (e.g. Fig. A1.1 a pour un cube).

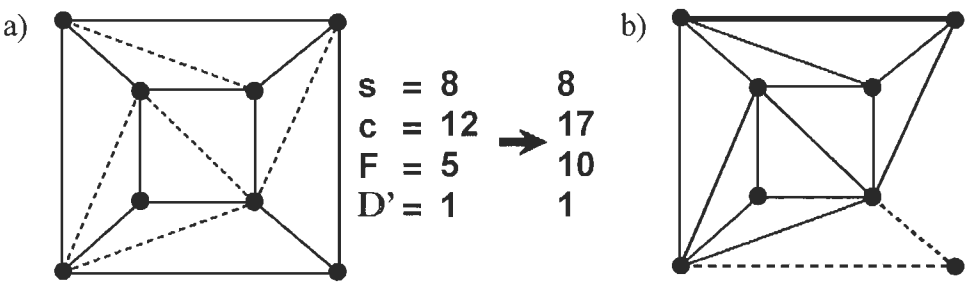


Figure A1.1. Démonstration du théorème d'Euler sur un cube.

Maintenant, si l'on divise toutes les faces polygonales de la figure précédente en triangles au moyen de segments pointillés (cf. Fig. A1.1 a) reliant des sommets déjà existants, nous

constatons qu'à chaque nouvel ajout de segment, nous avons : $s \rightarrow s$; $c \rightarrow c + 1$; $F \rightarrow F + 1$; $D' \rightarrow D'$.

Une fois que toute la figure est divisée en triangles, il ne reste plus qu'à ramener celle-ci à un seul triangle au moyen des deux opérations suivantes qui conservent D' :

1) On peut enlever un triangle périphérique en retirant un seul côté reliant deux sommets (cf. ligne en gras sur la Fig. A1.1 b). Pour cette opération : $s \rightarrow s$; $c \rightarrow c - 1$; $F \rightarrow F - 1$; $D' \rightarrow D'$.
2) On peut enlever un triangle périphérique en retirant deux côtés adjacents et un sommet (cf. lignes pointillées sur la Fig. A1.1 b). Pour cette opération : $s \rightarrow s - 1$; $c \rightarrow c - 2$; $F \rightarrow F - 1$; $D' \rightarrow D'$.

Mais, pour un seul triangle, nous savons que $s = 3$; $c = 3$; $F = 1$; $D' = 1$ de sorte que pour notre coquille de départ, nous avons $D' = 1$ et que pour notre polyèdre $D = D' + 1 = 2$, achevant ainsi notre démonstration.

A1.2. Caractéristiques des fullérènes.

Dans ce qui va suivre, nous allons noter par c le nombre de côtés de la fullérène considérée, par s son nombre de sommets, par f son nombre de faces, par h son nombre de faces hexagonales et par p son nombre de faces pentagonales. Alors, comme le montre la figure A1.2 il est aisé de relier les variables s, c, f, p, h entre elles :

$$2c = 5p + 6h$$
$$4c = 6s$$
$$f = h + p$$
$$s - c + f = 2 \quad \text{(Théorème d'Euler)}$$

Un système d'équations, qui après quelques manipulations permet d'exprimer toutes les variables en terme du nombre d'atomes de carbone et qui permet de conclure que toute fullérène (y compris les nanotubes) doit avoir 12 pentagones.

$$\begin{aligned} p &= 12 \\ c &= 3/2\ s \\ f &= s/2 + 2 \\ h &= s/2 - 10 \end{aligned} \qquad \text{(A1.1)}$$

Puisque h est un entier ≥ 0, on voit du système précédent que la plus petite fullérène possible doit avoir : p = 12; h = 0; s = 20; c = 30. Cette fullérène est illustrée à la figure A1.3.

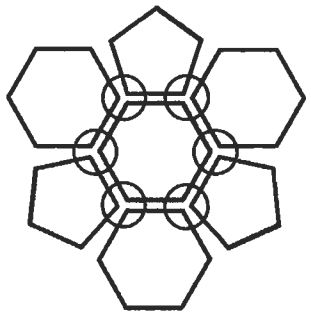


Figure A1.2. Lien entre les variables s, c, f, p, h d'une fullérène.

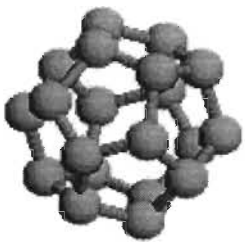


Figure A1.3. La plus petite fullérène possible : le C20.

Développement en séries des fonctions M_n(x) autour de la singularité x = 1, dans le but de pouvoir vérifier que le potentiel de Stan & Cole V(r, R) se réduit bel et bien au potentiel d'une feuille de graphène V_flat(z) lorsque r tend vers R (le rayon du nanotube).

```
>
> restart:
> M:= (x, n) -> int((1 + x^2 - 2*x*cos(phi))^(-n/2), phi = 0..Pi);
```

$$M := (x, n) \to \int_0^{\pi} (1 + x^2 - 2\,x\cos(\phi))^{(-1/2\,n)}\, d\phi$$

```
>
```

1. Calcul d'une expansion en série de la fonction M_5(x) autour de x = 1:

```
> serieM5:=series(M(x,5), x = 1, 10);
```

$$serieM5 := \frac{2}{3}(x-1)^{-4} - \frac{1}{3}(x-1)^{-3} + \frac{7}{24}(x-1)^{-2} - \frac{13}{48}(x-1)^{-1} + \left(\frac{331}{1536} + \frac{3}{128}\ln(8) - \frac{3}{128}\ln(x-1)\right) + \left(-\frac{403}{3072} - \frac{15}{256}\ln(8) + \frac{15}{256}\ln(x-1)\right)(x-1) + \left(\frac{339}{8192} + \frac{185}{2048}\ln(8) - \frac{185}{2048}\ln(x-1)\right)(x-1)^2 + \left(-\frac{455}{4096}\ln(8) + \frac{455}{4096}\ln(x-1) + \frac{583}{16384}\right)(x-1)^3 + O((x-1)^4)$$

Cette série **n'est pas une série de Laurent** malgré la présence des puissances entières négatives et positives de (x-1). Cela est dû à la présence des termes ln(x-1) présents dans les coefficients de la série. En effet, en interrogeant Maple:

```
> type(serieM5, laurent);
```

false

```
> whattype(serieM5);
```

series

Malgré cela, seulement le terme (x-1) ^ -4 domine dans la série précédente près de x = 1 puisque ln(x-1) * ((x-1) ^ n) ---> 0 pour n >= 1. En effet, en calculant cette limite:

```
> assume(n >= 1);
> limit( ((x-1)^n) * ln(x-1) ,x=1);
```

0

En plus du terme (x-1) ^ -4, les autres puissances négatives de (x-1) et le terme -(3/128) * ln(x-1) correspondant à la puissance (x-1) ^ 0 divergent aussi vers l'infini. Malgré cela, on peut ne retenir que le terme en (x-1) ^ -4 près de x = 1 puisqu'il diverge plus rapidement que les autres. En effet, en calculant le rapport des termes divergeants lorsque x ---> 1:

```
> limit( ((x-1)^(-3)) / ((x-1)^(-4)),x=1);        # pour les autres
  puissances négatives.
```

0

```
> limit( (ln(x-1)) / ((x-1)^(-4)),x=1);           # pour le terme
```

```
-(3/128)*(ln(x-1))*(x-1)^0.
```

$$0$$

Donc, de tous ces calculs nous pouvons conclure que M_5(x) ~ (2/3) * (x-1) ^ -4 près de x = 1.

>

2. Calcul du terme dominant de l'expansion en série de la fonction M_11(x) autour de x = 1:

```
> series('leadterm'(M(x, 11)), x = 1, 20);      # Maple calcule
  directement le terme dominant de la série.
```

$$\frac{128}{315}(x-1)^{-10}$$

>

ANNEXE 3. GRAPHIQUES DU COMPORTEMENT DU B_{AS} D'UN FAISCEAU DE SWNTs.

Dans cette annexe, nous allons présenter l'ensemble des courbes de B_{AS} obtenues lors de ce travail. Leur discussion et interprétation ont été données au chapitre 5. Cependant, il est bon de rappeler que dans tous les graphiques qui vont suivre, les courbes noires (grises) seront utilisées lorsque les SWNTs sont ouverts (fermés) aux extrémités. Les courbes continues (en tirets) seront pour N_{tubes} SWNTs regroupés en faisceau (isolés).

A3.1. Comportement du B_{AS} en fonction du pas du réseau.

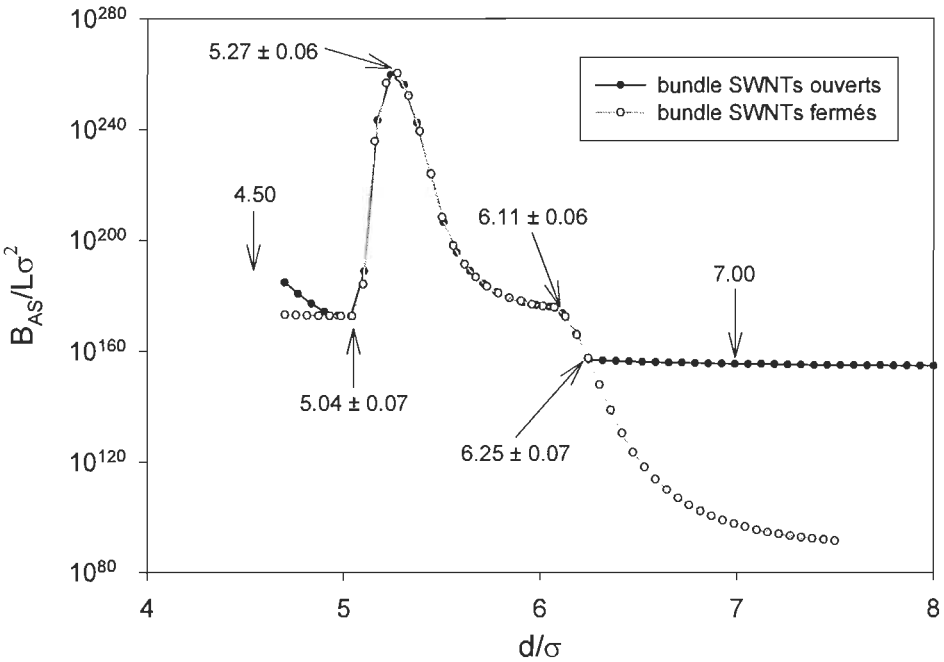


Figure A3.1. Comportement du B_{AS} en fonction du pas du réseau. $N_{tubes} = 25$, $R^* = 2.038$ (H_2 : 6.5 Å), $T^* = 0.005$ (H_2 : 1.85 K).

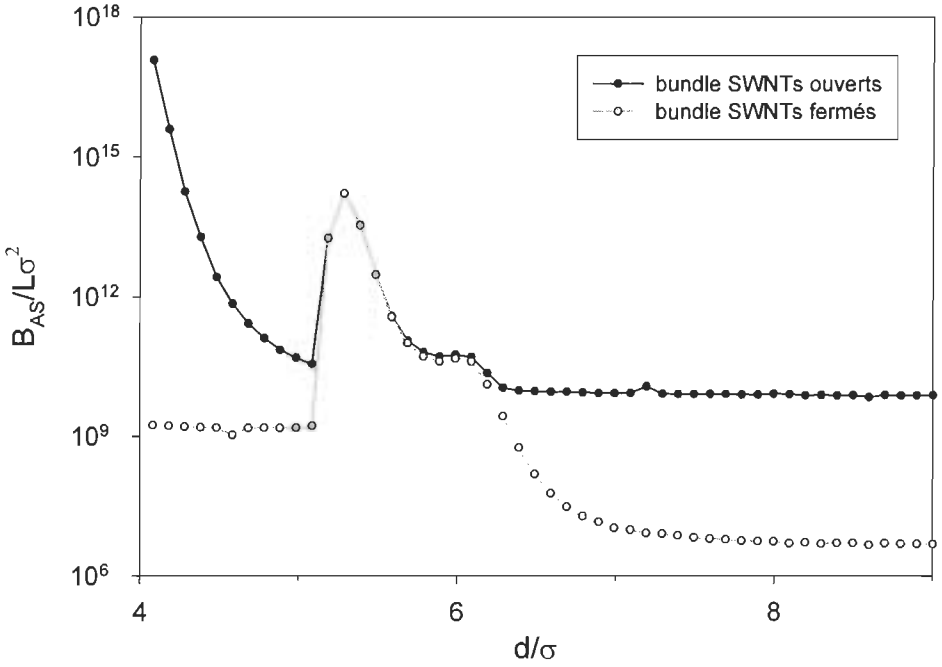


Figure A3.2. Comportement du B_{AS} en fonction du pas du réseau. $N_{tubes} = 25$, $R^* = 2.038$ (H_2 : 6.5 Å), $T^* = 0.08900$ (H_2 : 32.98 K).

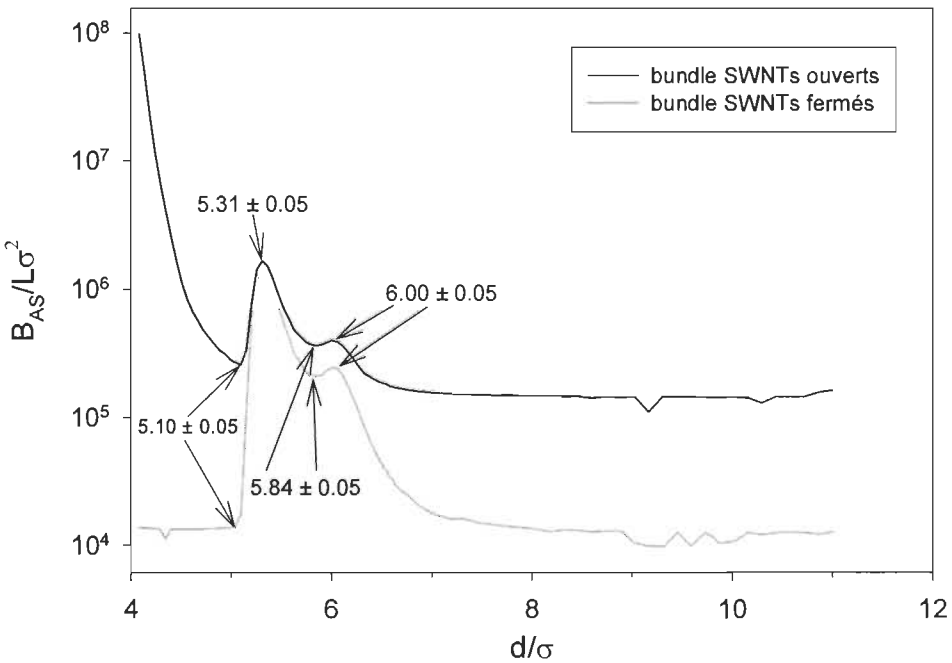


Figure A3.3. Comportement du B_{AS} en fonction du pas du réseau. $N_{tubes} = 25$, $R^* = 2.038$ (H_2 : 6.5 Å), $T^* = 0.2089$ (H_2 : 77.40 K).

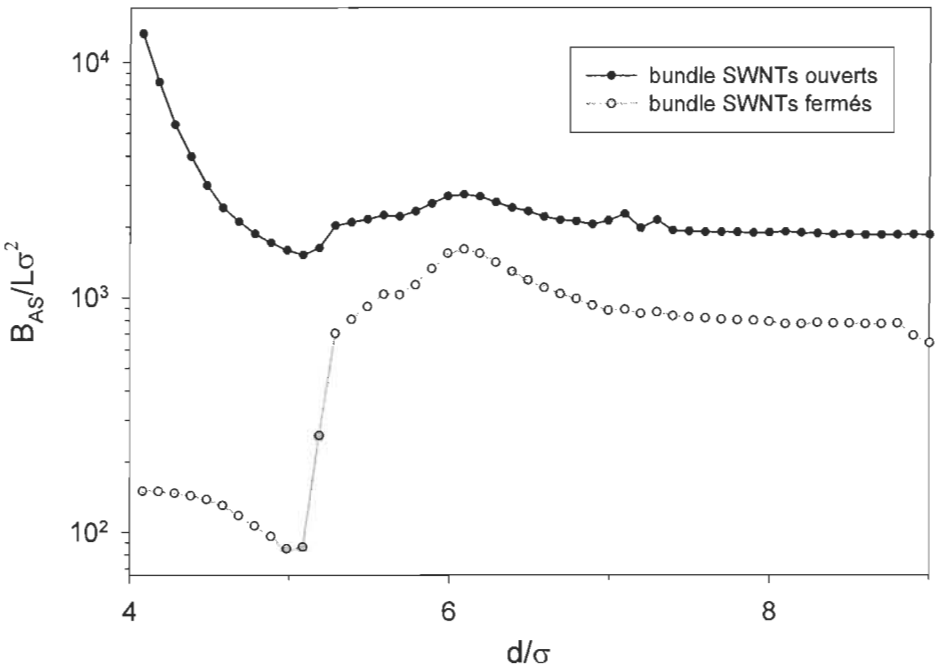


Figure A3.4. Comportement du B_{AS} en fonction du pas du réseau. $N_{tubes} = 25$, $R^* = 2.038$ (H_2 : 6.5 Å), $T^* = 0.5252$ (H_2 : 194.6 K).

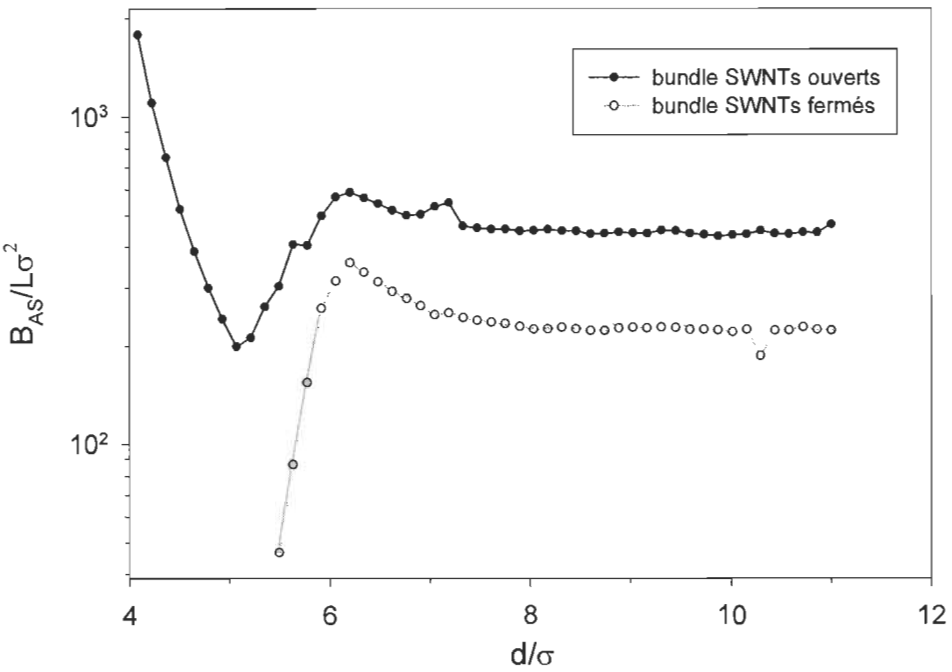


Figure A3.5. Comportement du B_{AS} en fonction du pas du réseau. $N_{tubes} = 25$, $R^* = 2.038$ (H_2 : 6.5 Å), $T^* = 0.8097$ (H_2 : 300.0 K).

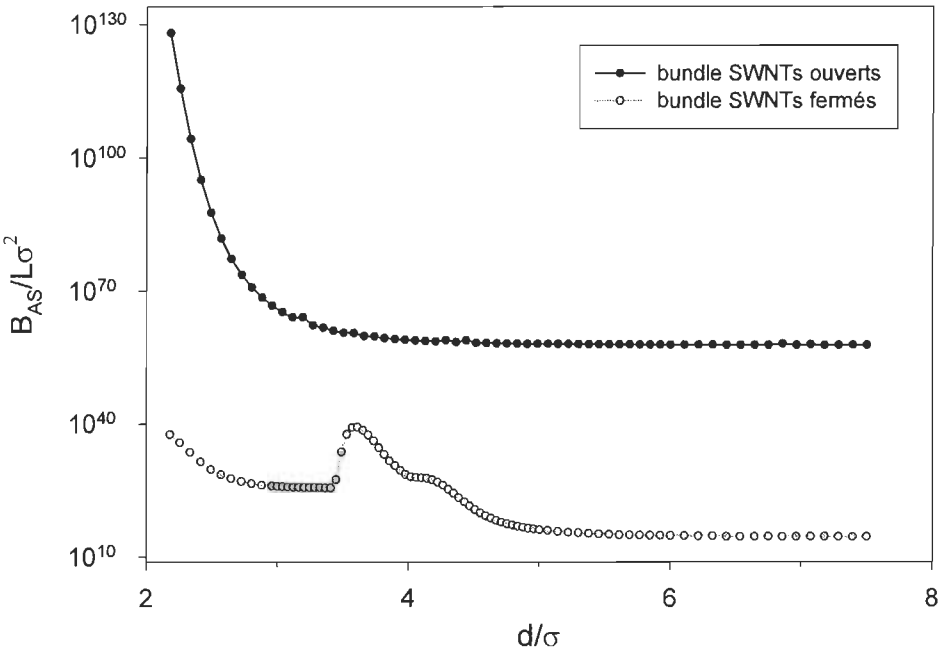


Figure A3.6. Comportement du B_{AS} en fonction du pas du réseau. $N_{tubes} = 25$, $R^* = 1.086$ (H_2 : 3.464 Å), $T^* = 0.03$ (H_2 : 11.12 K).

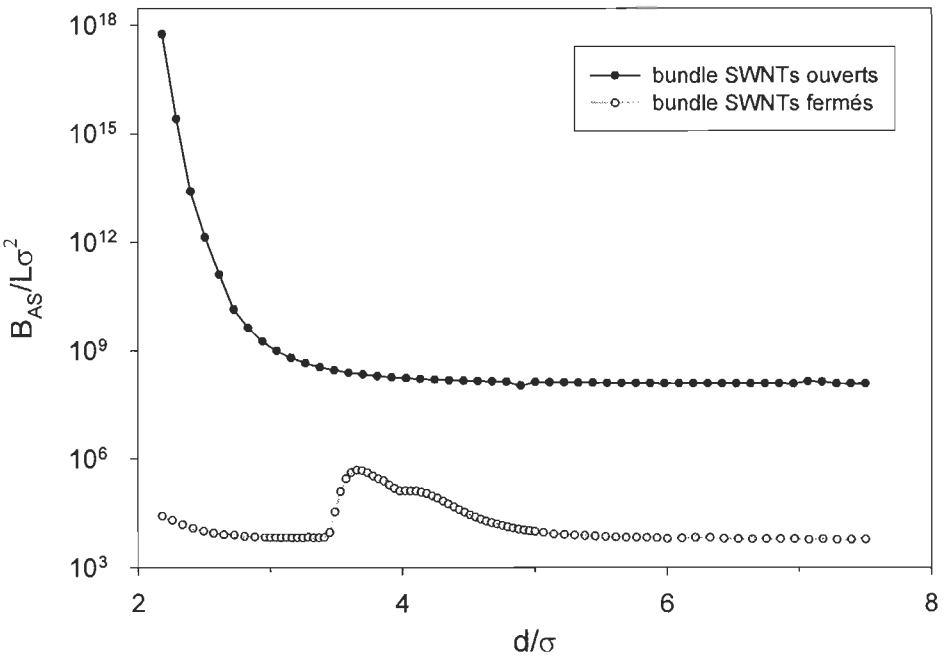


Figure A3.7. Comportement du B_{AS} en fonction du pas du réseau. $N_{tubes} = 25$, $R^* = 1.086$ (H_2 : 3.464 Å), $T^* = 0.2089$ (H_2 : 77.40 K).

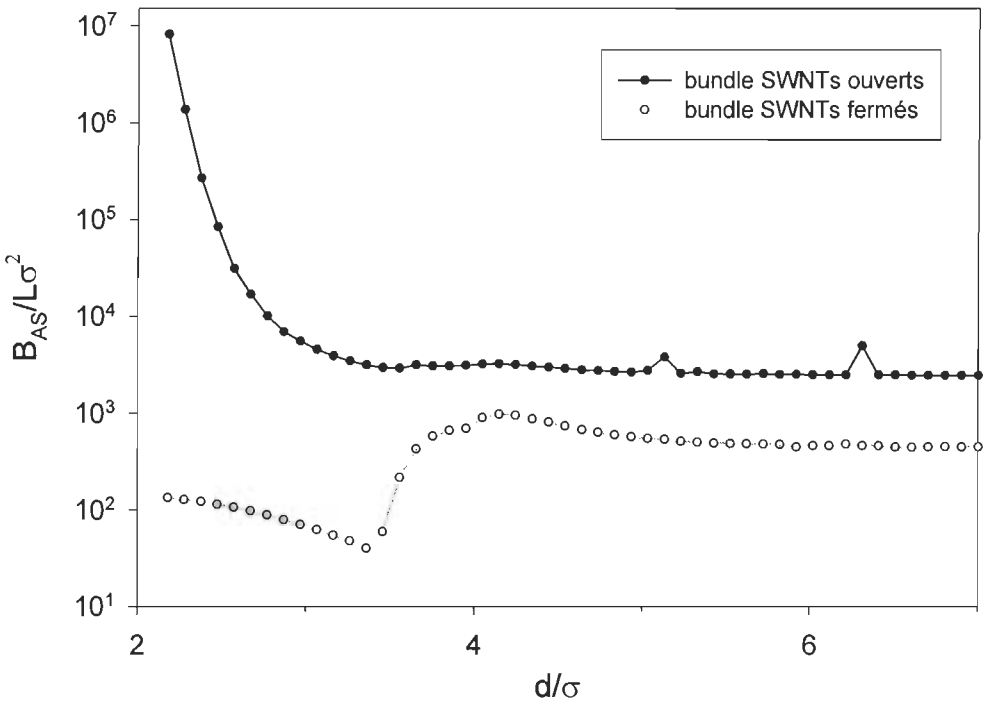


Figure A3.8. Comportement du B_{AS} en fonction du pas du réseau. $N_{tubes} = 25$, $R^* = 1.086$ (H_2 : 3.464 Å), $T^* = 0.5252$ (H_2 : 194.6 K).

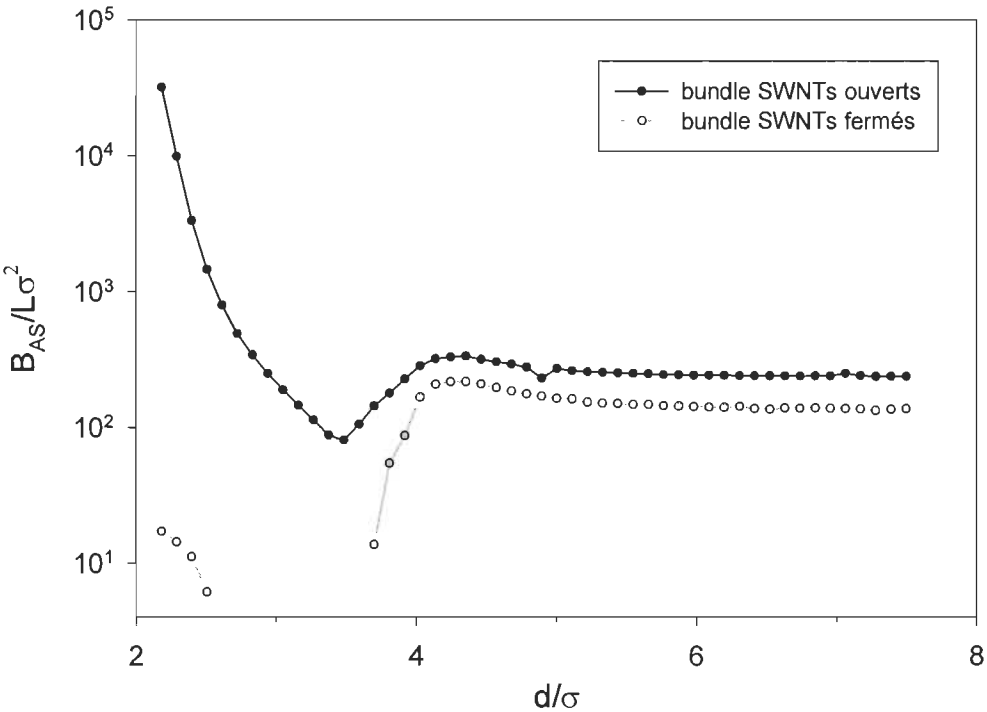


Figure A3.9. Comportement du B_{AS} en fonction du pas du réseau. $N_{tubes} = 25$, $R^* = 1.086$ (H_2 : 3.464 Å), $T^* = 0.8097$ (H_2 : 300.0 K).

A3.2. Comportement du B_{AS} en fonction du rayon des nanotubes.

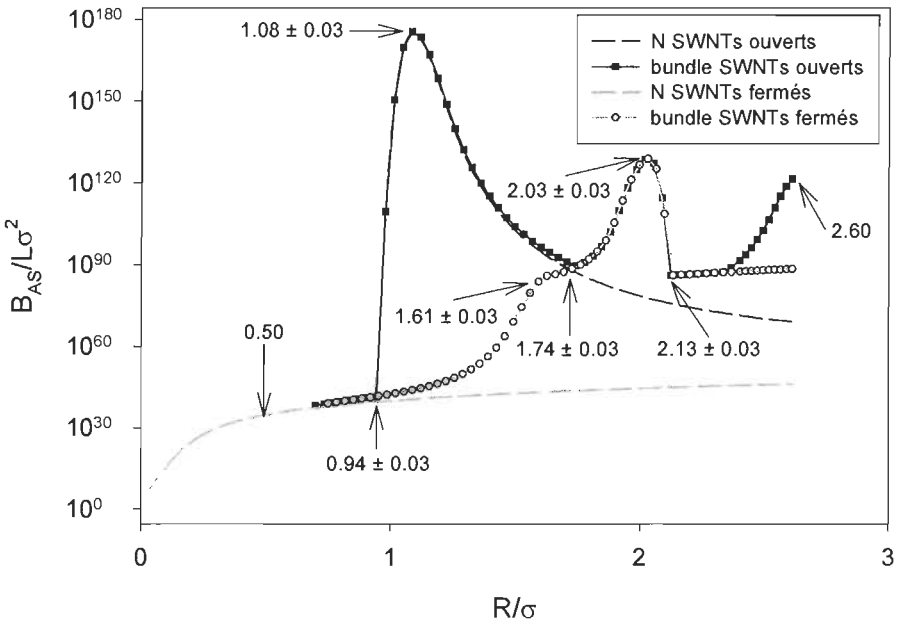


Figure A3.10. Comportement du B_{AS} en fonction du rayon des nanotubes. $N_{tubes} = 25$, $d^* = 5.235$ (H_2 : 16.7 Å), $T^* = 0.01$ (H_2 : 3.71 K).

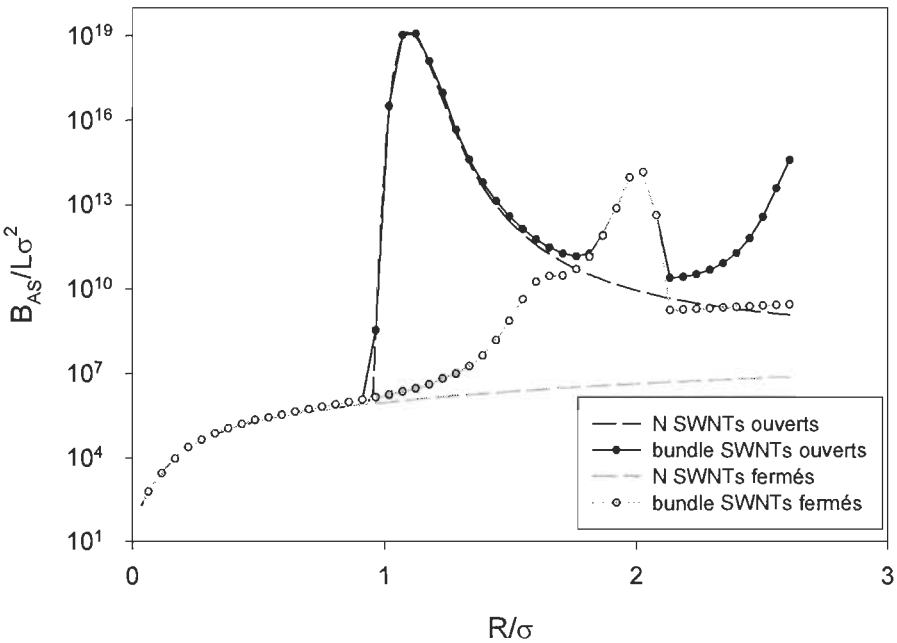


Figure A3.11. Comportement du B_{AS} en fonction du rayon des nanotubes. $N_{tubes} = 25$, $d^* = 5.235$ (H_2 : 16.7 Å), $T^* = 0.08900$ (H_2 : 32.98 K).

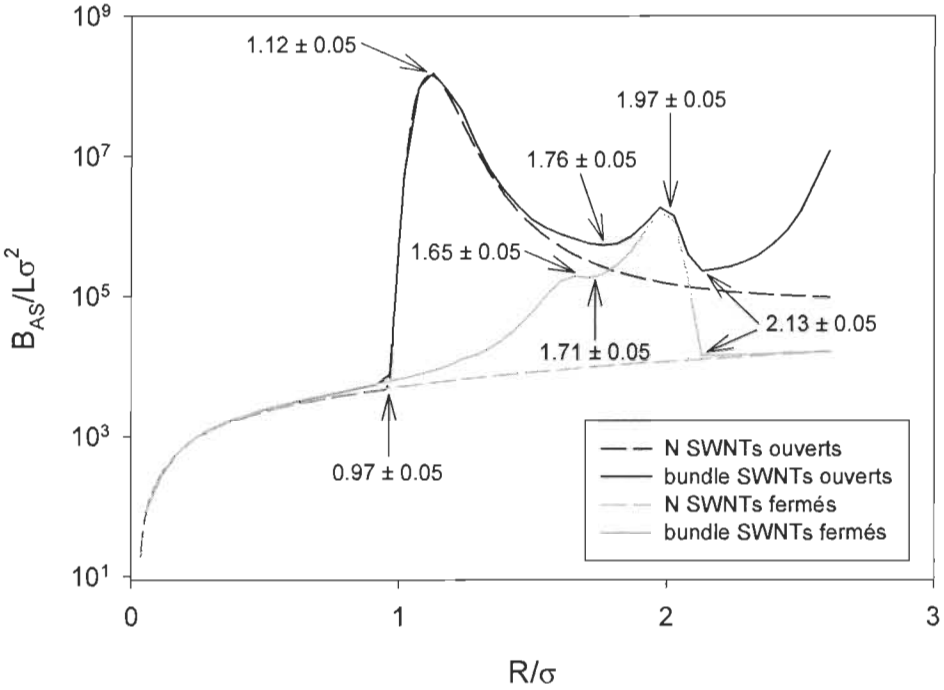


Figure A3.12. Comportement du B_{AS} en fonction du rayon des nanotubes. $N_{tubes} = 25$, $d^* = 5.235$ (H_2 : 16.7 Å), $T^* = 0.2089$ (H_2 : 77.40 K).

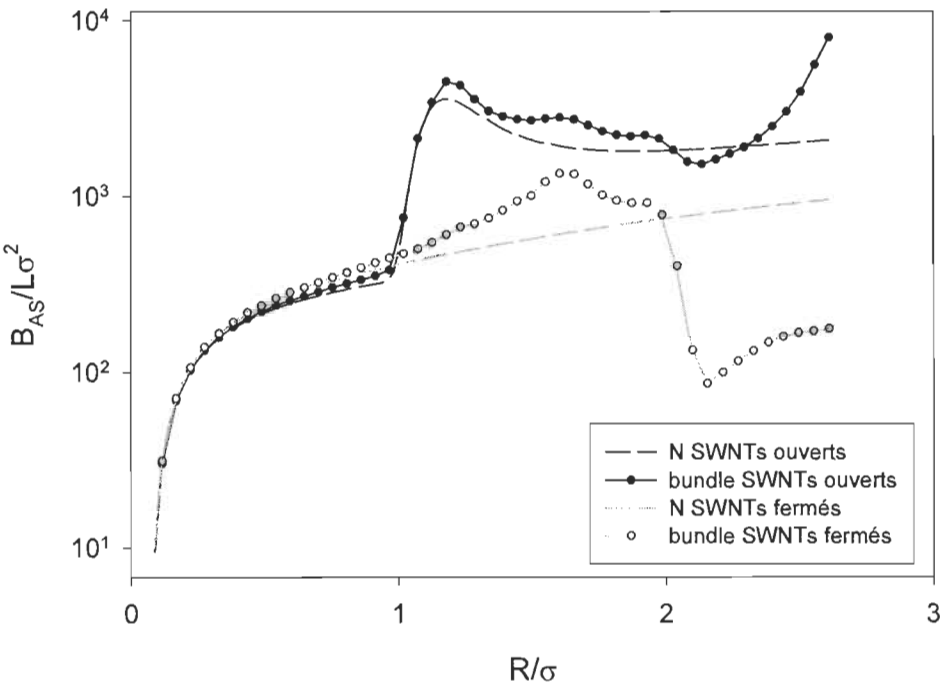


Figure A3.13. Comportement du B_{AS} en fonction du rayon des nanotubes. $N_{tubes} = 25$, $d^* = 5.235$ (H_2 : 16.7 Å), $T^* = 0.5252$ (H_2 : 194.6 K).

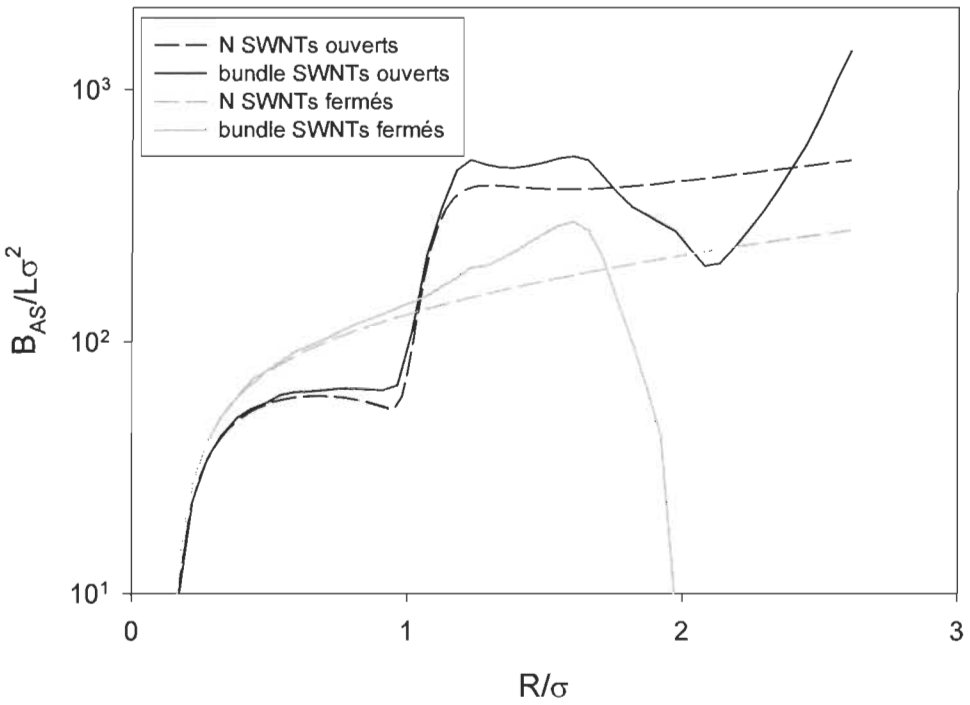


Figure A3.14. Comportement du B_{AS} en fonction du rayon des nanotubes. $N_{tubes} = 25$, $d^* = 5.235$ (H_2 : 16.7 Å), $T^* = 0.8097$ (H_2 : 300.0 K).

A3.3. Comportement du B_{AS} en fonction de la température.

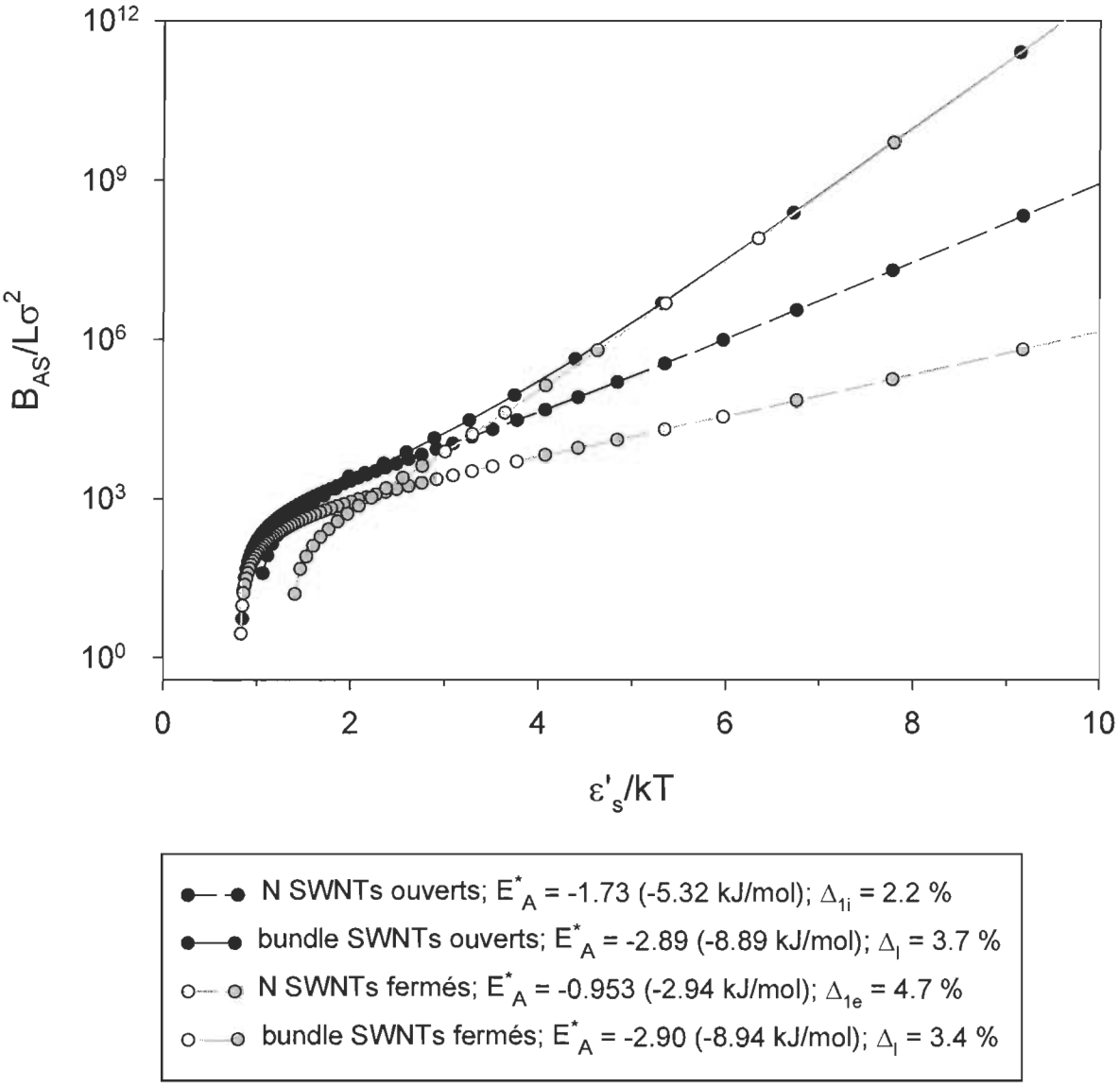


Figure A3.15. Comportement du B_{AS} en fonction de la température. $N_{tubes} = 25$, $d^* = 5.235$ (H_2 : 16.7 Å), $R^* = 2.038$ (H_2 : 6.5 Å).

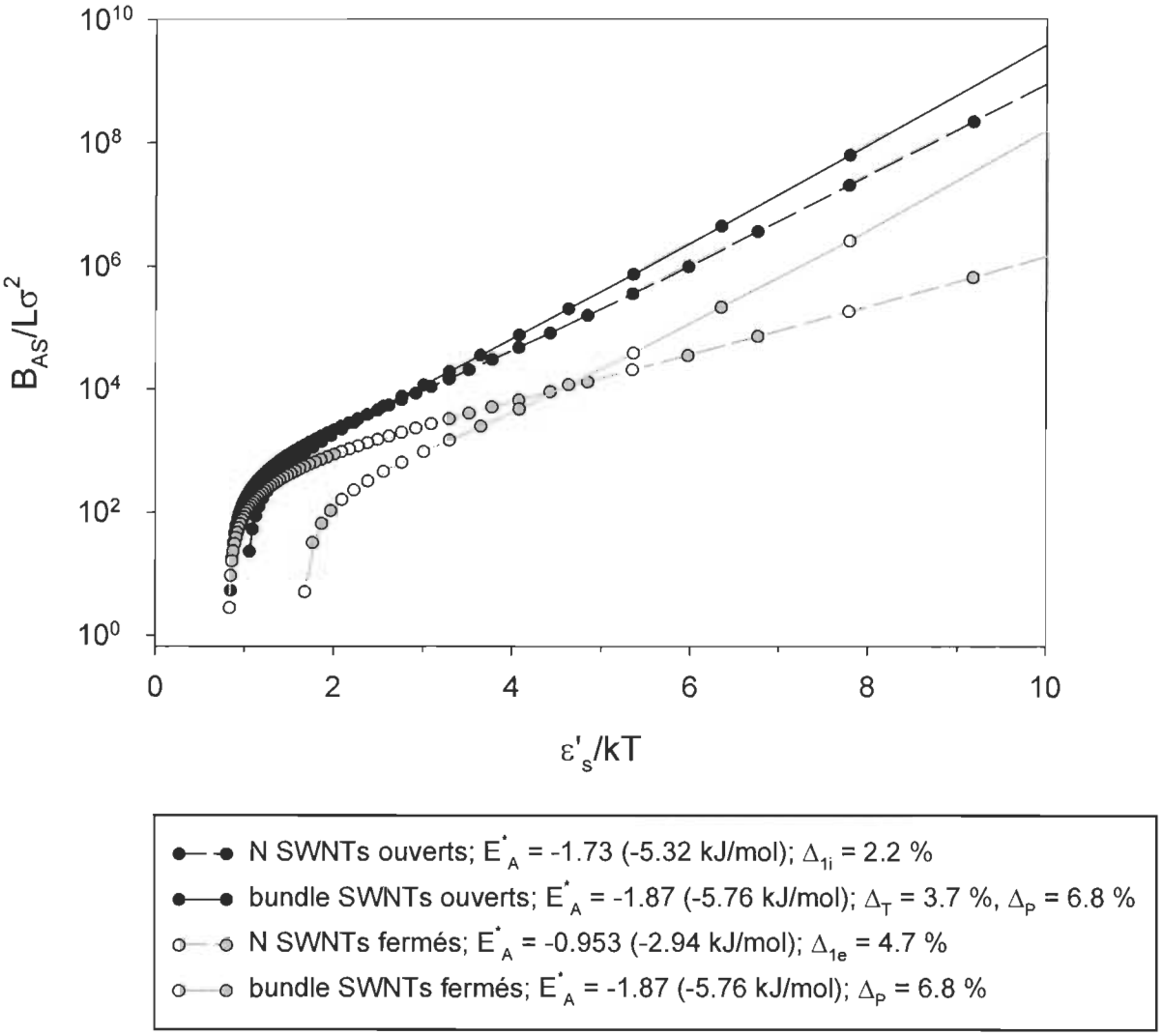


Figure A3.16. Comportement du B_{AS} en fonction de la température. $N_{tubes} = 25$, $d^* = 5.063$ (H_2 : 16.15 Å), $R^* = 2.038$ (H_2 : 6.5 Å).

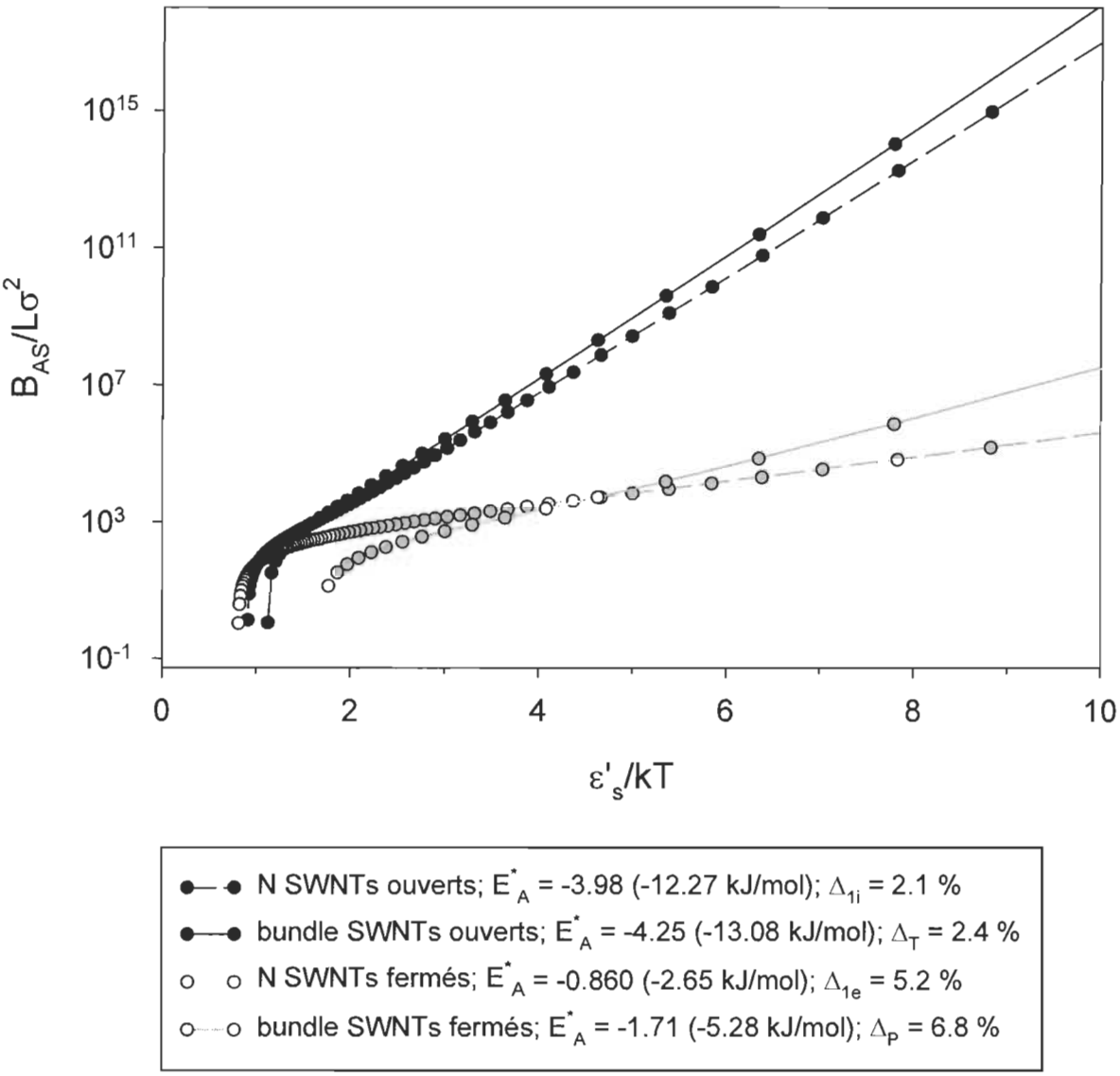


Figure A3.17. Comportement du B_{AS} en fonction de la température. $N_{tubes} = 25$, $d^* = 3.332$ (H_2 : 10.63 Å), $R^* = 1.086$ (H_2 : 3.464 Å).

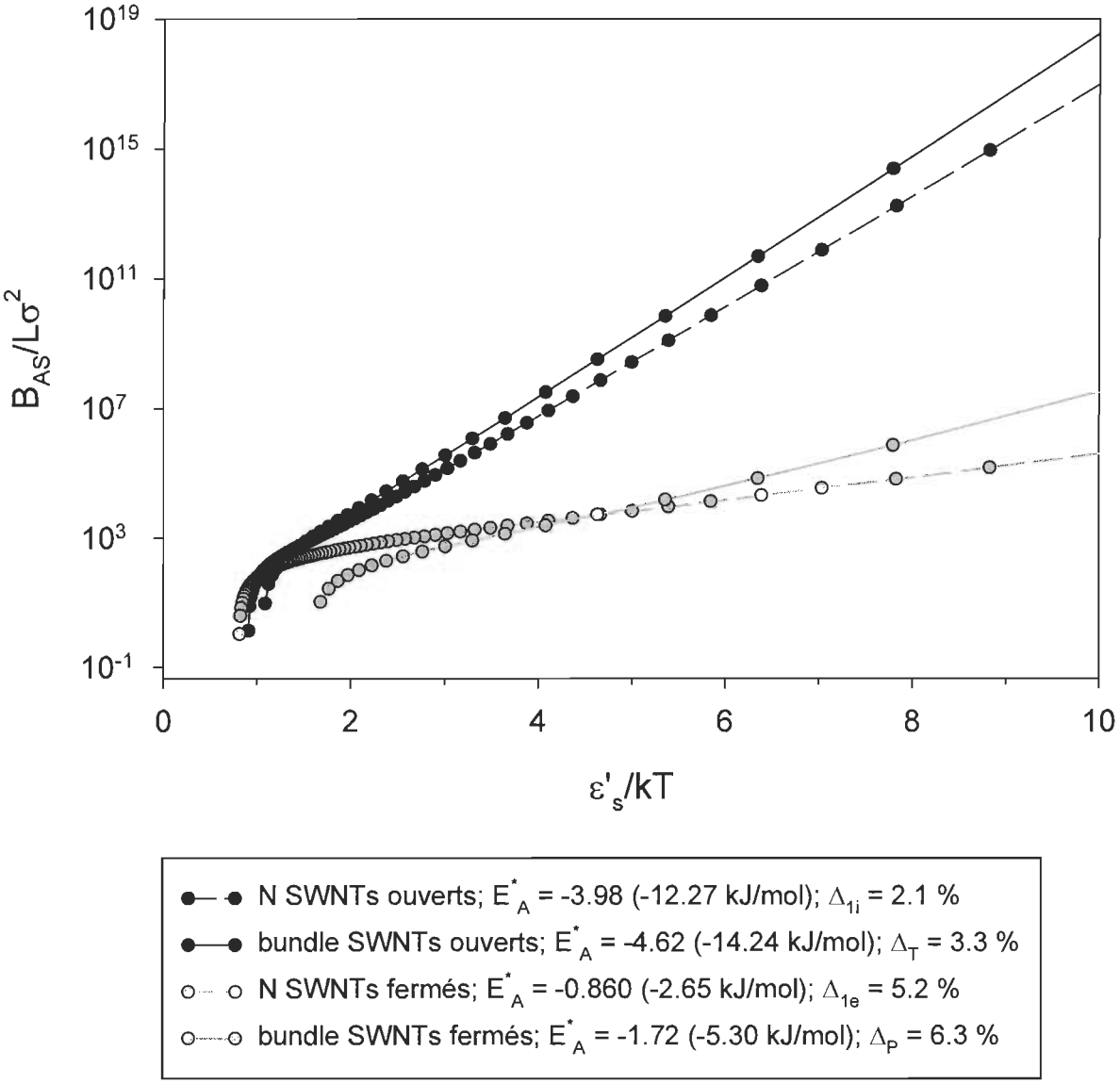


Figure A3.18. Comportement du B_{AS} en fonction de la température. $N_{tubes} = 25$, $d^* = 3.159$ (H_2 : 10.08 Å), $R^* = 1.086$ (H_2 : 3.464 Å).

A3.4. Comportement du B_{AS} en fonction du nombre de nanotubes.

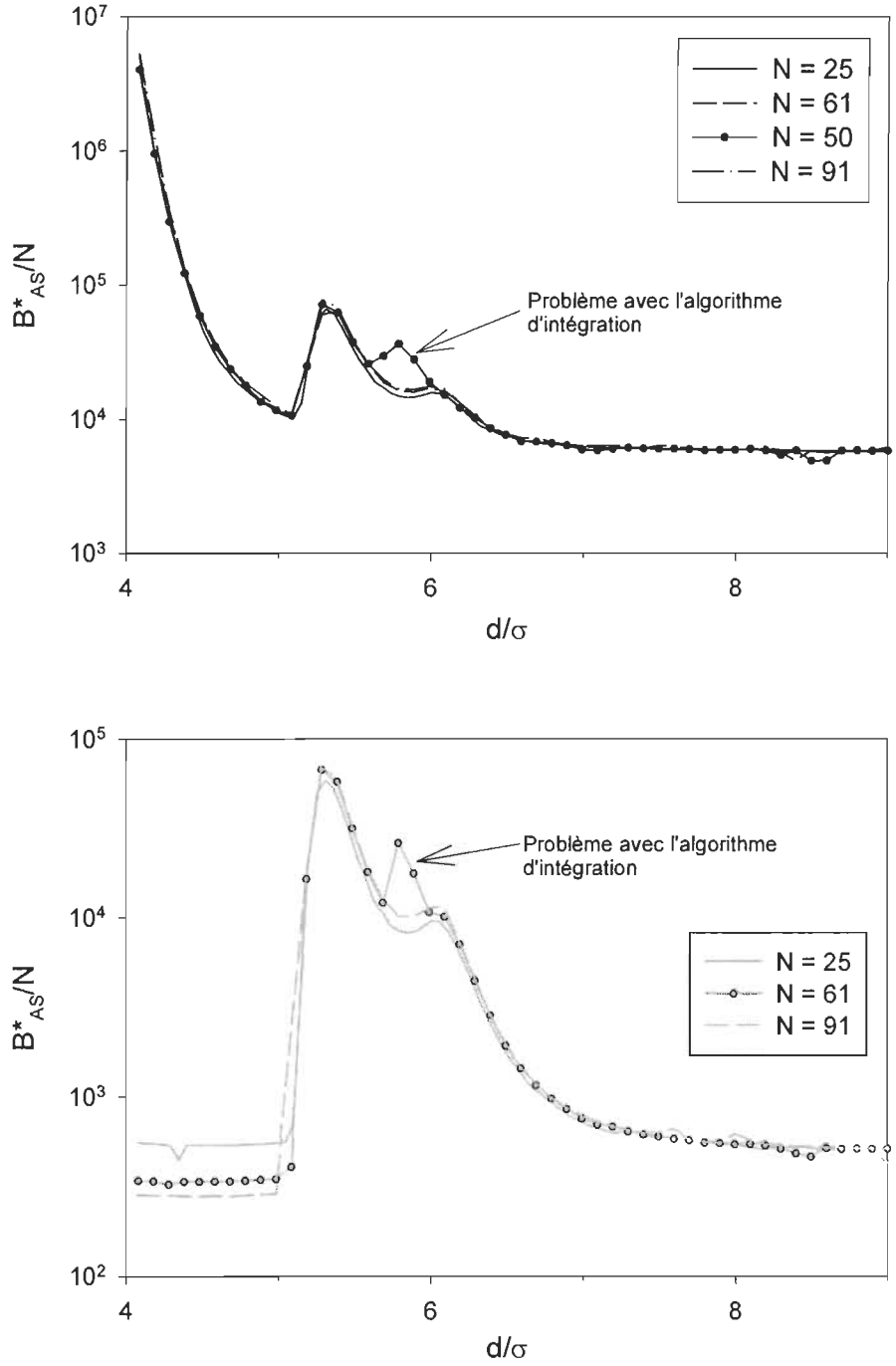


Figure A3.19. Influence du nombre de nanotubes sur le comportement du B_{AS} d'un faisceau en fonction du pas du réseau. R* = 2.038 (H_2 : 6.5 Å), T* = 0.2089 (H_2 : 77.40 K).

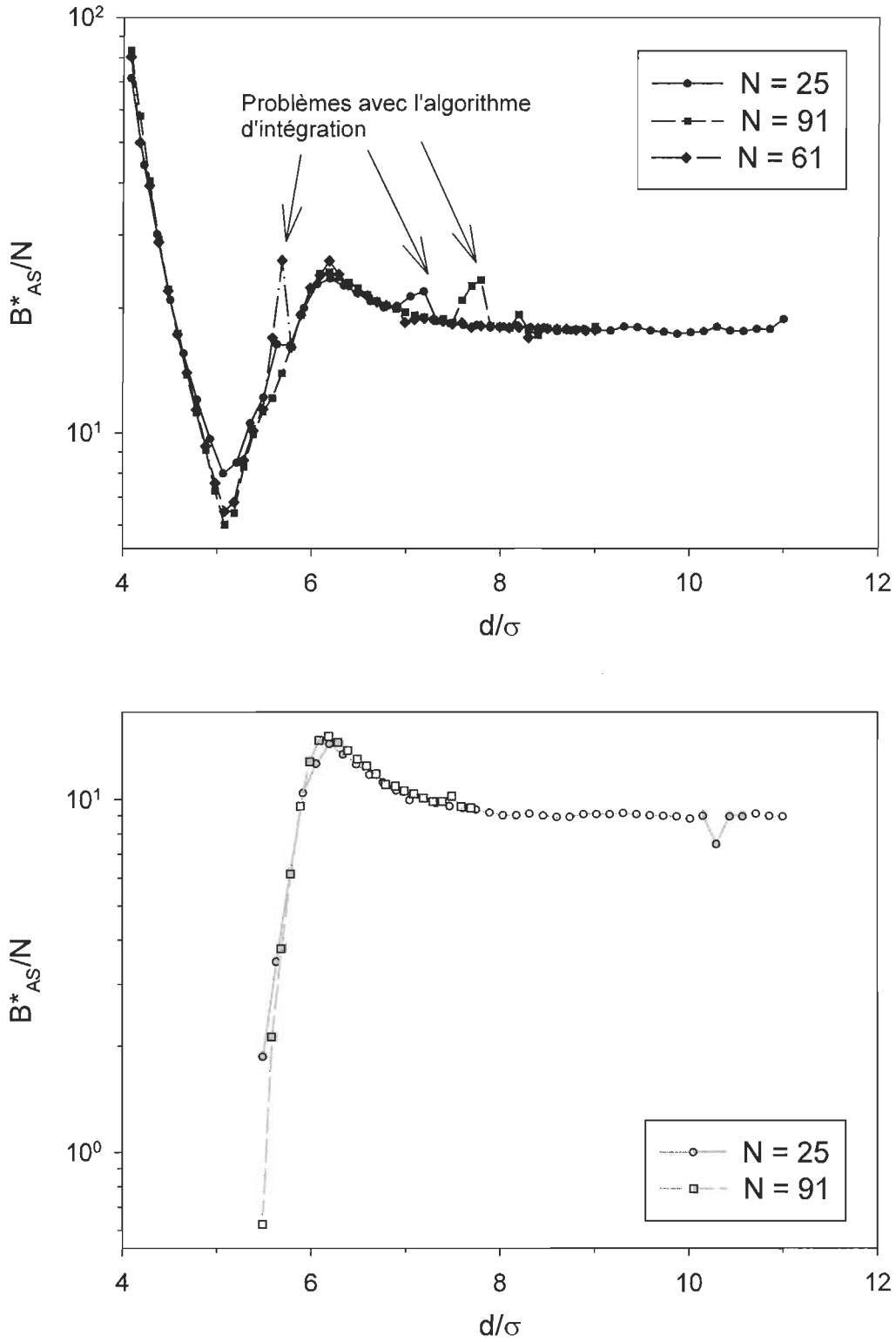


Figure A3.20. Influence du nombre de nanotubes sur le comportement du B_{AS} d'un faisceau en fonction du pas du réseau. R* = 2.038 (H_2 : 6.5 Å), T* = 0.8097 (H_2 : 300.0 K).

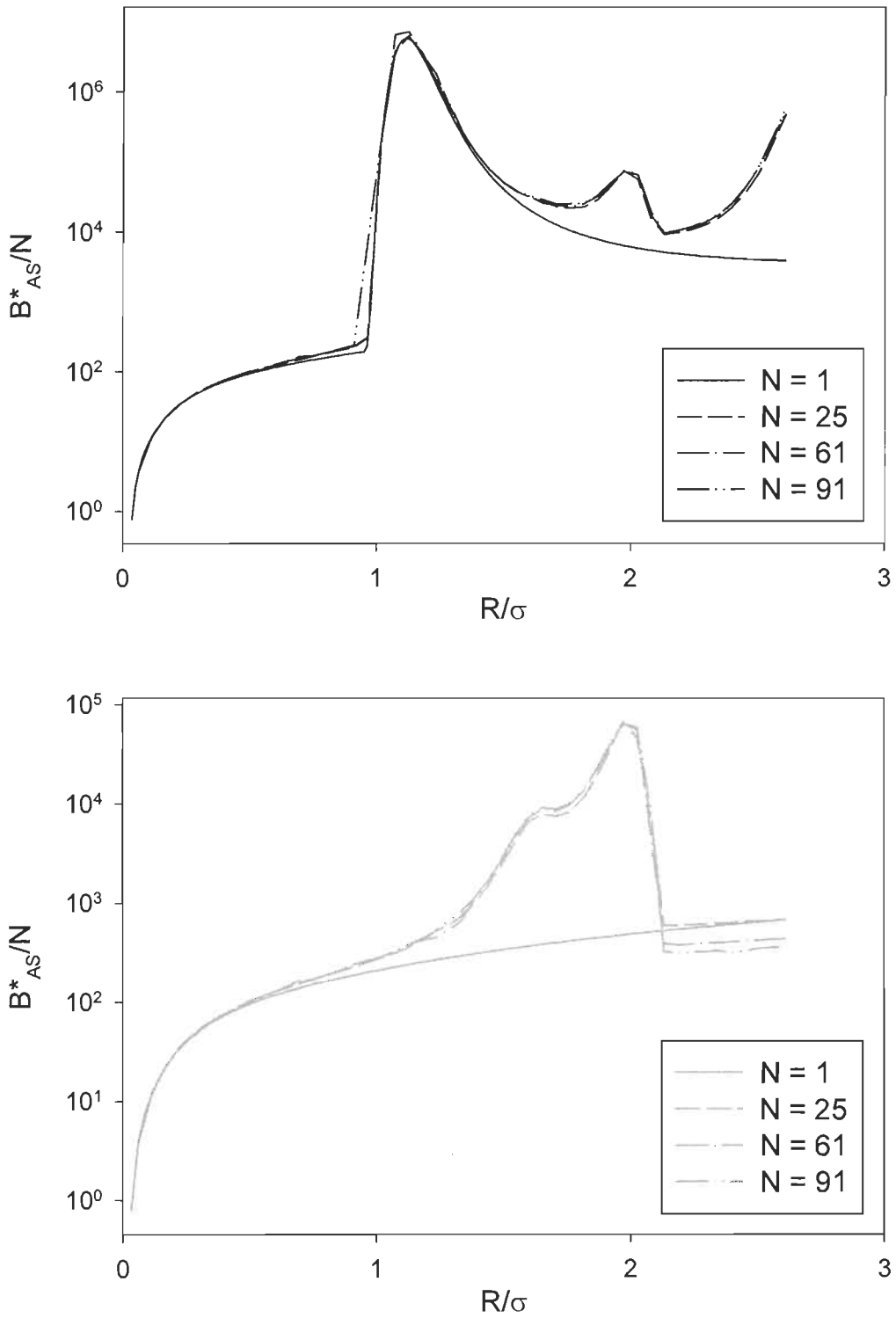


Figure A3.21. Influence du nombre de nanotubes sur le comportement du B_{AS} d'un faisceau en fonction du rayon des nanotubes. $d^* = 5.235$ (H_2 : 16.7 Å), $T^* = 0.2089$ (H_2 : 77.40 K).

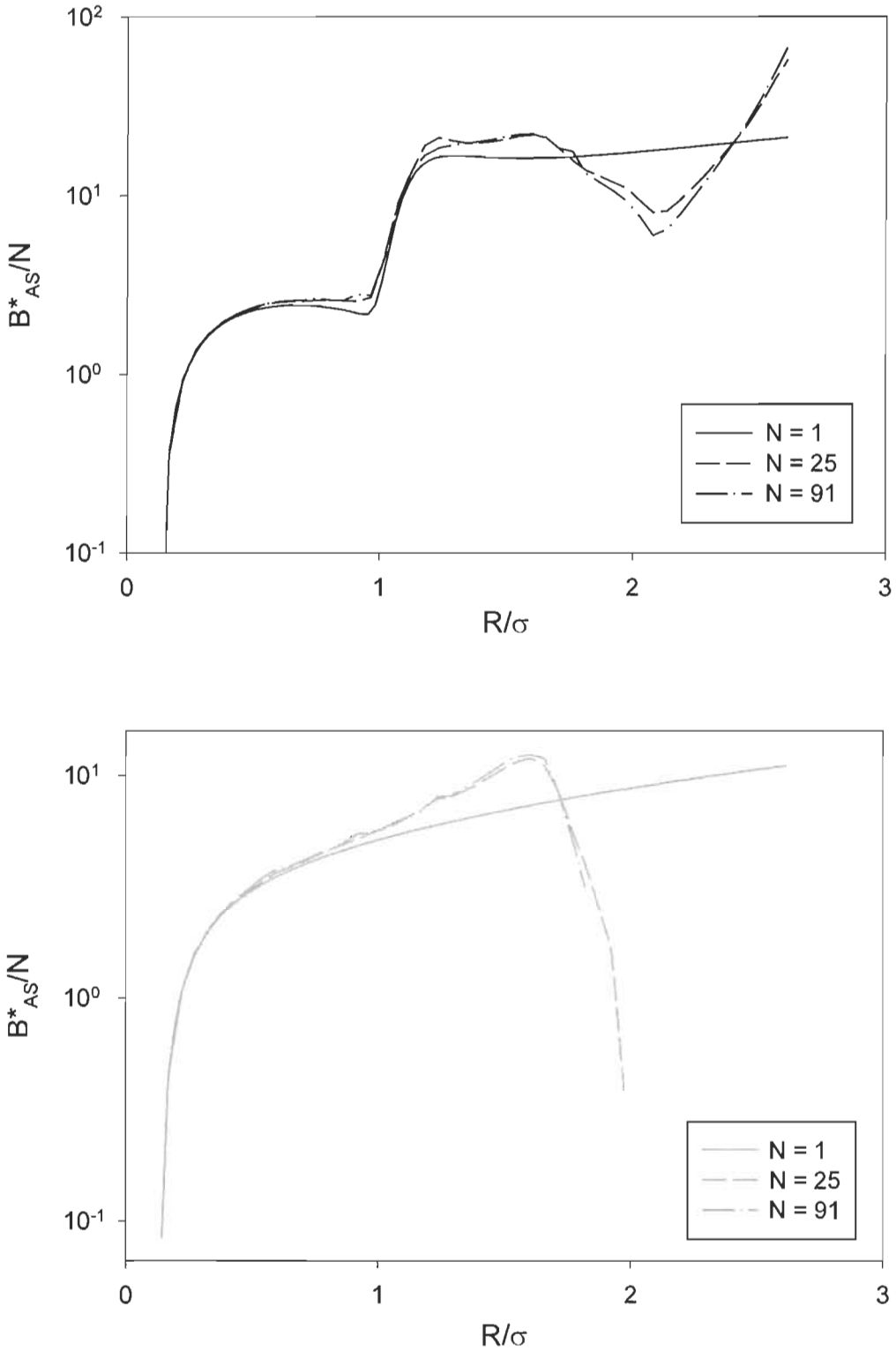


Figure A3.22. Influence du nombre de nanotubes sur le comportement du B_{AS} d'un faisceau en fonction du rayon des nanotubes. d* = 5.235 (H_2 : 16.7 Å), T* = 0.8097 (H_2 : 300.0 K).

A3.5. Vérification du programme calculant le B_{AS} d'un faisceau dans le cas d'un seul SWNT.

Afin de vérifier le bon fonctionnement du programme calculant le B_{AS} d'un faisceau de SWNTs, nous l'avons utilisé pour calculer le B_{AS} d'un seul nanotube ($N_{tubes} = 1$). Les graphiques montrent les résultats ainsi obtenus, superposés avec ceux calculés par le programme calculant le B_{AS} d'un seul nanotube à parois multiples (MWNT, où $N_{parois} = 1$). La concordance entre les deux programmes est excellente.

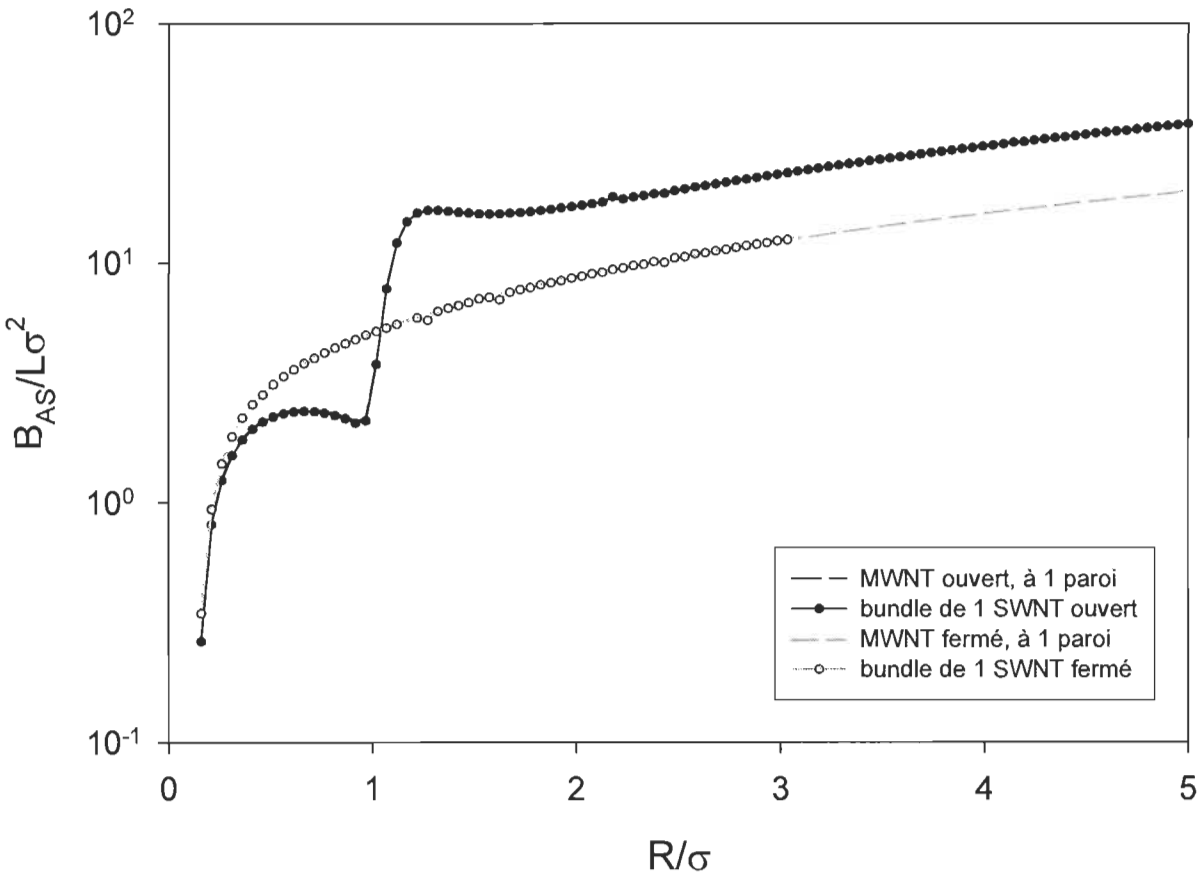


Figure A3.23. Vérification du comportement du B_{AS} d'un faisceau ayant un seul SWNT, pour R* variable. $N_{tubes} = 1$, $T^* = 0.8097$ (H_2 : 300.0 K).

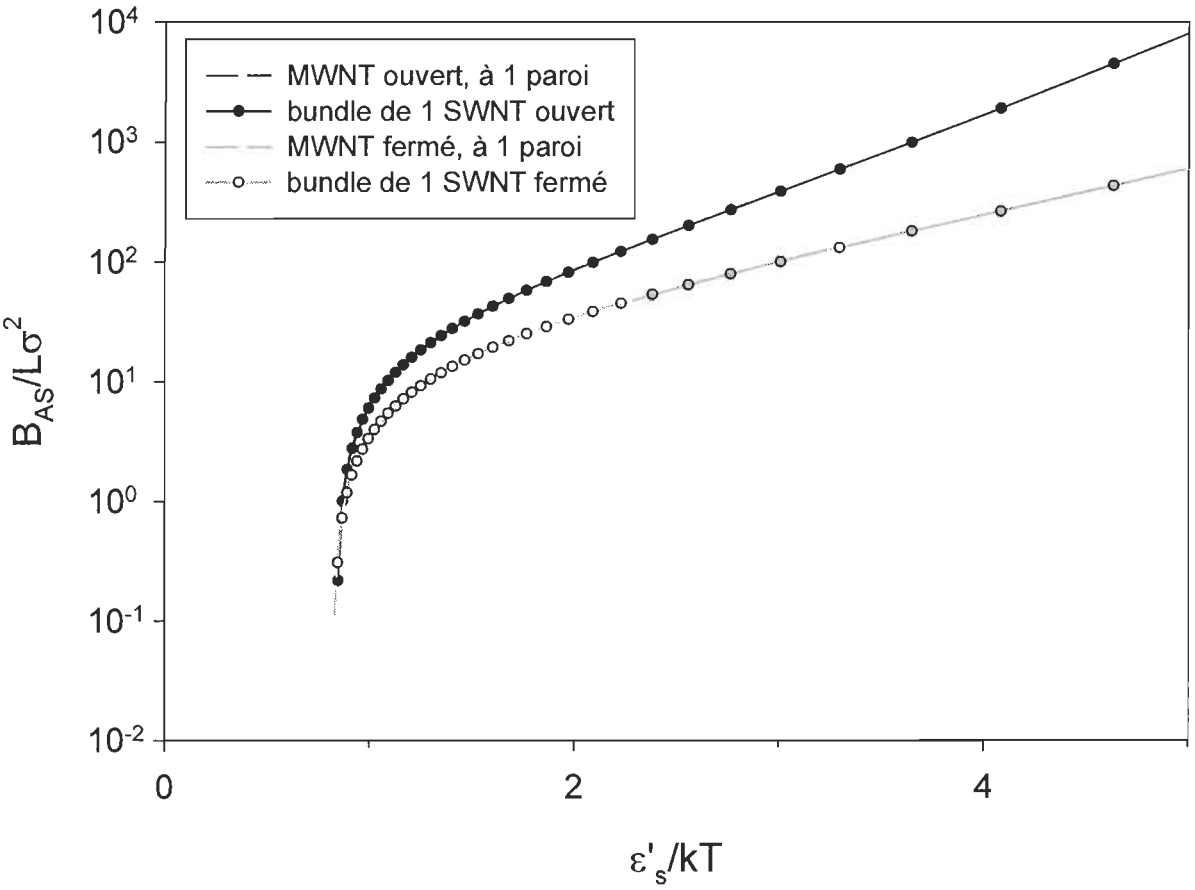


Figure A3.24. Vérification du comportement du B_{AS} d'un faisceau ayant un seul SWNT, pour T* variable. $N_{tubes} = 1$, R* = 2.038 (H_2 : 6.5 Å).

```
/*****************************************************************

  Fichier d'en-tête traitement.h

  Contient les librairies, les constantes et les déclaration
  (i.e les prototypes) de fonctions.

*****************************************************************/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <iostream.h>       /* Je dois utiliser la fonction
                               "cout << " à cause du bogue avec
                               scanf().
                               */

#define PI 3.14159265358979323846264338327 95
#define ERREUR_RELATIVE_Bas 1e-5
                            /* erreur (relative) tolérée dans la fonction
                               simpson_rafinee_Bas().  C'est l'incertitude
                               relative de l'intégrale calculant Bas.
                               (i.e. que c'est un % d'incertitude sur
                               l'intégrale).  */
#define JMAX 20             /* est le nombre de raffinages MAXIMUM qu'on
                               fait sur l'intégrale de départ.
                               2^(JMAX-1) est le nombre total (maximum)
                               de sous-intervalles pour l'intégration. */
#define THETA 0.38          /* Angstrom^-2
                               Densité surfacique des atomes sur la paroi
                               du nanotube (ou du plan de graphite). */

#define k 1.0               /* constante de Boltzmann (= 1 lorsque
                               les énergies sont exprimées en K).
                               */

#define DISTREDUITEMIN 0.01
                            /* distance (réduite) minimale de la paroi.
                               À l'intérieur de cette distance d'exclusion,
                               le potentiel est considéré comme infini et
                               integrand_Bas = exp(-V/kT) - 1 ===> -1.

                               Cette distance est choisie de sorte que
                               integrand_Bas soit très près de -1.
                               */
#define BORNE_GAUCHE_Bas 0.0
                            /* distance RÉDUITE (d'un seul côté du plan)
                               à partir de laquelle l'adsorption a lieu.
                               */
#define BORNE_DROITE_Bas 50.0
                            /* distance RÉDUITE (d'un seul côté du plan)
                               à partir de laquelle ON CONSIDÈRE que
                               l'adsorption fini.

                               On doit l'ajuster de sorte que le Bas
                               obtenu soit une valeur fiable.
                               */


double V_flat(double z, double theta, double eps, double sigma);
double integrand_Bas(double z, double T, double theta, double eps,
                     double sigma);
double trapeze_etape_N_Bas(double (*f)(double x, double T, double theta,
                           double eps, double sigma), double a, double b,
                           int N, double T, double theta, double eps,
```

```
                              double sigma);
double simpson_raffinee_Bas(double (*f)(double x, double T, double theta,
                             double eps, double sigma), double a, double b,
                             double T, double theta, double eps, double sigma);
double Bas_1plan(double T, double A, double theta, double eps,
                 double sigma, double borne_gauche,
                 double borne_droite);
double** matrice_Bas(double x_min, double x_max, int Nbre_x,
                double A, double theta, double eps,
                double sigma, double borne_gauche,
                double borne_droite);
void Ecrire_Bas(double** matriceBas, int Nbre_T, double A,
                double theta, double eps, double sigma);
void liberermatrice(double** matrice);
```

```
/*****************************************************************

  Programme: 1 plan graphite

  Ce programme calcule le second coefficient du viriel Bas pour
  l'adsorption d'un gaz (qui est considéré monoatomique) sur
  un SEUL plan de graphite d'aire A.

  L'usager entre les paramètres de Lennard-Jones sigma, eps,
  l'aire A du plan et la température.  On peut calculer Bas
  pour une seule température ou pour une série de températures
  comprises entre T_min et T_max.

  Appel:  1 plan graphite

*****************************************************************/

#include "traitement.h"

int main() {

    printf("Ce programme calcule le second coefficient du\n");
    printf("viriel pour l'adsorption sur 1 seul plan de graphite\n\n");

    printf("Entree des parametres de Lennard-Jones, de l'aire\n");
    printf("du plan de graphite et des temperatures ou on\n");
    printf("calcule le second coefficient du viriel Bas\n\n");

    printf("(Pour avoir la courbe universelle de Bas/A*sigma\n");
    printf("versus kT/Es, poser A = 1, sigma = 1\n");
    printf("epsilon = (5*k)/(6*Pi*theta) = 0.698047996\n");
    printf("Ici, Es = |minimum du potentiel|)\n\n");

    double eps;
    printf("epsilon (K) = ");
    scanf("%lf", &eps);

    double sigma;
    printf("sigma (Angstrom) = ");
    scanf("%lf", &sigma);

    double A;
    printf("Aire du plan (Angstrom^2) = ");
    scanf("%lf", &A);

    char SerieDeT;          /* variable disant si on
                               fait le calcul pour une série
                               de T.
                               */
    printf("\nVoulez-vous calculer Bas pour une serie de\n");
    printf("temperatures? (O/N)? ");
    cin >> SerieDeT;        /* ON N'A PAS PU UTILISER scanf()
                               ICI.  Il y a un BOGUE inexpliqué
                               quand on l'utilise!
                               */

    if ((SerieDeT == 'O') || (SerieDeT == 'o')) {
        double T_min, T_max;
        int Nbre_T;
        printf("Entrez la temperature minimale (K):\nT_min = ");
        scanf("%lf", &T_min);
        printf("Entrez la temperature maximale (K):\nT_max = ");
        scanf("%lf", &T_max);
        printf("Entrez le nombre de temperatures ou on calcule Bas:\nNbre_T = ");
        scanf("%d", &Nbre_T);
```

```
            /* Calculer la matriceBas contenant les Bas calculés
               aux différentes températures. */
            double** matriceBas = matrice_Bas(T_min, T_max, Nbre_T,
                    A, THETA, eps, sigma, BORNE_GAUCHE_Bas*sigma,
                    BORNE_DROITE_Bas*sigma);

            /* Imprimer la matriceBas dans un fichier. */
            Ecrire_Bas(matriceBas, Nbre_T, A, THETA, eps, sigma);

            /* libérer la mémoire de la matriceBas avant
               de quitter le bloc. */
            liberermatrice(matriceBas);
        }


        else {
            /* si SerieDeT != oui, on calcule le Bas pour
               la température demandée par l'usager. */
            double T;
            printf("T (K) = ");
            scanf("%lf", &T);

            double Bas = Bas_1plan(T, A, THETA, eps, sigma,
                BORNE_GAUCHE_Bas*sigma, BORNE_DROITE_Bas*sigma);

            printf("\nBas (Angstrom^3) = %.6lg\n", Bas);
        }

        return 0;
}
```

```
/*****************************************************************

  Fonction:  matrice_Bas()

  Cette fonction génère une matrice[2][Nbre_x] contenant
  Nbre_x valeurs de second coefficient du viriel Bas,
  pour chaque valeur de x.  Ici, x pourra être soit un vecteur
  contenant une série de températures ou un vecteur contenant
  une série de rayons R pour un nanotube simple ou multiple.
  (Pour un nanotube multiple, ce sera une série de rayons Ri où
  i est la ième paroi du nanotube).  Ou encore un vecteur
  contenant une série de séparations "d" entre des plans de
  graphite.

  Appel:  matriceBas = matrice_Bas(x_min, x_max, Nbre_x, A,
                   theta, eps, sigma, borne_gauche, borne_droite);


    matriceBas:  matrice contenant les Nbre_x valeurs
        x[i] et les valeurs Bas[i] correspondantes.
    x_min:  la valeur minimale de x (T par exemple).
    x_max:  la valeur maximale de x.
    Nbre_x:  le nombre de valeurs de x pour lesquelles
        on calcule les Bas correspondants.
    A:  l'aire du plan de graphite.
    theta: densité surfacique d'atomes de C formant le plan.
    eps:  constante d'énergie du potentiel de Lennard-Jones.
    sigma:  rayon caractéristique du potentiel de Lennard-Jones
        entre un atome de C et une molécule d'adsorbat.
    borne_gauche:  distance (d'un seul côté du plan) à partir
        de laquelle l'adsorption a lieu.
    borne_droite:  distance (d'un seul côté du plan) à partir
        de laquelle l'adsorption fini.

*****************************************************************/

#include "traitement.h"

double** matrice_Bas(double x_min, double x_max, int Nbre_x,
                double A, double theta, double eps,
                double sigma, double borne_gauche,
                double borne_droite) {

        /* espacement entre 2 valeurs x[i] */
    double delta_x = (fabs(x_max - x_min))/(Nbre_x - 1);

        /* formation de la matriceBas contenant les Bas */
    double** matriceBas = (double**) malloc(2 * sizeof(double*));
    if (matriceBas == NULL) {
        printf("\nErreur d'allocation de memoire\n");
        exit(1);
    }

    for (int i = 1; i <= 2; i++) {
            /* allouer la mémoire pour les 2 lignes de la
            matriceBas:  x et Bas. */
        matriceBas[i -1] = (double*) malloc(Nbre_x * sizeof(double));
        if (matriceBas[i -1] == NULL) {
            printf("\nErreur d'allocation de memoire\n");
            exit(1);
        }
    }


        /* remplir la matriceBas des valeurs de x et de Bas
```

```
        correspondantes. */
    for (i = 1; i <= Nbre_x; i++) {

        double x = x_min + (i - 1)*delta_x;
        double T = x;    /* dans ce cas ci, x == T. */
        double Bas = Bas_1plan(T, A, theta, eps, sigma,
            borne_gauche, borne_droite);

        matriceBas[1 -1][i -1] = x;
        matriceBas[2 -1][i -1] = Bas;
            /* Je préfère compter les indices à partir de 1, mais
            le C compte à partir de 0.  Je fais donc toujours une
            correction systématique de -1 aux indices des vecteurs
            et matrices.
            */
    }

    return matriceBas;

}
```

```
/*****************************************************************

  Fonction:  Bas_1plan()

  Cette fonction calcule le second coefficient du viriel Bas
  pour l'adsorption sur 1 SEUL plan de graphite, en calculant
  l'intégrale de l'integrand_Bas sur tout l'espace où il y a
  de l'adsorption.

  Appel:  Bas = Bas_1plan(T, A, theta, eps, sigma, borne_gauche,
                          borne_droite);

    Bas:  le second coefficient du viriel.
    A:  l'aire du plan de graphite.
    T:  température du gaz (en K).
    theta: densité surfacique d'atomes de C formant le plan.
    eps:  constante d'énergie du potentiel de Lennard-Jones.
    sigma:  rayon caractéristique du potentiel de Lennard-Jones
        entre un atome de C et une molécule d'adsorbat.
    borne_gauche:  distance (d'un seul côté du plan) à partir
        de laquelle l'adsorption a lieu.
    borne_droite:  distance (d'un seul côté du plan) à partir
        de laquelle l'adsorption fini.

*****************************************************************/

#include "traitement.h"

double Bas_1plan(double T, double A, double theta, double eps,
                 double sigma, double borne_gauche,
                 double borne_droite) {

    return 2*A*simpson_raffinee_Bas(&integrand_Bas, borne_gauche,
                                borne_droite, T, theta, eps, sigma);

}
```

```
/*****************************************************************

  Fonction: integrand_Bas()

  Cette fonction calcule l'intégrand (= exp(-V/kT) - 1) qui entre
  dans le calcul du second coefficient du viriel Bas.  Cet
  intégrand devient -1 lorsque l'on se situe très près de la
  paroi.

  Appel:  integrand = integrand_Bas(z, T, theta, eps, sigma);

    integrand:  = exp(-V/kT) - 1, entrant dans l'intégrale du Bas.
    z:  distance du plan où on calcule la valeur du potentiel (en
        Angstrom)
    T:  température du gaz (en K).
    theta: densité surfacique d'atomes de C formant le plan.
    eps:  constante d'énergie du potentiel de Lennard-Jones.
    sigma:  rayon caractéristique du potentiel de Lennard-Jones
        entre un atome de C et une molécule d'adsorbat.

*****************************************************************/

#include "traitement.h"

double integrand_Bas(double z, double T, double theta, double eps,
                     double sigma) {

        /* si on est dans la "distance d'exclusion" de la paroi,
        le potentiel tend vers l'infini et l'intégrand ===> -1. */
    if (fabs(z - 0.0) < DISTREDUITEMIN*sigma) {
        return -1.0;
    }

    else {
        return (exp((-1.0/(k*T))*V_flat(z, theta, eps, sigma)) - 1.0);
    }

}
```

```
/*************************************************************

  Fonction: V_flat()

  Cette fonction calcule le potentiel créé par un SEUL plan de
  graphite, à une distance z de celui-ci.

  Appel:  V = V_flat(z, theta, eps, sigma);

    V:  le potentiel dû au plan, à une distance z de celui-ci.
    z:  distance du plan où on calcule la valeur du potentiel (en
        Angstrom)
    T:  température du gaz (en K).
    theta: densité surfacique d'atomes de C formant le plan.
    eps:  constante d'énergie du potentiel de Lennard-Jones.
    sigma:  rayon caractéristique du potentiel de Lennard-Jones
        entre un atome de C et une molécule d'adsorbat.

*************************************************************/

#include "traitement.h"

double V_flat(double z, double theta, double eps, double sigma) {

    return 2.0*PI*theta*eps*sigma*sigma*((2.0/5.0)*pow(sigma/z, 10)
        - pow(sigma/z, 4));

}
```

```
/*****************************************************************

  Fonction: Ecrire_Bas()

  Cette fonction écrit dans un fichier "Bas_1plan_graphite.txt"
  les valeurs contenues dans la matriceBas pour les seconds
  coefficients du viriel Bas calculés aux différentes températures
  T[i].  Le fichier contient également les autres paramètres
  pour lesquels ces Bas ont été calculés (i.e. A, theta, eps,
  sigma).

  Appel:  Ecrire_Bas(matriceBas, Nbre_T, A, theta, eps, sigma);


    matriceBas:  matrice contenant les Nbre_T valeurs
        T[i] et les valeurs Bas[i] correspondantes.
    Nbre_T:  le nombre de valeurs de température pour lesquelles
        on calcule les Bas correspondants.
    A:  l'aire du plan de graphite.
    theta: densité surfacique d'atomes de C formant le plan.
    eps:  constante d'énergie du potentiel de Lennard-Jones.
    sigma:  rayon caractéristique du potentiel de Lennard-Jones
        entre un atome de C et une molécule d'adsorbat.

*****************************************************************/

#include "traitement.h"

void Ecrire_Bas(double** matriceBas, int Nbre_T, double A,
                double theta, double eps, double sigma) {

    FILE* fptr = fopen("Bas_1plan_graphite.txt", "w");
    if (fptr == NULL) {
        printf("\nErreur d'ouverture de fichier\n");
        exit(1);
    }


        /*  Mettre la valeur des paramètres physiques au début
        du fichier, avant de mettre les valeurs de Bas. */
    fprintf(fptr, "Fichier %cBas_1plan_graphite.txt%c donnant le second\n", 34, 34);
    fprintf(fptr, "coefficient du viriel Bas, pour un seul plan de graphite.\n\n\n");

    fprintf(fptr, "Aire du plan:\nA (angstrom^2) = %.8lg\n\n", A);
    fprintf(fptr, "Densite surfacique des atomes de C formant le plan:\n");
    fprintf(fptr, "theta (angstrom^-2) = %lf\n\n", theta);

    fprintf(fptr, "Valeurs des parametres du potentiel de Lennard-Jones:\n");
    fprintf(fptr, "epsilon (K) = %lf\t\tsigma (angstrom) = %lf\n\n", eps, sigma);

        /* Mettre la valeur de Bas[i] correspondant à chaque température
        T[i] dans le fichier.  (i.e. mettre les éléments de matriceBas
        dans le fichier, en faisant une validation de l'écriture dans le
        fichier pour chaque valeur de Bas ajoutée.
        */
    fprintf(fptr, "Nombre de valeurs de Bas calculees: %d\n\n\n", Nbre_T);
    fprintf(fptr, "Valeurs du second coefficient du viriel Bas pour chaque\n");
    fprintf(fptr, "temperature de la plage consideree:\n");

    fprintf(fptr, "T (K)\t\tBas (angstrom^3)\n");
    for (int i = 1; i <= Nbre_T; i++) {
        if (fprintf(fptr, "%-13.6lg      %-13.6lg\n", matriceBas[1 -1][i -1], matrice
Bas[2 -1][i -1]) == EOF) {
            printf("\nErreur d'ecriture dans le fichier\n");
            exit(1);
```

```
        }
    }

        /* Fermer le fichier avant de quitter la fonction. */
    fclose(fptr);
}
```

```
/*************************************************************************

  Fonction: simpson_raffinee_Bas()

  Cette fonction utilise la fonction trapeze_etape_N_Bas() pour calculer
  l'intégrale de f(x, T, theta, eps, sigma) par rapport à x sur
  l'intervalle [a, b].  L'erreur relative permise sur l'intégrale est
  de ERREUR_RELATIVE_Bas.

  Cette fonction calcule l'intégrale par améliorations successives.
  Elle calcule une valeur améliorée à partir de la valeur précédente
  (qui est moins exacte).  JMAX est le nombre maximal d'améliorations
  qu'on permet pour le calcul de l'intégrale finale.  Donc 2^(JMAX-1)
  est le nombre total (maximal) de sous-intervalles qu'on peut avoir
  pour le calcul de l'intégrale.

  utilisant l'équation (voir Numerical Recipes):
        S = (4/3)*S[2N] - (1/3)*S[N]
  on peut calculer l'intégrale de Simpson à partir d'intégrales "trapèze"
  sans complications supplémentaires.

  Appel: integrale = simpson_raffinee_Bas(f, a, b, T, theta, eps, sigma);

        f: f(x, T, theta, eps, sigma) la fonction à intégrer par rapport à x.
        a: borne gauche de l'intervalle d'intégration.
        b: borne droite de l'intervalle d'intégration.
        T, theta, eps, sigma:  variables gardées fixes lors de l'intégration.

*************************************************************************/

#include "traitement.h"

double simpson_raffinee_Bas(double (*f)(double x, double T, double theta,
                            double eps, double sigma), double a, double b,
                            double T, double theta, double eps, double sigma) {

    double integrale, old_integrale, integrale_trapeze,
           old_integrale_trapeze;
                        /* contient les integrales (actuelles et
                        anciennes) de Simpson et du trapèze. */

    old_integrale_trapeze = old_integrale = -1.0e30;
                        /* valeurs impossibles pour les vieilles
                        intégrales.  Ça permet de commencer la
                        boucle des "améliorations". */

    for (int j = 1; j <= JMAX; j++) {
            /* faire un nombre d'améliorations <= JMAX. */
        integrale_trapeze = trapeze_etape_N_Bas(f, a, b, j, T, theta, eps, sigma);
            /* calculer la jème approximation de l'intégrale de Simpson. */
        integrale = (4.0*integrale_trapeze - old_integrale_trapeze)/3.0;

        if (j > 5) {
                /* la condition j > 5 évite une (fausse) convergence
                trop hâtive. */
            if ((fabs(integrale - old_integrale)  < ERREUR_RELATIVE_Bas * fabs(old_in
tegrale))
                || (integrale == 0.0 && old_integrale == 0.0)) {
                    /* on a obtenu une valeur assez précise
                    de l'intégrale. */
                return integrale;
            }
        }

                /* les intégrales actuelles seront les "vieilles"
```

```
                intégrales de la prochaine approximation. */
            old_integrale = integrale;
            old_integrale_trapeze = integrale_trapeze;
    }

        /* Si on est rendu ici, c'est qu'on a pas réussi à converger.
        Ça peut être parce que le nombre d'étapes JMAX est trop petit
        (i.e. que le nombre d'améliorations maximal permis n'est pas
        assez grand pour atteindre la précision voulue). Ou que
        l'ERREUR_RELATIVE_Bas tolérée  est trop petite par rapport aux
        erreurs d'arrondis dûs à l'ordinateur (cause plus probable). */
    printf("\nL'erreur relative toleree sur l'integrale est trop\n");
    printf("petite pour avoir convergence.\n");
    exit(1);
    return 0.0; /* retourner une valeur double bidon car la fonction
                doit retourner quand même un double. */

}
```

```
/*************************************************************************

  Fonction: trapeze_etape_N_Bas()

    Cette fonction retourne la Nème valeur raffinée de l'intégrale
    de f(x, T, theta, eps, sigma) sur [a, b] à partir de la (N-1)ème
    valeur (i.e. à partir de sa valeur précédente).  L'intégration est
    faite par rapport à x, tandis que T, theta, eps et sigma sont gardées
    fixes.  La procédure a pour principe la règle du trapèze étendu
    (voir Numerical Recipes).

    Le premier appel calcule une intégrale grossière: un seul gros
    trapèze sur tout l'intervalle [a, b].  Les itérations suivantes
    N = 2, 3, ... rajoutent 2^(N-2) points de calculs au résultat
    précédent.

    Cette procédure a l'avantage de ne pas "jeter à la poubelle" le
    résultat précédent pour calculer une intégrale plus précise, mais
    d'utiliser le travail déjà fait.

  Appel: integrale = trapeze_etape_N_Bas(f, a, b, N, T, theta, eps, sigma);

        f: f(x, T, theta, eps, sigma) la fonction à intégrer par rapport à x.
        a: borne gauche de l'intervalle d'intégration.
        b: borne droite de l'intervalle d'intégration.
        N: indique qu'on est à la Nème étape de "raffinement".
        x: variable d'intégration.
        T, theta, eps, sigma:  variables gardées fixes lors de l'intégration.

*************************************************************************/

#include "traitement.h"

double trapeze_etape_N_Bas(double (*f)(double x, double T, double theta,
                           double eps, double sigma), double a, double b,
                           int N, double T, double theta, double eps,
                           double sigma) {

    double x, somme, delta;     /* somme est la somme de la nouvelle
                                   série de points à ajouter.
                                   delta est l'espace entre 2 de ces points.
                                   x est la position (abscisse) du point
                                   intermédiaire qu'on considère.
                                   */
    static double integrale;    /* contient présentement la (N-1)ème
                                   approximation de l'intégrale. (contiendra
                                   la Nème approximation à la fin de la
                                   procédure).
                                   */

    if (N == 1) {
            /* on calcule alors la première approximation très grossière
               donnée par un seul gros trapèze englobant tout l'intervalle
               [a, b]. */
        integrale = 0.5*(b-a)*((*f)(a, T, theta, eps, sigma) + (*f)(b, T, theta, eps,
 sigma));
        return integrale;
    }

    else {
            /* calculer la Nème approximation à partir de l'approximation
               précédente. */
        delta = (fabs(b - a))/pow(2.0, N-2);
        x = a + delta/2;    /* la position du premier point intermédiaire
                               à ajouter. */
```

```
            /* additionner les valeurs de la fonction f prises aux points
            intermédiaires x[j] = x + (j-1)*delta. j allant de 1 à 2^(N-2). */
        somme = 0.0;
        for (int j = 1; j <= ((int) pow(2.0, N-2)); j++) {
            somme = somme + (*f)(x, T, theta, eps, sigma);
            x = x + delta;  /* passer au prochain point intermédiaire. */
        }

            /* calculer la Nème approximation de l'intégrale à partir
            de la (N-1)ème approximation. */
        integrale = 0.5*(integrale + delta*somme);
        return integrale;
    }
}
```

```
/*************************************************************************

  Fonction: liberermatrice()

  Cette fonction libère la mémoire d'une matrice à 2 dimensions contenant
  les points (X, Y)

  Appel: liberermatrice(matrice);

  matrice: la matrice[2][nbrePts dans les vecteurs X et Y] à libérer

*************************************************************************/

#include "traitement.h"

void liberermatrice(double** matrice) {

        /* libérer les 2 lignes */
    free(matrice[1 -1]);
    free(matrice[2 -1]);

        /* libérer la matrice contenant l'adresse des 2 lignes déjà
        libérées */
    free(matrice);

}
```

```
/*****************************************************************

  Fichier d'en-tête traitement.h

  Contient les librairies, les constantes et les déclaration
  (i.e les prototypes) de fonctions.

*****************************************************************/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <iostream.h>       /* Je dois utiliser la fonction
                               "cout << " à cause du bogue avec
                               scanf().
                               */

#define PI 3.14159265358979323846264338327795
#define ERREUR_RELATIVE_Bas 1e-5
                            /* erreur (relative) tolérée dans la fonction
                            simpson_raffinee_Bas().  C'est l'incertitude
                            relative de l'intégrale calculant Bas.
                            (i.e. que c'est un % d'incertitude sur
                            l'intégrale).  */
#define JMAX 20             /* est le nombre de raffinages MAXIMUM qu'on
                            fait sur l'intégrale de départ.
                            2^(JMAX-1) est le nombre total (maximum)
                            de sous-intervalles pour l'intégration. */
#define THETA 0.38          /* Angstrom^-2
                            Densité surfacique des atomes sur la paroi
                            du nanotube (ou du plan de graphite). */

#define k 1.0               /* constante de Boltzmann (= 1 lorsque
                            les énergies sont exprimées en K).
                            */

#define DISTREDUITEMIN 0.01
                            /* distance (réduite) minimale de la paroi.
                            À l'intérieur de cette distance d'exclusion,
                            le potentiel est considéré comme infini et
                            integrand_Bas = exp(-V/kT) - 1 ===> -1.

                            Cette distance est choisie de sorte que
                            integrand_Bas soit très près de -1.
                            */


double V_flat(double z, double theta, double eps, double sigma);
double integrand_Bas(double z, double T, double d, double theta,
                     double eps, double sigma);
double trapeze_etape_N_Bas(double (*f)(double x, double T, double d, double theta,
                           double eps, double sigma), double a, double b,
                           int N, double T, double d, double theta, double eps,
                           double sigma);
double simpson_raffinee_Bas(double (*f)(double x, double T, double d, double theta,
                            double eps, double sigma), double a, double b,
                            double T, double d, double theta, double eps,
                            double sigma);
double Bas_2plans(double T, double d, double A, double theta, double eps,
                  double sigma, double borne_gauche,
                  double borne_droite);
double** matrice_Bas(double x_min, double x_max, int Nbre_x,
                 double y, double A, double theta, double eps,
                 double sigma, double borne_gauche,
                 double borne_droite, int T_ou_d);
```

```
void Ecrire_Bas(double** matriceBas, int Nbre_x, double y,
                double A, double theta, double eps, double sigma,
                int T_ou_d);
void liberermatrice(double** matrice);
```

```
/*****************************************************************

  Programme: 2 plans graphite

  Ce programme calcule le second coefficient du viriel Bas pour
  l'adsorption d'un gaz (qui est considéré monoatomique) ENTRE
  2 plans de graphite parallèles d'aire A (chacun), séparés
  d'une distance d.

  L'usager entre les paramètres de Lennard-Jones sigma, eps,
  l'aire A des plans, la séparation d et la température.  On peut
  calculer Bas pour une seule température ou pour une série de
  températures comprises entre T_min et T_max.  Aussi, on peut
  calculer Bas pour une seule distance ou pour une série de
  distances comprises entre d_min et d_max.

  Appel:  2 plans graphite

*****************************************************************/

#include "traitement.h"

int main() {

    printf("Ce programme calcule le second coefficient du\n");
    printf("viriel pour l'adsorption entre 2 plans de graphite\n");
    printf("paralleles\n\n");

    printf("Entree des parametres de Lennard-Jones, de l'aire\n");
    printf("des plans de graphite, des separations entre les plans\n");
    printf("et des temperatures ou on calcule le second\n");
    printf("coefficient du viriel Bas\n\n");

    printf("(Pour avoir la courbe universelle de Bas/A*sigma\n");
    printf("versus kT/Es' ou versus d/sigma, poser A = 1, sigma = 1\n");
    printf("epsilon = k/(Pi*theta) = 0.837657595\n");
    printf("Ici, Es' = Pi*theta*epsilon*sigma^2)\n\n");

    double eps;
    printf("epsilon (K) = ");
    scanf("%lf", &eps);

    double sigma;
    printf("sigma (Angstrom) = ");
    scanf("%lf", &sigma);

    double A;
    printf("Aire des plans (Angstrom^2) = ");
    scanf("%lf", &A);

    char SerieDeT, SerieDed;    /* variables disant si on
                                   fait le calcul pour une série
                                   de T ou une série de d.
                                   */
    printf("\nVoulez-vous calculer Bas pour une serie de\n");
    printf("temperatures? (O/N)? ");
    cin >> SerieDeT;          /* ON N'A PAS PU UTILISER scanf()
                                 ICI.  Il y a un BOGUE inexpliqué
                                 quand on l'utilise!
                                 */

    if ((SerieDeT == 'O') || (SerieDeT == 'o')) {
        double T_min, T_max, d;
        int Nbre_T;
        printf("Entrez la separation entre les 2 plans (Angstrom):\nd = ");
```

```
        scanf("%lf", &d);
        printf("Entrez la temperature minimale (K):\nT_min = ");
        scanf("%lf", &T_min);
        printf("Entrez la temperature maximale (K):\nT_max = ");
        scanf("%lf", &T_max);
        printf("Entrez le nombre de temperatures ou on calcule Bas:\nNbre_T = ");
        scanf("%d", &Nbre_T);

            /* Calculer la matriceBas contenant les Bas calculés
            aux différentes températures. */
        double** matriceBas = matrice_Bas(T_min, T_max, Nbre_T,
                d, A, THETA, eps, sigma, 0.0,
                d/2, 1);

            /* Imprimer la matriceBas dans un fichier. */
        Ecrire_Bas(matriceBas, Nbre_T, d, A, THETA, eps, sigma, 1);

            /* libérer la mémoire de la matriceBas avant
            de quitter le bloc. */
        liberermatrice(matriceBas);
    }


    else {
        printf("\nVoulez-vous calculer Bas pour une serie de\n");
        printf("separations entre les plans? (O/N)? ");
        cin >> SerieDed;    /* ON N'A PAS PU UTILISER scanf()
                            ICI.  Il y a un BOGUE inexpliqué
                            quand on l'utilise!
                            */

        if ((SerieDed == 'O') || (SerieDed == 'o')) {
            double d_min, d_max, T;
            int Nbre_d;
            printf("Entrez la temperature de l'adsorbat (K):\nT = ");
            scanf("%lf", &T);
            printf("Entrez la separation minimale (angstrom):\nd_min = ");
            scanf("%lf", &d_min);
            printf("Entrez la separation maximale (angstrom):\nd_max = ");
            scanf("%lf", &d_max);
            printf("Entrez le nombre de separations ou on calcule Bas:\nNbre_d = ");
            scanf("%d", &Nbre_d);

                /* Calculer la matriceBas contenant les Bas calculés
                aux différentes séparations entre les plans. */
            double** matriceBas = matrice_Bas(d_min, d_max, Nbre_d,
                    T, A, THETA, eps, sigma, 0.0,
                    0.0, 0);    /* ici, le paramètre borne_droite doit être pris comm
e
                                une valeur bidon, car ici c'est en réalité une valeur

                                variable (= d/2) déterminée à l'intérieur de la fonct
ion
                                matrice_Bas(). */

                /* Imprimer la matriceBas dans un fichier. */
            Ecrire_Bas(matriceBas, Nbre_d, T, A, THETA, eps, sigma, 0);

                /* libérer la mémoire de la matriceBas avant
                de quitter le bloc. */
            liberermatrice(matriceBas);
        }


        else {
```

```
                /* si SerieDeT != oui et SerieDed != oui, on calcule le Bas pour
                la température et la séparation demandées par l'usager. */
            double T, d;
            printf("Entrez la temperature de l'adsorbat (K):\nT = ");
            scanf("%lf", &T);
            printf("Entrez la separation entre les 2 plans (Angstrom):\nd = ");
            scanf("%lf", &d);

            double Bas = Bas_2plans(T, d, A, THETA, eps, sigma, 0.0, d/2);
            printf("\nBas (Angstrom^3) = %.6lg\n", Bas);
        }
    }

    return 0;
}
```

```
/*****************************************************************

  Fonction:  matrice_Bas()

  Cette fonction génère une matrice[2][Nbre_x] contenant
  Nbre_x valeurs de second coefficient du viriel Bas,
  pour chaque valeur de x.  Ici, x pourra être soit un vecteur
  contenant une série de températures ou un vecteur contenant
  une série de rayons R pour un nanotube simple ou multiple.
  (Pour un nanotube multiple, ce sera une série de rayons R[i] où
  i est la ième paroi du nanotube).  Ou encore un vecteur
  contenant une série de séparations "d" entre des plans de
  graphite.

  Appel:  matriceBas = matrice_Bas(x_min, x_max, Nbre_x, y, A,
                   theta, eps, sigma, borne_gauche, borne_droite,
                   T_ou_d);


    matriceBas:  matrice contenant les Nbre_x valeurs
        x[i] et les valeurs Bas[i] correspondantes.
    x_min:  la valeur minimale de x (T par exemple).
    x_max:  la valeur maximale de x.
    Nbre_x:  le nombre de valeurs de x pour lesquelles
        on calcule les Bas correspondants.
    y: l'autre paramètre à part x, qui est nécessaire pour
        déterminer Bas.  Ça peut être T ou d.  C'est le
        paramètre gardé fixe lors du calcul de Bas pour une
        série de x.
    A:  l'aire de chaque plan de graphite (Angstrom^2)
    theta: densité surfacique d'atomes de C formant le plan.
    eps:  constante d'énergie du potentiel de Lennard-Jones.
    sigma:  rayon caractéristique du potentiel de Lennard-Jones
        entre un atome de C et une molécule d'adsorbat.
    borne_gauche:  distance (>= 0), mesurée du centre des 2 plans,
        à partir de laquelle l'adsorption a lieu.
    borne_droite:  distance (>= 0), mesurée du centre des 2 plans,
        à partir de laquelle l'adsorption fini.
    T_ou_d:  variable (= 1 pour T, 0 pour d) disant si x est un
        vecteur de températures ou de séparations d.

*****************************************************************/

#include "traitement.h"

double** matrice_Bas(double x_min, double x_max, int Nbre_x,
                double y, double A, double theta, double eps,
                double sigma, double borne_gauche,
                double borne_droite, int T_ou_d) {

        /* espacement entre 2 valeurs x[i] */
    double delta_x = (fabs(x_max - x_min))/(Nbre_x - 1);

        /* formation de la matriceBas contenant les Bas */
    double** matriceBas = (double**) malloc(2 * sizeof(double*));
    if (matriceBas == NULL) {
        printf("\nErreur d'allocation de memoire\n");
        exit(1);
    }

    for (int i = 1; i <= 2; i++) {
            /* allouer la mémoire pour les 2 lignes de la
            matriceBas:  x et Bas. */
        matriceBas[i -1] = (double*) malloc(Nbre_x * sizeof(double));
        if (matriceBas[i -1] == NULL) {
```

```
            printf("\nErreur d'allocation de memoire\n");
            exit(1);
        }
    }


        /* remplir la matriceBas des valeurs de x et de Bas
        correspondantes. */
    for (i = 1; i <= Nbre_x; i++) {
        double x = x_min + (i - 1)*delta_x;
        double Bas;
        if (T_ou_d == 0) {
            Bas = Bas_2plans(y, x, A, theta, eps, sigma,
                borne_gauche, x/2);         /* ici borne_droite n'est
                                               pas fixe:  il vaut d/2.
                                               */
        }
        else {
            Bas = Bas_2plans(x, y, A, theta, eps, sigma,
                borne_gauche, borne_droite);
        }

        matriceBas[1 -1][i -1] = x;
        matriceBas[2 -1][i -1] = Bas;
            /* Je préfère compter les indices à partir de 1, mais
            le C compte à partir de 0.  Je fais donc toujours une
            correction systématique de -1 aux indices des vecteurs
            et matrices.
            */
    }

    return matriceBas;

}
```

```
/*****************************************************************

  Fonction:  Bas_2plans()

  Cette fonction calcule le second coefficient du viriel Bas
  pour l'adsorption dûe à 2 plans de graphite parallèles séparés
  d'une distance d, en calculant l'intégrale de l'integrand_Bas
  sur tout l'espace où il y a de l'adsorption.

  Appel:  Bas = Bas_2plans(T, d, A, theta, eps, sigma, borne_gauche,
                           borne_droite);

    Bas:  le second coefficient du viriel.
    A:  l'aire de chaque plan de graphite (Angstrom^2)
    T:  température du gaz (en K).
    d: la distance séparant les 2 plans de graphite (en Angstrom).
    theta: densité surfacique d'atomes de C formant le plan.
    eps:  constante d'énergie du potentiel de Lennard-Jones.
    sigma:  rayon caractéristique du potentiel de Lennard-Jones
        entre un atome de C et une molécule d'adsorbat.
    borne_gauche:  distance (>= 0), mesurée du centre des 2 plans,
        à partir de laquelle l'adsorption a lieu.
    borne_droite:  distance (>= 0), mesurée du centre des 2 plans,
        à partir de laquelle l'adsorption fini.

*****************************************************************/

#include "traitement.h"

double Bas_2plans(double T, double d, double A, double theta, double eps,
                  double sigma, double borne_gauche,
                  double borne_droite) {

    return 2*A*simpson_raffinee_Bas(&integrand_Bas, borne_gauche,
                                    borne_droite, T, d, theta, eps, sigma);

}
```

```
/*****************************************************************

  Fonction: integrand_Bas()

  Cette fonction calcule l'intégrand (= exp(-V/kT) - 1) qui entre
  dans le calcul du second coefficient du viriel Bas.  Cet
  intégrand devient -1 lorsque l'on se situe très près de la
  paroi.

  Appel:  integrand = integrand_Bas(z, T, d, theta, eps, sigma);

    integrand:  = exp(-V/kT) - 1, entrant dans l'intégrale du Bas.
    z:  distance par rapport au centre des plans, c'est où on
    calcule la valeur du potentiel (en Angstrom).
    T:  température du gaz (en K).
    d: distance séparant les 2 plans (en Angstrom).
    theta: densité surfacique d'atomes de C formant le plan.
    eps:  constante d'énergie du potentiel de Lennard-Jones.
    sigma:  rayon caractéristique du potentiel de Lennard-Jones
        entre un atome de C et une molécule d'adsorbat.

*****************************************************************/

#include "traitement.h"

double integrand_Bas(double z, double T, double d, double theta,
                     double eps, double sigma) {

        /* si on est dans la "distance d'exclusion" de la paroi,
        le potentiel tend vers l'infini et l'intégrand ===> -1. */
    if (fabs(z - (d/2)) < DISTREDUITEMIN*sigma) {
        return -1.0;
    }

    else {
        return (exp((-1.0/(k*T))*(V_flat(z + d/2, theta, eps, sigma) +
            V_flat(z - d/2, theta, eps, sigma))) - 1.0);
    }

}
```

```
/*****************************************************************

  Fonction: V_flat()

  Cette fonction calcule le potentiel créé par un SEUL plan de
  graphite, à une distance z de celui-ci.

  Appel:  V = V_flat(z, theta, eps, sigma);

    V:  le potentiel dû au plan, à une distance z de celui-ci.
    z:  distance du plan où on calcule la valeur du potentiel (en
        Angstrom)
    T:  température du gaz (en K).
    theta: densité surfacique d'atomes de C formant le plan.
    eps:  constante d'énergie du potentiel de Lennard-Jones.
    sigma:  rayon caractéristique du potentiel de Lennard-Jones
        entre un atome de C et une molécule d'adsorbat.

*****************************************************************/

#include "traitement.h"

double V_flat(double z, double theta, double eps, double sigma) {

    return 2.0*PI*theta*eps*sigma*sigma*((2.0/5.0)*pow(sigma/z, 10)
        - pow(sigma/z, 4));

}
```

```
/*************************************************************

  Fonction: Ecrire_Bas()

  Cette fonction écrit dans un fichier "Bas_2plans_graphite_T.txt"
  ou "Bas_2plans_graphite_d.txt" les valeurs contenues dans la
  matriceBas pour les seconds coefficients du viriel Bas
  calculés aux différentes valeurs de x (i.e T ou d).  Le fichier
  contient également les autres paramètres pour lesquels ces
  Bas ont été calculés (i.e. A, theta, eps, sigma, d ou T).

  Appel:  Ecrire_Bas(matriceBas, Nbre_x, y, A, theta, eps,
                  sigma, T_ou_d);


    matriceBas:  matrice contenant les Nbre_x valeurs
        x[i] et les valeurs Bas[i] correspondantes.
    Nbre_x:  le nombre de valeurs de x pour lesquelles
        on calcule les Bas correspondants.
    y: l'autre paramètre à part x, qui est nécessaire pour
        déterminer Bas.  Ça peut être T ou d.  C'est le
        paramètre gardé fixe lors du calcul de Bas pour une
        série de x.
    A:  l'aire de chaque plan de graphite (Angstrom^2)
    theta: densité surfacique d'atomes de C formant le plan.
    eps:  constante d'énergie du potentiel de Lennard-Jones.
    sigma:  rayon caractéristique du potentiel de Lennard-Jones
        entre un atome de C et une molécule d'adsorbat.
    T_ou_d:  variable (= 1 pour T, 0 pour d) disant si x est un
        vecteur de températures ou de séparations d.

*************************************************************/

#include "traitement.h"

void Ecrire_Bas(double** matriceBas, int Nbre_x, double y,
                double A, double theta, double eps, double sigma,
                int T_ou_d) {
    FILE* fptr;
    char* file_name;
    if (T_ou_d == 0) {
        file_name = "Bas_2plans_graphite_d.txt";
        fptr = fopen(file_name, "w");
    }
    else {
        file_name = "Bas_2plans_graphite_T.txt";
        fptr = fopen(file_name, "w");
    }
    if (fptr == NULL) {
        printf("\nErreur d'ouverture de fichier\n");
        exit(1);
    }


        /*  Mettre la valeur des paramètres physiques au début
        du fichier, avant de mettre les valeurs de Bas. */
    fprintf(fptr, "Fichier %c%s%c donnant le second\n", 34, file_name, 34);
    fprintf(fptr, "coefficient du viriel Bas, pour 2 plans de graphite paralleles.\n\
n\n");

    fprintf(fptr, "Aire de chaque plan:\nA (angstrom^2) = %.8lg\n\n", A);
    if (T_ou_d == 0) {
        fprintf(fptr, "Temperature de l'adsorbat:\nT (K) = %lf\n", y);
    }
    else {
```

```
        fprintf(fptr, "Distance entre les plans:\nd (angstrom) = %.8lf\n", y);
    }
    fprintf(fptr, "\nDensite surfacique des atomes de C formant le plan:\n");
    fprintf(fptr, "theta (angstrom^-2) = %lf\n\n", theta);

    fprintf(fptr, "Valeurs des parametres du potentiel de Lennard-Jones:\n");
    fprintf(fptr, "epsilon (K) = %lf\t\tsigma (angstrom) = %lf\n\n", eps, sigma);

        /* Mettre la valeur de Bas[i] correspondant à chaque valeur
        x[i] dans le fichier.  (i.e. mettre les éléments de matriceBas
        dans le fichier, en faisant une validation de l'écriture dans le
        fichier pour chaque valeur de Bas ajoutée.
        */
    fprintf(fptr, "Nombre de valeurs de Bas calculees: %d\n\n\n", Nbre_x);
    fprintf(fptr, "Valeurs du second coefficient du viriel Bas pour chaque\n");
    if (T_ou_d == 0) {
        fprintf(fptr, "separation entre les plans de la plage consideree:\n");
        fprintf(fptr, "d (angstrom)\tBas (angstrom^3)\n");
    }
    else {
        fprintf(fptr, "temperature de la plage consideree:\n");
        fprintf(fptr, "T (K)\t\tBas (angstrom^3)\n");
    }

    for (int i = 1; i <= Nbre_x; i++) {
        if (fprintf(fptr, "%-13.6lg      %-13.6lg\n", matriceBas[1 -1][i -1], matrice
Bas[2 -1][i -1]) == EOF) {
            printf("\nErreur d'ecriture dans le fichier\n");
            exit(1);
        }
    }

        /* Fermer le fichier avant de quitter la fonction. */
    fclose(fptr);
}
```

```
/*************************************************************************

  Fonction: simpson_raffinee_Bas()

  Cette fonction utilise la fonction trapeze_etape_N_Bas() pour calculer
  l'intégrale de f(x, T, d, theta, eps, sigma) par rapport à x sur
  l'intervalle [a, b].  L'erreur relative permise sur l'intégrale est
  de ERREUR_RELATIVE_Bas.

  Cette fonction calcule l'intégrale par améliorations successives.
  Elle calcule une valeur améliorée à partir de la valeur précédente
  (qui est moins exacte).  JMAX est le nombre maximal d'améliorations
  qu'on permet pour le calcul de l'intégrale finale.  Donc 2^(JMAX-1)
  est le nombre total (maximal) de sous-intervalles qu'on peut avoir
  pour le calcul de l'intégrale.

  utilisant l'équation (voir Numerical Recipes):
        S = (4/3)*S[2N] - (1/3)*S[N]
  on peut calculer l'intégrale de Simpson à partir d'intégrales "trapèze"
  sans complications supplémentaires.

  Appel: integrale = simpson_raffinee_Bas(f, a, b, T, d, theta, eps, sigma);

        f: f(x, T, d, theta, eps, sigma) la fonction à intégrer par rapport à x.
        a: borne gauche de l'intervalle d'intégration.
        b: borne droite de l'intervalle d'intégration.
        T, d, theta, eps, sigma:  variables gardées fixes lors de l'intégration.

*************************************************************************/

#include "traitement.h"

double simpson_raffinee_Bas(double (*f)(double x, double T, double d, double theta,
                            double eps, double sigma), double a, double b,
                            double T, double d, double theta, double eps,
                            double sigma) {

    double integrale, old_integrale, integrale_trapeze,
           old_integrale_trapeze;
                        /* contient les integrales (actuelles et
                        anciennes) de Simpson et du trapèze. */

    old_integrale_trapeze = old_integrale = -1.0e30;
                        /* valeurs impossibles pour les vieilles
                        intégrales.  Ça permet de commencer la
                        boucle des "améliorations". */

    for (int j = 1; j <= JMAX; j++) {
            /* faire un nombre d'améliorations <= JMAX. */
        integrale_trapeze = trapeze_etape_N_Bas(f, a, b, j, T, d, theta, eps, sigma);
            /* calculer la jème approximation de l'intégrale de Simpson. */
        integrale = (4.0*integrale_trapeze - old_integrale_trapeze)/3.0;

        if (j > 5) {
                /* la condition j > 5 évite une (fausse) convergence
                trop hâtive. */
            if ((fabs(integrale - old_integrale)  < ERREUR_RELATIVE_Bas * fabs(old_in
tegrale))
                || (integrale == 0.0 && old_integrale == 0.0)) {
                    /* on a obtenu une valeur assez précise
                    de l'intégrale. */
                return integrale;
            }
        }
```

```
                /* les intégrales actuelles seront les "vieilles"
                intégrales de la prochaine approximation. */
            old_integrale = integrale;
            old_integrale_trapeze = integrale_trapeze;
    }

        /* Si on est rendu ici, c'est qu'on a pas réussi à converger.
        Ça peut être parce que le nombre d'étapes JMAX est trop petit
        (i.e. que le nombre d'améliorations maximal permis n'est pas
        assez grand pour atteindre la précision voulue). Ou que
        l'ERREUR_RELATIVE_Bas tolérée  est trop petite par rapport aux
        erreurs d'arrondis dûs à l'ordinateur (cause plus probable). */
    printf("\nL'erreur relative toleree sur l'integrale est trop\n");
    printf("petite pour avoir convergence.\n");
    exit(1);
    return 0.0; /* retourner une valeur double bidon car la fonction
                doit retourner quand même un double. */

}
```

```
/*************************************************************************

  Fonction: trapeze_etape_N_Bas()

    Cette fonction retourne la Nème valeur raffinée de l'intégrale
    de f(x, T, d, theta, eps, sigma) sur [a, b] à partir de la (N-1)ème
    valeur (i.e. à partir de sa valeur précédente).  L'intégration est
    faite par rapport à x, tandis que T, d, theta, eps et sigma sont
    gardées fixes.  La procédure a pour principe la règle du trapèze
    étendu (voir Numerical Recipes).

    Le premier appel calcule une intégrale grossière: un seul gros
    trapèze sur tout l'intervalle [a, b].  Les itérations suivantes
    N = 2, 3, ... rajoutent 2^(N-2) points de calculs au résultat
    précédent.

    Cette procédure a l'avantage de ne pas "jeter à la poubelle" le
    résultat précédent pour calculer une intégrale plus précise, mais
    d'utiliser le travail déjà fait.

  Appel: integrale = trapeze_etape_N_Bas(f, a, b, N, T, d, theta, eps, sigma);

        f: f(x, T, d, theta, eps, sigma) la fonction à intégrer par rapport à x.
        a: borne gauche de l'intervalle d'intégration.
        b: borne droite de l'intervalle d'intégration.
        N: indique qu'on est à la Nème étape de "raffinement".
        x: variable d'intégration.
        T, d, theta, eps, sigma:  variables gardées fixes lors de l'intégration.

*************************************************************************/

#include "traitement.h"

double trapeze_etape_N_Bas(double (*f)(double x, double T, double d, double theta,
                           double eps, double sigma), double a, double b,
                           int N, double T, double d, double theta, double eps,
                           double sigma) {

    double x, somme, delta;     /* somme est la somme de la nouvelle
                                   série de points à ajouter.
                                   delta est l'espace entre 2 de ces points.
                                   x est la position (abscisse) du point
                                   intermédiaire qu'on considère.
                                   */
    static double integrale;    /* contient présentement la (N-1)ème
                                   approximation de l'intégrale. (contiendra
                                   la Nème approximation à la fin de la
                                   procédure).
                                   */

    if (N == 1) {
            /* on calcule alors la première approximation très grossière
            donnée par un seul gros trapèze englobant tout l'intervalle
            [a, b]. */
        integrale = 0.5*(b-a)*((*f)(a, T, d, theta, eps, sigma) + (*f)(b, T, d, theta
, eps, sigma));
        return integrale;
    }

    else {
            /* calculer la Nème approximation à partir de l'approximation
            précédente. */
        delta = (fabs(b - a))/pow(2.0, N-2);
        x = a + delta/2;    /* la position du premier point intermédiaire
                               à ajouter. */
```

```
            /* additionner les valeurs de la fonction f prises aux points
               intermédiaires x[j] = x + (j-1)*delta. j allant de 1 à 2^(N-2). */
            somme = 0.0;
            for (int j = 1; j <= ((int) pow(2.0, N-2)); j++) {
                somme = somme + (*f)(x, T, d, theta, eps, sigma);
                x = x + delta;  /* passer au prochain point intermédiaire. */
            }

            /* calculer la Nème approximation de l'intégrale à partir
               de la (N-1)ème approximation. */
            integrale = 0.5*(integrale + delta*somme);
            return integrale;
        }
}
```

```
/*****************************************************************************

  Fonction: liberermatrice()

  Cette fonction libère la mémoire d'une matrice à 2 dimensions contenant
  les points (X, Y)

  Appel: liberermatrice(matrice);

  matrice: la matrice[2][nbrePts dans les vecteurs X et Y] à libérer

*****************************************************************************/

#include "traitement.h"

void liberermatrice(double** matrice) {

        /* libérer les 2 lignes */
    free(matrice[1 -1]);
    free(matrice[2 -1]);

        /* libérer la matrice contenant l'adresse des 2 lignes déjà
        libérées */
    free(matrice);

}
```

```
/*****************************************************************

  Fichier d'en-tête traitement.h

  Contient les librairies, les constantes et les déclarations
  (i.e les prototypes) de fonctions.

*****************************************************************/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <iostream.h>       /* Je dois utiliser la fonction
                               "cin >> " à cause du bogue avec
                               scanf() lorsqu'on veut lire un
                               char sur le clavier.
                               */

#define PI 3.14159265358979323846264338327795
#define ERREUR_RELATIVE_Bas 1e-5
                            /* erreur (relative) tolérée dans la fonction
                            simpson_raffinee_Bas().  C'est l'incertitude
                            relative de l'intégrale calculant Bas.
                            (i.e. que c'est un % d'incertitude sur
                            l'intégrale).  */
#define ERREUR_RELATIVE_Mn 1e-6
                            /* erreur (relative) tolérée dans la fonction
                            simpson_raffinee_Mn().  C'est l'incertitude
                            relative de l'intégrale calculant Mn(x).
                            (i.e. que c'est un % d'incertitude sur
                            l'intégrale).  */
#define JMAX 30             /* est le nombre de raffinages MAXIMUM qu'on
                            fait sur l'intégrale de départ.
                            2^(JMAX-1) est le nombre total (maximum)
                            de sous-intervalles pour l'intégration.

                            Cette valeur est GARDÉE LA MÊME pour
                            l'intégrale calculant Mn(x) et celle calculant
                            Bas.*/
#define THETA 0.38          /* Angstrom^-2
                            Densité surfacique des atomes sur la paroi
                            du nanotube (ou du plan de graphite). */

#define k 1.0               /* constante de Boltzmann (= 1 lorsque
                            les énergies sont exprimées en K).
                            */

#define DISTREDUITEMIN 0.01
                            /* distance (réduite) minimale de la paroi.
                            À l'intérieur de cette distance d'exclusion,
                            le potentiel est considéré comme infini et
                            exp(-V/kT) - 1 ===> -1.
                            */

#define NBRE_DE_SIGMA_PASSE_Rmax 50
                            /* La distance (mesurées en sigma)
                            dépassée la paroi externe du nanotube
                            où on considère que l'adsorption fini
                            */


    /* fonctions servant au calcul de V(r, Rtubes[]), le
    potentiel du nanotube à parois multiples. */
```

```
double f(double phi, double x, int n);
double M(double x, int n);
double trapeze_etape_N_Mn(double (*f)(double x, double y, int n),
                          double a, double b, int N, double y, int n);
double simpson_raffinee_Mn(double (*f)(double x, double y, int n),
                           double a, double b, double y, int n);
double potentiel1wall(double r, double R, double theta,
                      double eps, double sigma);
double potentielMultiwall(double r, double* Rtubes, int NbreParois,
                          double theta, double eps, double sigma);

    /* fonctions servant à la partie principale du programme:
    le calcul de Bas. */
double integrand_Bas(double r, double T, double* Rtubes, int NbreParois,
                     double theta, double eps, double sigma);
double trapeze_etape_N_Bas(double (*f)(double x, double T,
                           double* Rtubes, int NbreParois,
                           double theta, double eps, double sigma),
                           double a, double b, int N, double T,
                           double* Rtubes, int NbreParois,
                           double theta, double eps, double sigma);
double simpson_raffinee_Bas(double (*f)(double x, double T, double* Rtubes,
                            int NbreParois, double theta, double eps,
                            double sigma), double a, double b,
                            double T, double* Rtubes, int NbreParois,
                            double theta, double eps, double sigma);
double Bas_Nanotube_multiwall(double T, double* Rtubes,
                int NbreParois, double L, double theta,
                double eps, double sigma, double borne_gauche,
                double borne_droite);
double R_externe(double* Rtubes, int NbreParois);
double** matrice_Bas(double x_min, double x_max, int Nbre_x,
                double T_ou_rien, double* Rtubes, int NbreParois,
                double L, double theta, double eps, double sigma,
                int T_ou_R, int flag_ouvert);
void Ecrire_Bas(double** matriceBas, int Nbre_x, double T_ou_rien,
                double* Rtubes, int NbreParois, double L,
                double theta, double eps, double sigma,
                int T_ou_R, int flag_ouvert);
void liberermatrice(double** matrice);
```

```
/*****************************************************************

  Programme: Nanotube multiwall

  Ce programme calcule le second coefficient du viriel Bas pour
  l'adsorption d'un gaz (qui est considéré monoatomique) sur un
  nanotube de longueur L à parois multiples, possédant NbreParois
  parois, et de rayons Rtubes[i] (i = 1 à NbreParois).  Le
  nanotube peut avoir les extrémités OUVERTES ou FERMÉES.

  L'usager commence par dire si le nanotube est OUVERT ou FERMÉ
  aux extrémités.  Ensuite, il entre les paramètres de
  Lennard-Jones sigma, eps, la longueur L du nanotube, les rayons
  Rtubes[i] et la température.  On peut calculer Bas pour une seule
  température ou pour une série de températures comprises entre
  T_min et T_max.  Aussi, on peut calculer Bas pour un seul rayon
  Rtubes[i] de la ième paroi, ou pour une série de rayons compris
  entre R_min et R_max.

  Appel:  Nanotube multiwall

*****************************************************************/

/*****************************************************************
    NOTE IMPORTANTE SUR L'INDIÇAGE DES VECTEURS ET MATRICES:

    Je préfère compter les indices à partir de 1, mais
    le C compte à partir de 0.  Je fais donc toujours une
    correction systématique de -1 aux indices des vecteurs
    et matrices.

*****************************************************************/

#include "traitement.h"

int main() {

    printf("Ce programme calcule le second coefficient du\n");
    printf("viriel pour l'adsorption sur un nanotube de carbone\n");
    printf("OUVERT ou FERMER a parois multiples\n\n");

    char input_flag_ouvert;
    int flag_ouvert = 1;
    printf("Le nanotube est-il ouvert ou fermer? (ouvert = o; fermer = f): ");
    cin >> input_flag_ouvert;
    if ( (input_flag_ouvert == 'f') || (input_flag_ouvert == 'F') ) {
        flag_ouvert = 0;
    }

    printf("\nEntree des parametres de Lennard-Jones, de la\n");
    printf("longueur du nanotube, des rayons des parois\n");
    printf("et des temperatures ou on calcule le second\n");
    printf("coefficient du viriel Bas\n\n");

    printf("(Pour avoir la courbe universelle de Bas/(L*sigma^2)\n");
    printf("versus kT/Es' ou versus R[i]/sigma, poser L = 1, sigma = 1\n");
    printf("epsilon = k/(Pi*theta) = 0.837657595\n");
    printf("Ici, Es' = Pi*theta*epsilon*sigma^2)\n\n");


    double eps;
    printf("epsilon (K) = ");
    scanf("%lf", &eps);

    double sigma;
```

```
printf("sigma (Angstrom) = ");
scanf("%lf", &sigma);

double L;
printf("Longueur du nanotube (Angstrom) = ");
scanf("%lf", &L);

    /* Mettre les rayons qui sont gardés fixés dans la
    suite Rtubes. */
int NbreParois;
printf("Entrez le nombre de parois formant le nanotube: ");
scanf("%d", &NbreParois);

double* Rtubes = (double*) malloc(NbreParois*sizeof(double));
if (Rtubes == NULL) {
    printf("\nErreur d'allocation de memoire\n");
    exit(1);
}

if (NbreParois > 1) {
    printf("\nEntrez les rayons (gardes fixes) des parois\n");
    printf("cylindriques du nanotube (en Angstroms):\n");
    for (int i = 1; i <= (NbreParois - 1); i++) {
        printf("R[%d] = ", i);
        scanf("%lf", &Rtubes[i -1]);
    }
            /* 1e rayon Rtubes[NbreParois -1] pourra être
            soit fixé, soit variable.  Cela sera déterminé
            ci-bas.
            */
}

char SerieDeT, SerieDeR;    /* variables disant si on
                            fait le calcul pour une série
                            de T ou une série de
                            R[NbreParois -1].
                            */
printf("\nVoulez-vous calculer Bas pour une serie de\n");
printf("temperatures? (O/N)? ");
cin >> SerieDeT;        /* ON N'A PAS PU UTILISER scanf()
                        ICI.  Il y a un BOGUE inexpliqué
                        quand on l'utilise!
                        */

if ((SerieDeT == 'O') || (SerieDeT == 'o')) {
    double T_min, T_max;
    int Nbre_T;
    printf("Entrez le rayon de la derniere paroi R[%d] (Angstrom):\n",
        NbreParois);
    printf("R[%d] = ", NbreParois);
    scanf("%lf", &Rtubes[NbreParois -1]);
    printf("Entrez la temperature minimale (K):\nT_min = ");
    scanf("%lf", &T_min);
    printf("Entrez la temperature maximale (K):\nT_max = ");
    scanf("%lf", &T_max);
    printf("Entrez le nombre de temperatures ou on calcule Bas:\nNbre_T = ");
    scanf("%d", &Nbre_T);

        /* Calculer la matriceBas contenant les Bas calculés
        aux différentes températures (pour un nanotube
        OUVERT ou FERMÉ aux extrémités). */
    double** matriceBas = matrice_Bas(T_min, T_max, Nbre_T, 0.0,
            Rtubes, NbreParois, L, THETA, eps, sigma, 1, flag_ouvert);
            /* ici, T_ou_rien est pris comme une valeur bidon car
            la température est variable.
```

```
                */

            /* Imprimer la matriceBas dans un fichier. */
        Ecrire_Bas(matriceBas, Nbre_T, 0.0, Rtubes, NbreParois,
                L, THETA, eps, sigma, 1, flag_ouvert);

            /* libérer la mémoire de la matriceBas avant
            de quitter le bloc. */
        liberermatrice(matriceBas);
    }

    else {
        printf("\nVoulez-vous calculer Bas pour une serie de\n");
        printf("rayons R[%d] de la %deme paroi du nanotube? (O/N)? ",
            NbreParois, NbreParois);
        cin >> SerieDeR;    /* ON N'A PAS PU UTILISER scanf()
                            ICI.  Il y a un BOGUE inexpliqué
                            quand on l'utilise!
                            */

        if ((SerieDeR == 'O') || (SerieDeR == 'o')) {
            double R_min, R_max, T;
            int Nbre_R;
            printf("Entrez la temperature de l'adsorbat (K):\nT = ");
            scanf("%lf", &T);
            printf("Entrez le rayon minimal de la %deme paroi (angstrom):\nR[%d]_min
= ",
                NbreParois, NbreParois);
            scanf("%lf", &R_min);
            printf("Entrez le rayon maximal de la %deme paroi (angstrom):\nR[%d]_max
= ",
                NbreParois, NbreParois);
            scanf("%lf", &R_max);
            printf("Entrez le nombre de rayons R[%d] ou on calcule Bas:\nNbre_R[%d] =
 ",
                NbreParois, NbreParois);
            scanf("%d", &Nbre_R);

                /* Calculer la matriceBas contenant les Bas calculés
                aux différentes valeurs du rayon variable R[NbreParois]. */
            double** matriceBas = matrice_Bas(R_min, R_max, Nbre_R, T, Rtubes,
                    NbreParois, L, THETA, eps, sigma, 0, flag_ouvert);

                /* Imprimer la matriceBas dans un fichier. */
            Ecrire_Bas(matriceBas, Nbre_R, T, Rtubes, NbreParois,
                    L, THETA, eps, sigma, 0, flag_ouvert);

                /* libérer la mémoire de la matriceBas avant
                de quitter le bloc. */
            liberermatrice(matriceBas);
        }


        else {
                /* si SerieDeT != oui et SerieDeR != oui, on calcule le Bas pour
                la température et le rayon de la paroi R[NbreParois] demandés par
                l'usager (et cela pour un nanotube à parois multiples OUVERT ou
                FERMÉ aux extrémités). */
            double T;
            printf("Entrez la temperature de l'adsorbat (K):\nT = ");
            scanf("%lf", &T);
            printf("Entrez le rayon de la derniere paroi R[%d] (Angstrom):\n",
                NbreParois);
            printf("R[%d] = ", NbreParois);
            scanf("%lf", &Rtubes[NbreParois -1]);
```

```
                /* On a besoin de connaître le rayon de la paroi externe
                avant de pouvoir faire le calcul de Bas.  i.e pour savoir
                où l'adsorption fini. */
            double Rexterne = R_externe(Rtubes, NbreParois);

                /* La distance du centre du nanotube où commence l'adsorption
                sera Rexterne ou 0.0 selon que le nanotube est FERMÉ ou
                OUVERT aux extrémités. */
            double r_min_adsorption = 0.0;
            if (flag_ouvert == 0) {
                    /* Le nanotube est alors fermé aux extrémités. */
                r_min_adsorption = Rexterne;
            }

            double Bas = Bas_Nanotube_multiwall(T, Rtubes, NbreParois,
                            L, THETA, eps, sigma, r_min_adsorption,
                            Rexterne + NBRE_DE_SIGMA_PASSE_Rmax * sigma);
            printf("\nBas (Angstrom^3) = %.6lg\n", Bas);
        }
    }

        /* libérer la mémoire du vecteur contenant les rayons des parois du
        nanotube avant de quitter le programme.
        */
    free(Rtubes);

        /* Avertir l'utilisateur que le programme à terminé tous les calculs
        (après un temps qui peut être pas mal long...)
        */
    for (int i = 1; i <= 4; i++) {
        time_t temps = time(NULL);
        for ( ; ; ) {
                /* Attendre 1 seconde entre les "bips". */
            if (time(NULL) == (temps + 1)) {
                printf("\a");
                break;
            }
        }
    }

    return 0;
}
```

```
/*****************************************************************

  Fonction:  matrice_Bas()

  Cette fonction génère une matrice[2][Nbre_x] contenant
  Nbre_x valeurs de second coefficient du viriel Bas,
  pour chaque valeur de x.  Ici, x pourra être soit un vecteur
  contenant une série de températures ou un vecteur contenant
  une série de rayons R pour une des parois du nanotube
  (simple ou multiple).  (Le dernier rayon du vecteur Rtubes
  est celui qu'on pourra faire varier, les autres étant
  toujours fixes).

  Appel:  matriceBas = matrice_Bas(x_min, x_max, Nbre_x,
                           T_ou_rien, Rtubes, NbreParois, L,
                           theta, eps, sigma, T_ou_R);


    matriceBas:  matrice contenant les Nbre_x valeurs
        x[i] et les valeurs Bas[i] correspondantes.
    x_min:  la valeur minimale de x (peut être T_min ou R_min).
    x_max:  la valeur maximale de x.
    Nbre_x:  le nombre de valeurs de x pour lesquelles
        on calcule les Bas correspondants.
    T_ou_rien: représente la température (fixée) lorsque l'on
        calcule Bas pour une série de rayons R.  Lorsque l'on
        calcule Bas pour une série de T, ça devient un paramètre
        inutile, car tous les rayons nécessaires
        (et gardés fixes) sont alors dans le vecteur Rtubes[].
    Rtubes:  vecteur contenant les rayons des parois du nanotube.
        Le dernier élément de ce vecteur pourra être variable ou
        fixe (fixe lorsque l'on calcule Bas pour une série de T;
        variable lorsque l'on calcule Bas pour une série de R).
    NbreParois: le nombre de parois cylindriques formant le
        nanotube.
    L:  la longueur du nanotube (Angstrom).
    theta: densité surfacique d'atomes de C formant les parois
        du nanotube.
    eps:  constante d'énergie du potentiel de Lennard-Jones.
    sigma:  rayon caractéristique du potentiel de Lennard-Jones
        entre un atome de C et une molécule d'adsorbat.
    T_ou_R:  variable (= 1 pour T, 0 pour R) disant si x est un
        vecteur de températures ou de rayons R.
    flag_ouvert:  flag indiquant si le nanotube a les extrémités
        ouvertes (flag_ouvert = 1) ou fermées (flag_ouvert = 0).

*****************************************************************/

#include "traitement.h"

double** matrice_Bas(double x_min, double x_max, int Nbre_x,
                double T_ou_rien, double* Rtubes, int NbreParois,
                double L, double theta, double eps, double sigma,
                int T_ou_R, int flag_ouvert) {

        /* espacement entre 2 valeurs x[i] */
    double delta_x = (fabs(x_max - x_min))/(Nbre_x - 1);

        /* formation de la matriceBas contenant les Bas */
    double** matriceBas = (double**) malloc(2 * sizeof(double*));
    if (matriceBas == NULL) {
        printf("\nErreur d'allocation de memoire\n");
        exit(1);
    }
```

```
for (int i = 1; i <= 2; i++) {
        /* allouer la mémoire pour les 2 lignes de la
        matriceBas:  x et Bas. */
    matriceBas[i -1] = (double*) malloc(Nbre_x * sizeof(double));
    if (matriceBas[i -1] == NULL) {
        printf("\nErreur d'allocation de memoire\n");
        exit(1);
    }
}

    /* Variable contenant la distance "r_min_adsorption"
    de l'axe du nanotube à partir de laquelle l'adsorption
    a lieu. (= 0 si le nanotube est ouvert; = Rexterne si
    il est fermé aux extrémités). */
double r_min_adsorption = 0.0;

    /* Si on fait le calcul des Bas pour une série de T,
    alors tous les rayons des parois seront fixes.  Donc
    le rayon externe reste fixe et on va le calculer ici
    une seule fois au lieu de le calculer à chaque T
    dans la boucle for suivante.
    */
double Rexterne;
if (T_ou_R != 0) {
    Rexterne = R_externe(Rtubes, NbreParois);

    if (flag_ouvert == 0) {
            /* l'intérieur du nanotube est alors
            inaccessible aux molécules de gaz. */
        r_min_adsorption = Rexterne;
    }
}

    /* remplir la matriceBas des valeurs de x et de Bas
    correspondantes (pour un nanotube à parois multiples
    OUVERT ou FERMÉ aux extrémités). */
for (i = 1; i <= Nbre_x; i++) {
    double x = x_min + (i - 1)*delta_x;
    double Bas;
    if (T_ou_R == 0) {
            /* R[NbreParois] est alors variable. */
        Rtubes[NbreParois -1] = x;

            /* Trouver le rayon de la paroi externe
            avec la fonction R_externe.  En effet, les
            rayons n'ont pas été ordonnés selon leur
            grandeur, dans le vecteur Rtubes[].
            */
        Rexterne = R_externe(Rtubes, NbreParois);

            /* Calculer la distance du centre du
            nanotube à partir de laquelle l'adsorption
            a lieu, selon que le nanotube est ouvert
            ou fermé. */
        if (flag_ouvert == 0) {
                /* le nanotube est alors fermé aux
                deux bouts. */
            r_min_adsorption = Rexterne;
        }

        Bas = Bas_Nanotube_multiwall(T_ou_rien, Rtubes,
                    NbreParois, L, theta, eps, sigma, r_min_adsorption,
                    Rexterne + NBRE_DE_SIGMA_PASSE_Rmax * sigma);
    }
    else {
```

```
            Bas = Bas_Nanotube_multiwall(x, Rtubes,
                NbreParois, L, theta, eps, sigma, r_min_adsorption,
                Rexterne + NBRE_DE_SIGMA_PASSE_Rmax * sigma);

        }

        matriceBas[1 -1][i -1] = x;
        matriceBas[2 -1][i -1] = Bas;
            /* Je préfère compter les indices à partir de 1, mais
            le C compte à partir de 0.  Je fais donc toujours une
            correction systématique de -1 aux indices des vecteurs
            et matrices.
            */
    }

    return matriceBas;

}
```

```
/*****************************************************************

  Fonction:  Bas_Nanotube_multiwall()

  Cette fonction calcule le second coefficient du viriel Bas
  pour l'adsorption sur un nanotube à parois multiples, en
  calculant l'intégrale de l'integrand_Bas sur tout l'espace
  où il y a de l'adsorption.

  Appel:  Bas = Bas_Nanotube_multiwall(T, Rtubes, NbreParois, L,
  theta, eps, sigma, borne_gauche, borne_droite);

    Bas:  le second coefficient du viriel.
    L:  la longueur du nanotube (Angstrom)
    T:  température du gaz (en K).
    Rtubes: suite contenant le rayon de chaque paroi
        cylindrique du nanotube (en Angstrom).
    NbreParois: le nombre de parois cylindriques formant le
        nanotube.
    theta: densité surfacique des atomes de C sur les parois du nanotube.
    eps: constante d'énergie du potentiel de Lennard-Jones.
    sigma: rayon caractéristique du potentiel de Lennard-Jones entre un
        atome de C et une molécule d'adsorbat.
    borne_gauche:  distance (>= 0), mesurée du centre du nanotube,
        à partir de laquelle l'adsorption a lieu.
    borne_droite:  distance (>= 0), mesurée du centre du nanotube,
        à partir de laquelle l'adsorption fini.

*****************************************************************/

#include "traitement.h"

double Bas_Nanotube_multiwall(double T, double* Rtubes,
                int NbreParois, double L, double theta,
                double eps, double sigma, double borne_gauche,
                double borne_droite) {

    return 2*PI*L*simpson_raffinee_Bas(&integrand_Bas,
                borne_gauche, borne_droite, T, Rtubes,
                NbreParois, theta, eps, sigma);

}
```

```
/*****************************************************************

  Fonction: integrand_Bas()

  Cette fonction calcule l'intégrand (= (exp(-V/kT) - 1)*r) qui
  entre dans le calcul du second coefficient du viriel Bas.  Cet
  intégrand devient -r lorsque l'on se situe très près d'une des
  parois.

  Appel:  integrand = integrand_Bas(r, T, Rtubes, NbreParois,
                                    theta, eps, sigma);

    integrand:  = (exp(-V/kT) - 1)*r, entrant dans l'intégrale du Bas.
    r:  distance par rapport au centre du nanotube, c'est où on
        calcule la valeur du potentiel (en Angstrom).
    T:  température du gaz (en K).
    Rtubes: suite contenant le rayon de chaque paroi cylindrique du nanotube
        (en Angstrom).
    NbreParois: le nombre de parois cylindriques formant le nanotube.
    theta: densité surfacique des atomes de C sur les parois du nanotube.
    eps: constante d'énergie du potentiel de Lennard-Jones.
    sigma: rayon caractéristique du potentiel de Lennard-Jones entre un
        atome de C et une molécule d'adsorbat.

*****************************************************************/

#include "traitement.h"

double integrand_Bas(double r, double T, double* Rtubes, int NbreParois,
                     double theta, double eps, double sigma) {

        /* si on est dans la "distance d'exclusion" d'une des
        parois, le potentiel tend vers l'infini et
        l'intégrand ===> -r. */
    for (int i = 1; i <= NbreParois; i++) {
        if (fabs(r - Rtubes[i -1]) < DISTREDUITEMIN*sigma) {
            return -r;
        }
    }

        /* si on est loin des parois, alors l'intégrand sera
        (exp(-V_multiwall/kT) - 1)*r. */
    return (exp((-1.0/(k*T))*(potentielMultiwall(r, Rtubes,
                NbreParois, theta, eps, sigma))) - 1.0)*r;

}
```

```
/*************************************************************************

  Fonction: potentielMultiwall()

  Cette fonction calcule la valeur du potentiel de Stan et Cole pour un
  nanotube à parois multiples en fonction de la distance par rapport à
  l'axe du nanotube.

  Appel: potentielMultiwall(r, Rtubes, NbreParois, theta, eps, sigma);

  r: distance par rapport à l'axe, du point où on mesure le potentiel
     (total, obtenu de la superposition du potentiel dû à chaque paroi).
  Rtubes: suite contenant le rayon de chaque paroi cylindrique du nanotube
  (l'ordre de ceux-ci dans la suite est sans importance).
  NbreParois: le nombre de parois cylindriques formant le nanotube.
  theta: densité surfacique des atomes de C sur les parois du nanotube.
  eps: constante d'énergie du potentiel de Lennard-Jones
  sigma: rayon caractéristique du potentiel de Lennard-Jones entre un
        atome de C et une molécule d'adsorbat.

*************************************************************************/

#include "traitement.h"

double potentielMultiwall(double r, double* Rtubes, int NbreParois,
                          double theta, double eps, double sigma) {

        /* le potentiel total est obtenu pour une valeur de r donnée en
        additionnant le potentiel V(r, R[i]) de chacune des "NbreParois"
        parois du nanotube. */
    double potentiel = 0;

    for (int i = 1; i <= NbreParois; i++) {
            potentiel = potentiel + potentiel1wall(r, Rtubes[i -1], theta, eps, sigma
);
    }

    return potentiel;

}
```

```
/*****************************************************************************

  Fonction: potentiel1wall()

  Cette fonction calcule la valeur du potentiel de Stan et Cole pour un
  nanotube à paroi simple en fonction de la distance par rapport à l'axe
  du nanotube.

  Appel: potentiel1wall(r, R, theta, eps, sigma);

  r: distance par rapport à l'axe, du point où on mesure le potentiel.
  R: rayon du nanotube.
  theta: densité surfacique des atomes de C sur les parois du nanotube.
  eps: constante d'énergie du potentiel de Lennard-Jones
  sigma: rayon caractéristique du potentiel de Lennard-Jones entre un
        atome de C et une molécule d'adsorbat.

*****************************************************************************/

#include "traitement.h"

double potentiellwall(double r, double R, double theta,
                      double eps, double sigma) {

    double x = r/R;     /* distance réduite */

    return 3*PI*theta*eps*sigma*sigma *
           ((21.0/32.0)*(pow(sigma/R, 10))*M(x, 11)
           - (pow(sigma/R, 4))*M(x, 5));

}
```

```
/*****************************************************************************

  Fonction M()

  Cette fonction calcule la valeur des M(x, n) servant au calcul du
  potentiel de Stan et Cole.

  appel: M(x, n)

  x: distance réduite ( = r/R) du centre du nanotube
     (à paroi simple)
  n: exposant.

*****************************************************************************/

#include "traitement.h"

double M(double x, int n) {

    double Mn = simpson_raffinee_Mn(&f, 0, PI, x, n);

    return Mn;

}
```

```
/*************************************************************************

  Fonction: f(phi, x, n)

  Cette fonction est la fonction dont on cherche la valeur de l'intégrale
  M(x, n) utilisée pour calculer le potentiel de Stan et Cole.

  L'INTÉGRAND POSSÈDE UNE SINGULARITÉ À x=1 et phi = 0.  Donc M(x, n)
  est non définie pour x=1. i.e. que le potentiel diverge à l'infini sur
  le nanotube.  On ne peut pas calculer V(r, R) pour r = R.

  Appel: F = f(phi, x, n);

    phi: variable d'intégration variant sur l'intervalle [0, PI]
    x: = r/R distance "réduite" de l'axe du nanotube.
    n: exposant apparaissant dans l'intégrand.

*************************************************************************/

#include "traitement.h"

double f(double phi, double x, int n) {

    return pow((1 + pow(x, 2) - 2*x*cos(phi)) , -(n/2.0));

}
```

```
/*****************************************************************

  Fonction: Ecrire_Bas()

  Cette fonction écrit dans un fichier "Bas_nanotube_OUVERT_T.txt"
  ou "Bas_nanotube_FERMER_T.txt" ou "Bas_nanotube_OUVERT_R.txt"
  ou "Bas_nanotube_FERMER_R.txt" les valeurs contenues dans la
  matriceBas pour les seconds coefficients du viriel Bas
  calculés aux différentes valeurs de x (i.e T ou R[NbreParois]).
  Le nanotube (à parois simples ou multiples) peut être OUVERT ou
  FERMÉ à ses deux extrémités.  Le fichier contient également
  les autres paramètres pour lesquels ces Bas ont été calculés
  (i.e. L; theta; eps; sigma; les autres R qui sont gardés
  fixes: R[1] à R[NbreParois - 1]; et R[NbreParois] ou T selon
  le cas).

  Appel:  Ecrire_Bas(matriceBas, Nbre_x, T_ou_rien, Rtubes,
                NbreParois, L, theta, eps, sigma, T_ou_R,
                flag_ouvert);


    matriceBas:  matrice contenant les Nbre_x valeurs
        x[i] et les valeurs Bas[i] correspondantes.
    Nbre_x:  le nombre de valeurs de x pour lesquelles
        on calcule les Bas correspondants.
    T_ou_rien: représente la température (fixée) lorsque l'on
        calcule Bas pour une série de rayons R.  Lorsque l'on
        calcule Bas pour une série de T, ça devient un paramètre
        inutile, car tous les rayons nécessaires
        (et gardés fixes) sont alors dans le vecteur Rtubes[].
    Rtubes:  vecteur contenant les rayons des parois du nanotube.
        Le dernier élément de ce vecteur pourra être variable ou
        fixe (fixe lorsque l'on calcule Bas pour une série de T;
        variable lorsque l'on calcule Bas pour une série de R).
    NbreParois: le nombre de parois cylindriques formant le
        nanotube.
    L:  la longueur du nanotube (Angstrom).
    theta: densité surfacique d'atomes de C formant les parois
        du nanotube.
    eps:  constante d'énergie du potentiel de Lennard-Jones.
    sigma:  rayon caractéristique du potentiel de Lennard-Jones
        entre un atome de C et une molécule d'adsorbat.
    T_ou_R:  variable (= 1 pour T, 0 pour R) disant si x est un
        vecteur de températures ou de rayons R.
    flag_ouvert:  flag indiquant si le nanotube a les extrémités
        ouvertes (flag_ouvert = 1) ou fermées (flag_ouvert = 0).

*****************************************************************/

#include "traitement.h"

void Ecrire_Bas(double** matriceBas, int Nbre_x, double T_ou_rien,
                double* Rtubes, int NbreParois, double L,
                double theta, double eps, double sigma,
                int T_ou_R, int flag_ouvert) {

        /* définition d'une chaîne OUV_FERM contenant le mot
        OUVERT ou FERMER selon que le nanotube est ouvert ou
        fermé aux extrémités.  Ce mot "variable" sera substitué
        dans les noms de fichiers et dans le texte contenu dans
        ceux-ci. */
    char* OUV_FERM;
    if (flag_ouvert == 0) {
            /* le nanotube est fermé aux extrémités. */
        OUV_FERM = "FERMER";
```

```
}
else {
    OUV_FERM = "OUVERT";
}

    /* Fabrication de la chaîne de caractères contenant
    le nom du fichier, selon que R ou T est le paramètre
    variable en fonction duquel on calcule Bas et selon
    que le nanotube est OUVERT ou FERMÉ aux extrémités. */
char file_name[30] = "Bas_nanotube_";
        /* file_name doit être assez longue pour contenir
        la chaîne contenant le nom de fichier final. */
if (T_ou_R == 0) {
    strcat(strcat(file_name, OUV_FERM), "_R.txt");
}
else {
    strcat(strcat(file_name, OUV_FERM), "_T.txt");
}

    /* Création du fichier contenant les valeurs de Bas pour
    les différentes valeurs de x (dont le nom a été formé
    ci-haut). */
FILE* fptr;
fptr = fopen(file_name, "w");
if (fptr == NULL) {
    printf("\nErreur d'ouverture de fichier\n");
    exit(1);
}


    /*  Mettre la valeur des paramètres physiques au début
    du fichier, avant de mettre les valeurs de Bas. */
fprintf(fptr, "Fichier %c%s%c donnant le second\n", 34, file_name, 34);
fprintf(fptr, "coefficient du viriel Bas, pour un nanotube a parois\n");
fprintf(fptr, "multiples %s a ses deux extremites.\n\n\n", OUV_FERM);

fprintf(fptr, "Nombre de parois cylindriques formant le nanotube:\n");
fprintf(fptr, "NbreParois = %d\n\n", NbreParois);
fprintf(fptr, "Longueur du nanotube:\nL (angstrom) = %.8lg\n\n", L);
    /* Le seul cas ou on auras des R[i] qui seront gardés fixes est
    lorsque l'on a plus d'une paroi ou que l'on calcule Bas pour une
    série de températures.
    */
if ((NbreParois > 1) || (T_ou_R != 0)) {
    fprintf(fptr, "Rayons des parois gardees fixes (angstrom):\n");
    for (int i = 1; i <= (NbreParois - 1); i++) {
        fprintf(fptr, "\tR[%d] = %.8lf\n", i, Rtubes[i -1]);
    }
}
if (T_ou_R == 0) {
    fprintf(fptr, "Temperature de l'adsorbat:\nT (K) = %lf\n", T_ou_rien);
}
else {
    fprintf(fptr, "\tR[%d] = %.8lf\n", NbreParois, Rtubes[NbreParois -1]);
}
fprintf(fptr, "\nDensite surfacique des atomes de C formant les parois:\n");
fprintf(fptr, "theta (angstrom^-2) = %lf\n\n", theta);

fprintf(fptr, "Valeurs des parametres du potentiel de Lennard-Jones:\n");
fprintf(fptr, "epsilon (K) = %lf\t\tsigma (angstrom) = %lf\n\n", eps, sigma);

    /* Mettre la valeur de Bas[i] correspondant à chaque valeur
    x[i] dans le fichier.  (i.e. mettre les éléments de matriceBas
    dans le fichier, en faisant une validation de l'écriture dans le
    fichier pour chaque valeur de Bas ajoutée.
```

```
        */
    fprintf(fptr, "Nombre de valeurs de Bas calculees: %d\n\n\n", Nbre_x);
    fprintf(fptr, "Valeurs du second coefficient du viriel Bas pour chaque\n");
    if (T_ou_R == 0) {
        fprintf(fptr, "valeur du rayon R[%d] dans la plage consideree:\n", NbreParois
);
        fprintf(fptr, "R[%d] (angstrom)\t\tBas (angstrom^3)\n", NbreParois);
    }
    else {
        fprintf(fptr, "temperature de la plage consideree:\n");
        fprintf(fptr, "T (K)\t\t\tBas (angstrom^3)\n");
    }

    for (int i = 1; i <= Nbre_x; i++) {
        if (fprintf(fptr, "%-13.6lg                %-13.6lg\n", matriceBas[1 -1][i -1
], matriceBas[2 -1][i -1]) < 0) {
            printf("\nErreur d'ecriture dans le fichier\n");
            exit(1);
        }
    }

        /* Fermer le fichier avant de quitter la fonction. */
    fclose(fptr);
}
```

```
/*****************************************************************

  Fonction: R_externe()

    Cette fonction trouve le rayon de la paroi externe d'un
    nanotube à parois multiples qui possède NbreParois parois.
    En effet, les rayons des parois, contenus dans le vecteur
    Rtubes[] ne sont pas classés du plus petit au plus grand,
    mais sont désordonnés.  Et on a besoin du rayon externe
    du nanotube pour pouvoir faire l'intégrale sur r calculant
    le Bas.

    Appel:  R_max = R_externe(Rtubes, NbreParois);

        R_max: le rayon de la paroi externe du nanotube.
        Rtubes: le vecteur contenant les rayons des parois du
            nanotube.
        NbreParois: le nombre de parois que le nanotube possède.

*****************************************************************/

#include "traitement.h"

double R_externe(double* Rtubes, int NbreParois) {

    double Rexterne = Rtubes[1 -1];
    for (int i = 2; i <= NbreParois; i++) {
        if (Rtubes[i -1] > Rexterne) {
            Rexterne = Rtubes[i -1];
        }
    }

    return Rexterne;
}
```

```
/*****************************************************************************

  Fonction: simpson_raffinee_Bas()

  Cette fonction utilise la fonction trapeze_etape_N_Bas() pour calculer
  l'intégrale de f(x, T, Rtubes, NbreParois, theta, eps, sigma) par rapport
  à x sur l'intervalle [a, b].  L'erreur relative permise sur l'intégrale est
  de ERREUR_RELATIVE_Bas.

  Cette fonction calcule l'intégrale par améliorations successives.
  Elle calcule une valeur améliorée à partir de la valeur précédente
  (qui est moins exacte).  JMAX est le nombre maximal d'améliorations
  qu'on permet pour le calcul de l'intégrale finale.  Donc 2^(JMAX-1)
  est le nombre total (maximal) de sous-intervalles qu'on peut avoir
  pour le calcul de l'intégrale.

  utilisant l'équation (voir Numerical Recipes):
        S = (4/3)*S[2N] - (1/3)*S[N]
  on peut calculer l'intégrale de Simpson à partir d'intégrales "trapèze"
  sans complications supplémentaires.

  Appel: integrale = simpson_raffinee_Bas(f, a, b, T, Rtubes, NbreParois,
                                    theta, eps, sigma);

        f: f(x, T, Rtubes, NbreParois, theta, eps, sigma) la fonction à
            intégrer par rapport à x.
        a: borne gauche de l'intervalle d'intégration.
        b: borne droite de l'intervalle d'intégration.
        T, Rtubes, NbreParois, theta, eps, sigma:  variables gardées fixes
            lors de l'intégration.

*****************************************************************************/

#include "traitement.h"

double simpson_raffinee_Bas(double (*f)(double x, double T, double* Rtubes,
                            int NbreParois, double theta, double eps,
                            double sigma), double a, double b,
                            double T, double* Rtubes, int NbreParois,
                            double theta, double eps, double sigma) {

    double integrale, old_integrale, integrale_trapeze,
           old_integrale_trapeze;
                        /* contient les integrales (actuelles et
                        anciennes) de Simpson et du trapèze. */

    old_integrale_trapeze = old_integrale = -1.0e30;
                        /* valeurs impossibles pour les vieilles
                        intégrales.  Ça permet de commencer la
                        boucle des "améliorations". */

    for (int j = 1; j <= JMAX; j++) {
            /* faire un nombre d'améliorations <= JMAX. */
        integrale_trapeze = trapeze_etape_N_Bas(f, a, b, j, T, Rtubes, NbreParois,
                                                theta, eps, sigma);
            /* calculer la jème approximation de l'intégrale de Simpson. */
        integrale = (4.0*integrale_trapeze - old_integrale_trapeze)/3.0;

        if (j > 5) {
                /* la condition j > 5 évite une (fausse) convergence
                trop hâtive. */
            if ((fabs(integrale - old_integrale)  < ERREUR_RELATIVE_Bas * fabs(old_in
tegrale))
                || (integrale == 0.0 && old_integrale == 0.0)) {
                    /* on a obtenu une valeur assez précise
```

```
                    de l'intégrale. */
                return integrale;
            }
        }

            /* les intégrales actuelles seront les "vieilles"
            intégrales de la prochaine approximation. */
        old_integrale = integrale;
        old_integrale_trapeze = integrale_trapeze;
    }

        /* Si on est rendu ici, c'est qu'on a pas réussi à converger.
        Ça peut être parce que le nombre d'étapes JMAX est trop petit
        (i.e. que le nombre d'améliorations maximal permis n'est pas
        assez grand pour atteindre la précision voulue). Ou que
        l'ERREUR_RELATIVE_Bas tolérée  est trop petite par rapport aux
        erreurs d'arrondis dûs à l'ordinateur (cause plus probable). */
    printf("\nL'erreur relative toleree sur l'integrale faite\n");
    printf("par la procedure simpson_raffinee_Bas() est trop\n");
    printf("petite pour avoir convergence.\n");
    exit(1);
    return 0.0; /* retourner une valeur double bidon car la fonction
                doit retourner quand même un double. */

}
```

```
/*************************************************************************

  Fonction: simpson_raffinee_Mn()

  Cette fonction utilise la fonction trapeze_etape_N_Mn() pour calculer
  l'intégrale de f(x, y, n) par rapport à x sur l'intervalle [a, b].
  L'erreur relative permise sur l'intégrale est de ERREUR_RELATIVE_Mn.

  Cette fonction calcule l'intégrale par améliorations successives.
  Elle calcule une valeur améliorée à partir de la valeur précédente
  (qui est moins exacte).  JMAX est le nombre maximal d'améliorations
  qu'on permet pour le calcul de l'intégrale finale.  Donc 2^(JMAX-1)
  est le nombre total (maximal) de sous-intervalles qu'on peut avoir
  pour le calcul de l'intégrale.

  utilisant l'équation (voir Numerical Recipes):
        S = (4/3)*S[2N] - (1/3)*S[N]
  on peut calculer l'intégrale de Simpson à partir d'intégrales "trapèze"
  sans complications supplémentaires.

  Appel: integrale = simpson_raffinee_Mn(f, a, b, y, n);

        f: f(x, y, n) la fonction à intégrer par rapport à x.
        a: borne gauche de l'intervalle d'intégration.
        b: borne droite de l'intervalle d'intégration.
        y: deuxième variable gardée fixe.
        n: exposant apparaissant dans f (gardé fixe).

*************************************************************************/

#include "traitement.h"

double simpson_raffinee_Mn(double (*f)(double x, double y, int n),
                        double a, double b, double y, int n) {

    double integrale, old_integrale, integrale_trapeze,
           old_integrale_trapeze;
                        /* contient les integrales (actuelles et
                        anciennes) de Simpson et du trapèze. */

    old_integrale_trapeze = old_integrale = -1.0e30;
                        /* valeurs impossibles pour les vieilles
                        intégrales.  Ça permet de commencer la
                        boucle des "améliorations". */

    for (int j = 1; j <= JMAX; j++) {
            /* faire un nombre d'améliorations <= JMAX. */
        integrale_trapeze = trapeze_etape_N_Mn(f, a, b, j, y, n);
            /* calculer la jème approximation de l'intégrale de Simpson. */
        integrale = (4.0*integrale_trapeze - old_integrale_trapeze)/3.0;

        if (j > 5) {
                /* la condition j > 5 évite une (fausse) convergence
                trop hâtive. */
            if ((fabs(integrale - old_integrale)  < ERREUR_RELATIVE_Mn * fabs(old_int
egrale))
                || (integrale == 0.0 && old_integrale == 0.0)) {
                    /* on a obtenu une valeur assez précise
                    de l'intégrale. */
                return integrale;
            }
        }

                /* les intégrales actuelles seront les "vieilles"
                intégrales de la prochaine approximation. */
```

```
            old_integrale = integrale;
            old_integrale_trapeze = integrale_trapeze;
        }

            /* Si on est rendu ici, c'est qu'on a pas réussi à converger.
            Ça peut être parce que le nombre d'étapes JMAX est trop petit
            (i.e. que le nombre d'améliorations maximal permis n'est pas
            assez grand pour atteindre la précision voulue). Ou que
            l'ERREUR_RELATIVE_Mn tolérée  est trop petite par rapport aux
            erreurs d'arrondis dûs à l'ordinateur (cause plus probable). */
        printf("\nL'erreur relative toleree sur l'integrale faite\n");
        printf("par la procedure simpson_raffinee_Mn() est trop\n");
        printf("petite pour avoir convergence.\n");
        exit(1);
        return 0.0; /* retourner une valeur double bidon car la fonction
                       doit retourner quand même un double. */

}
```

```
/*************************************************************************

  Fonction: trapeze_etape_N_Bas()

    Cette fonction retourne la Nème valeur raffinée de l'intégrale
    de f(x, T, Rtubes, NbreParois, theta, eps, sigma) sur [a, b] à partir
    de la (N-1)ème valeur (i.e. à partir de sa valeur précédente).
    L'intégration est faite par rapport à x, tandis que T, Rtubes, NbreParois,
    theta, eps et sigma sont gardées fixes.  La procédure a pour principe
    la règle du trapèze étendu (voir Numerical Recipes).

    Le premier appel calcule une intégrale grossière: un seul gros
    trapèze sur tout l'intervalle [a, b].  Les itérations suivantes
    N = 2, 3, ... rajoutent 2^(N-2) points de calculs au résultat
    précédent.

    Cette procédure a l'avantage de ne pas "jeter à la poubelle" le
    résultat précédent pour calculer une intégrale plus précise, mais
    d'utiliser le travail déjà fait.

  Appel: integrale = trapeze_etape_N_Bas(f, a, b, N, T, Rtubes, NbreParois,
                                         theta, eps, sigma);

        f: f(x, T, Rtubes, NbreParois, theta, eps, sigma) la fonction à intégrer
           par rapport à x.
        a: borne gauche de l'intervalle d'intégration.
        b: borne droite de l'intervalle d'intégration.
        N: indique qu'on est à la Nème étape de "raffinement".
        x: variable d'intégration.
        T, Rtubes, NbreParois, theta, eps, sigma:  variables gardées fixes
           lors de l'intégration.

*************************************************************************/

#include "traitement.h"

double trapeze_etape_N_Bas(double (*f)(double x, double T, double* Rtubes,
                           int NbreParois, double theta, double eps, double sigma),
                           double a, double b, int N, double T, double* Rtubes,
                           int NbreParois, double theta, double eps, double sigma) {

    double x, somme, delta;     /* somme est la somme de la nouvelle
                                   série de points à ajouter.
                                   delta est l'espace entre 2 de ces points.
                                   x est la position (abscisse) du point
                                   intermédiaire qu'on considère.
                                   */
    static double integrale;    /* contient présentement la (N-1)ème
                                   approximation de l'intégrale. (contiendra
                                   la Nème approximation à la fin de la
                                   procédure).
                                   */

    if (N == 1) {
            /* on calcule alors la première approximation très grossière
            donnée par un seul gros trapèze englobant tout l'intervalle
            [a, b]. */
        integrale = 0.5*(b-a)*((*f)(a, T, Rtubes, NbreParois, theta, eps, sigma)
                    + (*f)(b, T, Rtubes, NbreParois, theta, eps, sigma));
        return integrale;
    }

    else {
            /* calculer la Nème approximation à partir de l'approximation
            précédente. */
```

```
        delta = (fabs(b - a))/pow(2.0, N-2);
        x = a + delta/2;    /* la position du premier point intermédiaire
                            à ajouter. */

            /* additionner les valeurs de la fonction f prises aux points
            intermédiaires x[j] = x + (j-1)*delta. j allant de 1 à 2^(N-2). */
        somme = 0.0;
        for (int j = 1; j <= ((int) pow(2.0, N-2)); j++) {
            somme = somme + (*f)(x, T, Rtubes, NbreParois, theta, eps, sigma);
            x = x + delta;  /* passer au prochain point intermédiaire. */
        }

            /* calculer la Nème approximation de l'intégrale à partir
            de la (N-1)ème approximation. */
        integrale = 0.5*(integrale + delta*somme);
        return integrale;
    }
}
```

```
/*************************************************************************

  Fonction: trapeze_etape_N_Mn()

    Cette fonction retourne la Nème valeur raffinée de l'intégrale
    de f(x, y, n) sur [a, b] à partir de la (N-1)ème valeur (i.e.
    à partir de sa valeur précédente.  L'intégration est faite par
    rapport à x, tandis que y et n sont gardées fixes.  La procédure
    a pour principe la règle du trapèze étendu (voir Numerical Recipes).

    Le premier appel calcule une intégrale grossière: un seul gros
    trapèze dur tout l'intervalle [a, b].  Les itérations suivantes
    N = 2, 3, ... rajoutent 2^(N-2) points de calculs au résultat
    précédent.

    Cette procédure a l'avantage de ne pas "jeter à la poubelle" le
    résultat précédent pour calculer une intégrale plus précise, mais
    d'utiliser le travail déjà fait.

  Appel: integrale = trapeze_etape_N_Mn(f, a, b, N);

        f: f(x, y, n) la fonction à intégrer par rapport à x.
        a: borne gauche de l'intervalle d'intégration.
        b: borne droite de l'intervalle d'intégration.
        N: indique qu'on est à la Nème étape de "raffinement".
        x: variable d'intégration.
        y: deuxième variable gardée fixe.
        n: exposant apparaissant dans f (gardé fixe).

*************************************************************************/

#include "traitement.h"

double trapeze_etape_N_Mn(double (*f)(double x, double y, int n),
                          double a, double b, int N, double y, int n) {

    double x, somme, delta;     /* somme est la somme de la nouvelle
                                   série de points à ajouter.
                                   delta est l'espace entre 2 de ces points.
                                   x est la position (abscisse) du point
                                   intermédiaire qu'on considère.
                                   */
    static double integrale;    /* contient présentement la (N-1)ème
                                   approximation de l'intégrale. (contiendra
                                   la Nème approximation à la fin de la
                                   procédure).
                                   */

    if (N == 1) {
            /* on calcule alors la première approximation très grossière
            donnée par un seul gros trapèze englobant tout l'intervalle
            [a, b]. */
        integrale = 0.5*(b-a)*((*f)(a, y, n) + (*f)(b, y, n));
        return integrale;
    }

    else {
            /* calculer la Nème approximation à partir de l'approximation
            précédente. */
        delta = (fabs(b - a))/pow(2.0, N-2);
        x = a + delta/2;    /* la position du premier point intermédiaire
                               à ajouter. */

            /* additionner les valeurs de la fonction f prises aux points
            intermédiaires x[j] = x + (j-1)*delta. j allant de 1 à 2^(N-2). */
```

```
        somme = 0;
        for (int j = 1; j <= ((int) pow(2.0, N-2)); j++) {
            somme = somme + (*f)(x, y, n);
            x = x + delta;  /* passer au prochain point intermédiaire. */
        }

            /* calculer la Nème approximation de l'intégrale à partir
            de la (N-1)ème approximation. */
        integrale = 0.5*(integrale + delta*somme);
        return integrale;
    }
}
```

```
/*************************************************************************

  Fonction: liberermatrice()

  Cette fonction libère la mémoire d'une matrice à 2 dimensions contenant
  les points (X, Y)

  Appel: liberermatrice(matrice);

  matrice: la matrice[2][nbrePts dans les vecteurs X et Y] à libérer

*************************************************************************/

#include "traitement.h"

void liberermatrice(double** matrice) {

        /* libérer les 2 lignes */
    free(matrice[1 -1]);
    free(matrice[2 -1]);

        /* libérer la matrice contenant l'adresse des 2 lignes déjà
        libérées */
    free(matrice);

}
```

```
/*****************************************************************

  Fichier d'en-tête traitement.h

  Contient les librairies, les constantes et les déclarations
  (i.e les prototypes) de fonctions.

*****************************************************************/

#ifndef TRAITEMENT_H        /* Sert à éviter l'inclusion multiple de
                               traitement.h */
#define TRAITEMENT_H


#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <iostream.h>       /* Je dois utiliser la fonction
                               "cin >> " à cause du bogue avec
                               scanf() lorsqu'on veut lire un
                               char sur le clavier.
                               */


    /* Fichiers d'en-tête des modules servant à calculer
    les fonctions M(x, n) et l'intégrale double d'une
    fonction f(x, y) sur une région A du plan XY. */

#include "Mn.h"
#include "trapeze_2D.h"

    /* Définition de constantes symboliques servant au
    calcul du Bas d'un bundle de SWNT. */

#define PI 3.14159265358979323846264338327950288419 7

#define THETA 0.38
                            /* Angstrom^-2
                            Densité surfacique des atomes sur les parois
                            des nanotubes SWNT du bundle. */

#define k 1.0
                            /* constante de Boltzmann (= 1 lorsque
                            les énergies sont exprimées en K).
                            */

#define DISTANCE_REDUITE_EXCLUSION 0.40

                            /* distance (réduite) minimale d'une des
                            parois des nanotubes.  À l'intérieur de
                            cette distance d'exclusion, le potentiel
                            est considéré comme infini et
                            f = exp(-V/kT) - 1 ====> -1.  Cette
                            distance d'exclusion est nécessaire car
                            la procédure M(x, n) ne converge pas bien
                            pour |x - 1| < 1e-4.
                            */

#define T_MAX_REDUITE_DIST_EXCLUSION 176.863

                            /* Température (réduite) maximale pour
                            que la distance d'exclusion précédente
                            demeure valide.  Si la T est plus
```

```
                            grande, la distance d'exclusion devrait
                            être réduite (en gardant la condition
                            que |x - 1| > 1e-4). */

#define R_MIN_REDUIT_DIST_EXCLUSION 0.05

                            /* Rayon (réduit, i.e. R/sigma) minimal
                            des nanotubes du bundle pour que la
                            distance d'exclusion précédente demeure
                            valide (i.e. assez petite). */

#define NBRE_DE_SIGMA_AUTOUR_DU_BUNDLE 7

                            /* La distance (mesurées en sigma)
                            à partir des parois des SWNT les plus
                            externes du bundle où on considère
                            que l'adsorption fini.
                            */

#define PRINT_PROGRESSION_CALCUL_Bas 1

                            /* constante permettant d'imprimer le
                            nombre de Bas de la matriceBas qui ont
                            déjà été calculés à un instant donné,
                            afin de pouvoir suivre la progression
                            du très long calcul à l'écran.
                            ( = 1 pour l'imprimer, = 0 pour ne pas
                            l'imprimer). */



    /* fonction servant au calcul de V(r, R), le
    potentiel d'un nanotube à paroi simple (SWNT). */

double potentiel_SWNT(double r, double R, double theta,
                      double eps, double sigma);



    /* fonctions servant à la partie principale du programme:
    le calcul des Bas. */

double** GenererReseau(double d, int N_reseau, double *R_reseau);



void Fixer_ctes_integrand_Bas(double T_, double R_,
                              double** MatriceReseau_, int N_reseau_,
                              int flag_fermer_, double theta_,
                              double eps_, double sigma_);
double integrand_Bas(double x, double y);



void Fixer_R_cercle(double R_cercle_);
double frontiere_demi_cercle_gauche(double y);
double frontiere_demi_cercle_droit(double y);



double Bas_bundle_SWNT(double T, double R, double L,
                       double** MatriceReseau, int N_reseau,
                       double R_reseau, int flag_fermer,
                       double theta, double eps, double sigma);
```

```
    /* Autres fonctions accessoires servant à faire d'autres
    tâches connexes qui sont utiles au programme, mais qui
    n'interviennent pas directement dans le calcul des Bas. */

void ImprimerReseau(double** MatriceReseau, int N_reseau, double d,
                    double R_reseau, int flag_reseau_variable);



double** matrice_Bas(double x_min, double x_max, int Nbre_x,
                double T, double R, double** MatriceReseau,
                double R_reseau, int N_reseau, double L,
                double theta, double eps, double sigma,
                int T_ou_R_ou_d, int flag_fermer);

void Ecrire_Bas(double** matriceBas, int Nbre_x, double T,
                double R, double d, double R_reseau, int N_reseau,
                double L, double theta, double eps, double sigma,
                int T_ou_R_ou_d, int flag_fermer);



void liberermatrice(double** matrice);


#endif
```

```
/*****************************************************************

  Programme: Bas bundle SWNT

  Ce programme calcule le second coefficient du viriel Bas pour
  l'adsorption d'un gaz (qui est considéré monoatomique) sur un
  bundle de N_reseau nanotubes à parois simples (SWNT) de
  longueur L et de rayon R, dont les centres sont situés sur un
  réseau hexagonal de pas d.  Les nanotubes peuvent avoir les
  extrémités OUVERTES ou FERMÉES.

  L'usager commence par dire si les nanotubes sont OUVERTS ou
  FERMÉS aux extrémités.  Ensuite, il entre les paramètres de
  Lennard-Jones epsilon, sigma, la longueur L des nanotubes,
  le nombre de nanotubes N_reseau formant le bundle, leur rayon
  R, la distance d entre les centres des nanotubes, et la
  température.

  On peut calculer Bas pour un seul pas du réseau d,
  ou pour une série de pas du réseau compris entre d_min et d_max.
  On peut également calculer Bas pour une seule température ou pour
  une série de températures comprises entre T_min et T_max.  Aussi,
  on peut calculer Bas pour un seul rayon R, ou pour une série de
  rayons compris entre R_min et R_max.


  Appel:  Bas bundle SWNT

*****************************************************************/

/*****************************************************************
    NOTE IMPORTANTE SUR L'INDIÇAGE DES VECTEURS ET MATRICES:

    Je préfère compter les indices à partir de 1, mais
    le C compte à partir de 0.  Je fais donc toujours une
    correction systématique de -1 aux indices des vecteurs
    et matrices.

*****************************************************************/

#include "traitement.h"

int main(void) {

    printf("Ce programme calcule le second coefficient du viriel\n");
    printf("Bas pour l'adsorption sur un bundle de nanotubes\n");
    printf("de carbone OUVERTS ou FERMES a parois simples (SWNT)\n\n");

    char input_flag_fermer;
    int flag_fermer = 0;
    printf("Les nanotubes sont-ils ouverts ou fermes? (ouverts = o; fermes = f): ");
    cin >> input_flag_fermer;
    if ( (input_flag_fermer == 'f') || (input_flag_fermer == 'F') ) {
        flag_fermer = 1;
    }

    printf("\nEntree des parametres de Lennard-Jones, de la\n");
    printf("longueur des nanotubes, du nombre de nanotubes formant\n");
    printf("le bundle, du rayon de leur parois, du pas du reseau de\n");
    printf("nanotubes, et des temperatures ou on calcule le second\n");
    printf("coefficient du viriel Bas\n\n");

    printf("(Pour avoir la courbe universelle de Bas/(L*sigma^2)\n");
    printf("versus kT/Es', ou versus R/sigma, ou versus d/sigma,\n");
    printf("poser L = 1, sigma = 1, epsilon = k/(Pi*theta) = %.15lg\n",
```

```
       k/(PI*THETA));
printf("Ici, Es' = Pi*theta*epsilon*sigma^2)\n\n");


double eps;
printf("epsilon (K) = ");
scanf("%lf", &eps);

double sigma;
printf("sigma (Angstrom) = ");
scanf("%lf", &sigma);

double L;
printf("Longueur des nanotubes (Angstrom) = ");
scanf("%lf", &L);

int N_reseau;
printf("Nombre de nanotubes formant le bundle = ");
scanf("%d", &N_reseau);


char SerieDeT, SerieDeR, SerieDed;
                        /* variables disant si on
                        fait le calcul pour une série
                        de T, une série de R, ou une
                        série de d.
                        */

printf("\nVoulez-vous calculer Bas pour une serie de\n");
printf("temperatures (O/N)? ");
cin >> SerieDeT;
                        /* ON N'A PAS PU UTILISER scanf()
                        ICI.  Il y a un BOGUE inexpliqué
                        quand on l'utilise!
                        */


if ((SerieDeT == 'O') || (SerieDeT == 'o')) {

    double T_min, T_max;
    int Nbre_T;
    double T = 0.0, R, d, R_reseau;

    int T_ou_R_ou_d = 1;
    int flag_reseau_variable = 0;
            /* car T est le paramètre variable et qu'alors
            les centres des SWNTs sont gardés fixes. */

    printf("Entrez le rayon R des nanotubes du bundle (angstrom):\n");
    printf("R = ");
    scanf("%lf", &R);
    printf("Entrez la distance entre les centres des nanotubes:\n");
    printf("d (angstrom) = ");
    scanf("%lf", &d);
    if ((N_reseau > 1) && (d < 2*R)) {
        printf("\nVous avez entrer des dimensions inconsistantes\n");
        exit(1);
    }
    printf("Entrez la temperature minimale (K):\nT_min = ");
    scanf("%lf", &T_min);
    printf("Entrez la temperature maximale (K):\nT_max = ");
    scanf("%lf", &T_max);
    printf("Entrez le nombre de temperatures ou on calcule Bas:\nNbre_T = ");
    scanf("%d", &Nbre_T);
```

```
        /* Calculer la matrice MatriceReseau contenant les
        positions des points du réseau de pas d, constitué
        des centres des SWNTs du bundle. */

    double** MatriceReseau = GenererReseau(d, N_reseau, &R_reseau);

        /* Mettre les points du réseau (qui sont gardés fixes)
        dans un fichier. */

    ImprimerReseau(MatriceReseau, N_reseau, d, R_reseau,
                   flag_reseau_variable);

        /* Calculer la matriceBas contenant les Bas calculés
        aux différentes températures (pour un bundle de SWNTs
        OUVERTS ou FERMÉS aux extrémités).

        (le paramètre T apparaissant dans les procédures
        matrice_Bas() et Ecrire_Bas() est bidon). */

    double** matriceBas = matrice_Bas(T_min, T_max, Nbre_T,
                            T, R, MatriceReseau, R_reseau,
                            N_reseau, L, THETA, eps, sigma,
                            T_ou_R_ou_d, flag_fermer);

        /* Imprimer la matriceBas dans un fichier. */

    Ecrire_Bas(matriceBas, Nbre_T, T, R, d, R_reseau,
            N_reseau, L, THETA, eps, sigma, T_ou_R_ou_d,
            flag_fermer);

        /* libérer la mémoire de la MatriceReseau et de
        la matriceBas avant de quitter le bloc. */

    liberermatrice(MatriceReseau);
    liberermatrice(matriceBas);

}


else {

    printf("\nVoulez-vous calculer Bas pour une serie de\n");
    printf("rayons des SWNTs du bundle (O/N)? ");
    cin >> SerieDeR;
                            /* ON N'A PAS PU UTILISER scanf()
                            ICI.  Il y a un BOGUE inexpliqué
                            quand on l'utilise!
                            */


    if ((SerieDeR == 'O') || (SerieDeR == 'o')) {

        double R_min, R_max;
        int Nbre_R;
        double T, R = 0.0, d, R_reseau;

        int T_ou_R_ou_d = 2;
        int flag_reseau_variable = 0;
                /* car R est le paramètre variable et qu'alors
                les centres des SWNTs sont gardés fixes. */

        printf("Entrez la temperature de l'adsorbat (K):\n");
        printf("T = ");
        scanf("%lf", &T);
        printf("Entrez la distance entre les centres des nanotubes:\n");
```

```
    printf("d (angstrom) = ");
    scanf("%lf", &d);
    printf("Entrez le rayon minimal des SWNTs (angstrom):\nR_min = ");
    scanf("%lf", &R_min);
    printf("Entrez le rayon maximal des SWNTs (angstrom):\nR_max = ");
    scanf("%lf", &R_max);
    if ((N_reseau > 1) && ((d < 2*R_max) || (d < 2*R_min))) {
        printf("\nVous avez entrer des dimensions inconsistantes\n");
        exit(1);
    }
    printf("Entrez le nombre de rayons ou on calcule Bas:\nNbre_R = ");
    scanf("%d", &Nbre_R);

        /* Calculer la matrice MatriceReseau contenant les
        positions des points du réseau de pas d, constitué
        des centres des SWNTs du bundle. */

    double** MatriceReseau = GenererReseau(d, N_reseau, &R_reseau);

        /* Mettre les points du réseau (qui sont gardés fixes)
        dans un fichier. */

    ImprimerReseau(MatriceReseau, N_reseau, d, R_reseau,
                   flag_reseau_variable);

        /* Calculer la matriceBas contenant les Bas calculés
        pour les différents rayons des SWNTs du bundle.

        (le paramètre R apparaissant dans les procédures
        matrice_Bas() et Ecrire_Bas() est bidon). */

    double** matriceBas = matrice_Bas(R_min, R_max, Nbre_R,
                            T, R, MatriceReseau, R_reseau,
                            N_reseau, L, THETA, eps, sigma,
                            T_ou_R_ou_d, flag_fermer);

        /* Imprimer la matriceBas dans un fichier. */

    Ecrire_Bas(matriceBas, Nbre_R, T, R, d, R_reseau,
            N_reseau, L, THETA, eps, sigma, T_ou_R_ou_d,
            flag_fermer);

        /* libérer la mémoire de la MatriceReseau et de
        la matriceBas avant de quitter le bloc. */

    liberermatrice(MatriceReseau);
    liberermatrice(matriceBas);

}

else {

    printf("\nVoulez-vous calculer Bas pour une serie de\n");
    printf("distances d entre les centres des nanotubes\n");
    printf("(SWNTs) du bundle (O/N)? ");
    cin >> SerieDed;
                            /* ON N'A PAS PU UTILISER scanf()
                            ICI.  Il y a un BOGUE inexpliqué
                            quand on l'utilise!
                            */


    if ((SerieDed == 'O') || (SerieDed == 'o')) {

        double d_min, d_max;
```

```
    int Nbre_d;
    double T, R, d = 0.0, R_reseau = 0.0;

    int T_ou_R_ou_d = 3;
    int flag_reseau_variable = 1;
            /* car d est le paramètre variable et qu'alors
            les centres des SWNTs sont à espacement
            variable. */

    printf("Entrez la temperature de l'adsorbat (K):\n");
    printf("T = ");
    scanf("%lf", &T);
    printf("Entrez le rayon R des nanotubes du bundle (angstrom):\n");
    printf("R = ");
    scanf("%lf", &R);
    printf("Entrez la distance minimale entre les centres des SWNTs:\n");
    printf("d_min (angstrom) = ");
    scanf("%lf", &d_min);
    printf("Entrez la distance maximale entre les centres des SWNTs:\n");
    printf("d_max (angstrom) = ");
    scanf("%lf", &d_max);
    if ((N_reseau > 1) && ((d_min < 2*R) || (d_max < 2*R))) {
        printf("\nVous avez entrer des dimensions inconsistantes\n");
        exit(1);
    }
    printf("Entrez le nombre de distances d ou on calcule Bas:\n");
    printf("Nbre_d = ");
    scanf("%d", &Nbre_d);

        /* Déclarer une MatriceReseau bidon, puisque le réseau
        est variable, et que sa matrice est calculée dans la
        procédure matrice_Bas() dans ce cas. */

    double** MatriceReseau = NULL;

        /* Faire une tentative vaine d'imprimer le réseau. */

    ImprimerReseau(MatriceReseau, N_reseau, d, R_reseau,
                   flag_reseau_variable);

        /* Calculer la matriceBas contenant les Bas calculés
        pour les différents pas d du réseau.

        (les paramètres d, MatriceReseau et R_reseau
        apparaissant dans les procédures matrice_Bas() et
        Ecrire_Bas() sont alors bidons). */

    double** matriceBas = matrice_Bas(d_min, d_max, Nbre_d,
                            T, R, MatriceReseau, R_reseau,
                            N_reseau, L, THETA, eps, sigma,
                            T_ou_R_ou_d, flag_fermer);

        /* Imprimer la matriceBas dans un fichier. */

    Ecrire_Bas(matriceBas, Nbre_d, T, R, d, R_reseau,
            N_reseau, L, THETA, eps, sigma, T_ou_R_ou_d,
            flag_fermer);

        /* libérer la mémoire de la matriceBas avant
        de quitter le bloc. */

    liberermatrice(matriceBas);

}
```

```
        else {

                /* Si SerieDeT, SerieDeR et SerieDed ont tous reçu
                la valeur "non", on calcule la valeur de Bas
                correspondant à la température, au rayon des SWNTs
                et au pas d du réseau donnés par l'usager. */

            double T, R, d, R_reseau;

            int flag_reseau_variable = 0;
                    /* car les centres des SWNTs ont un
                    espacement fixé par l'usager. */

            printf("Entrez la temperature de l'adsorbat (K):\n");
            printf("T = ");
            scanf("%lf", &T);
            printf("Entrez le rayon R des nanotubes du bundle (angstrom):\n");
            printf("R = ");
            scanf("%lf", &R);
            printf("Entrez la distance entre les centres des nanotubes:\n");
            printf("d (angstrom) = ");
            scanf("%lf", &d);
            if ((N_reseau > 1) && (d < 2*R)) {
                printf("\nVous avez entrer des dimensions inconsistantes\n");
                exit(1);
            }

                /* Calculer la matrice MatriceReseau contenant les
                positions des points du réseau de pas d, constitué
                des centres des SWNTs du bundle. */

            double** MatriceReseau = GenererReseau(d, N_reseau, &R_reseau);

                /* Mettre les points du réseau (qui sont gardés fixes)
                dans un fichier. */

            ImprimerReseau(MatriceReseau, N_reseau, d, R_reseau,
                           flag_reseau_variable);

                /* Calculer la valeur de Bas demandée par l'usager
                (pour T, R, d donnés) et l'imprimer à l'écran. */

            double Bas_bundle_SWNTs = Bas_bundle_SWNT(T, R, L, MatriceReseau,
                                        N_reseau, R_reseau, flag_fermer,
                                        THETA, eps, sigma);

            printf("\nBas (angstrom^3) = %.6lg\n", Bas_bundle_SWNTs);

                /* libérer la mémoire de la MatriceReseau avant
                de quitter le bloc. */

            liberermatrice(MatriceReseau);

        }
    }
}


    /* Avertir l'utilisateur que le programme à terminé tous les calculs
    (après un temps qui peut être pas mal long...)
    */

for (int i = 1; i <= 4; i++) {
    time_t temps = time(NULL);
    for ( ; ; ) {
```

```
                /* Attendre 1 seconde entre les "bips". */
            if (time(NULL) == (temps + 1)) {
                printf("\a");
                break;
            }
        }
    }

    return 0;
}
```

```
/*****************************************************************

  Fonction:  Bas_bundle_SWNT()

  Cette fonction calcule le second coefficient du viriel Bas pour
  l'adsorption d'un gaz (considéré monoatomique) sur un bundle de
  nanotubes OUVERTS ou FERMÉS à parois simples (SWNTs).

  Elle calcule Bas en effectuant l'intégrale de (exp(-V_total/kT) - 1)
  sur tout l'espace où on considère qu'il y a de l'adsorption.

  Appel:  Bas = Bas_bundle_SWNT(T, R, L, MatriceReseau, N_reseau,
                                R_reseau, flag_fermer, theta, eps,
                                sigma);

    Bas:            le second coefficient du viriel pour le bundle
                        de SWNTs OUVERTS ou FERMÉS.
    T:              température du gaz (en K).
    R:              rayon des nanotubes SWNT du bundle (en Angstrom).
    L:              longueur des nanotubes SWNT formant le bundle
                        (en Angstrom).
    MatriceReseau:  matrice contenant les points du réseau (centres
                        des nanotubes).
    N_reseau:       le nombre de points du réseau.
    R_reseau:       le rayon du plus petit cercle englobant tous les
                        points du réseau.
    flag_fermer:    flag disant si les nanotubes à parois simples
                        du bundle sont OUVERTS ou FERMÉS.
    theta:          densité surfacique des atomes de C sur les
                        parois des nanotubes SWNT du bundle.
    eps:            constante d'énergie du potentiel de
                        Lennard-Jones.
    sigma:          rayon caractéristique du potentiel de
                        Lennard-Jones entre un atome de C et une
                        molécule d'adsorbat.

*****************************************************************/

#include "traitement.h"

double Bas_bundle_SWNT(double T, double R, double L,
                       double** MatriceReseau, int N_reseau,
                       double R_reseau, int flag_fermer,
                       double theta, double eps, double sigma) {

        /* Commencer par s'assurer que la température et le rayon
        des nanotubes du bundle pour lesquels on calcule Bas soient
        dans les plages de valeurs permises pour que la distance
        d'exclusion demeure valide. */

    if ((T > T_MAX_REDUITE_DIST_EXCLUSION * (PI*THETA*eps*sigma*sigma))
        || (R < R_MIN_REDUIT_DIST_EXCLUSION * sigma)) {

        printf("\nSoit que la temperature ou on calcule Bas est trop\n");
        printf("elevee ou que le rayon des nanotubes du bundle est\n");
        printf("trop petit pour que la distance d'exclusion demeure\n");
        printf("valide.  Reduisez la DISTANCE_REDUITE_EXCLUSION.\n");

        exit(1);
    }

        /* On calcule le rayon du cercle de rayon R_cercle entourant
        le bundle de SWNTs. */

    double R_cercle = R_reseau + R
```

```
                                + NBRE_DE_SIGMA_AUTOUR_DU_BUNDLE * sigma;

        /* Fixer les constantes apparaissant dans l'integrand_Bas() et
        le rayon R_cercle apparaissant dans les fonctions frontieres
        définissant le cercle.  Ces constantes sont maintenues fixes
        lors de l'intégration calculant la contribution au Bas de la
        région circulaire centrale. */

    Fixer_ctes_integrand_Bas(T, R, MatriceReseau, N_reseau,
                             flag_fermer, theta, eps, sigma);

    Fixer_R_cercle(R_cercle);

        /* Calculer la contribution au Bas due au cercle de rayon
        R_cercle entourant le bundle. */

    double Bas_du_au_centre = L * trapeze_2D(&integrand_Bas,
                  &frontiere_demi_cercle_gauche,
                  &frontiere_demi_cercle_droit, -R_cercle, R_cercle);

        /* Retourner la valeur totale du Bas pour le bundle de SWNTs
        OUVERTS ou FERMÉS aux extrémités. */

    return Bas_du_au_centre;

}
```

```
/*****************************************************************

  Fonction: integrand_Bas()

  Cette fonction calcule l'intégrand (= exp(-V_total/kT) - 1) qui
  entre dans le calcul du second coefficient du viriel Bas pour
  un bundle de SWNTs OUVERTS ou FERMÉS à leurs extrémités.  Cet
  intégrand devient -1 lorsque l'on se situe très près d'une des
  parois d'un des nanotubes.  Il devient 0 lorsque l'on se situe à
  l'intérieur d'un nanotube FERMÉ (i.e. que l'on a un bundle de
  SWNT FERMÉS à leurs extrémités).  Il ne restera alors plus
  qu'à calculer son intégrale double sur tout le plan XY pour
  avoir le second coefficient du viriel Bas d'un bundle de
  nanotubes OUVERTS ou FERMÉS.

  Appel:  integrand = integrand_Bas(x, y);

    integrand:  = exp(-V_total/kT) - 1, entrant dans l'intégrale du
                  Bas d'un bundle de SWNT.
    (x, y):     Position du point où on calcule la valeur de
                  l'intégrand.


  Les autres paramètres, qui sont gardés fixes lors de l'intégration
  sur le plan XY, seront fixés par une autre fonction:
  Fixer_ctes_integrand_Bas().

  Appel:  Fixer_ctes_integrand_Bas(T, R, MatriceReseau, N_reseau,
                                   flag_fermer, theta, eps, sigma);

    T:              température du gaz (en K).
    R:              rayon des nanotubes SWNT du bundle (en Angstrom).
    MatriceReseau:  matrice contenant les points du réseau (centres
                      des nanotubes).
    N_reseau:       le nombre de points du réseau.
    flag_fermer:    flag disant si les nanotubes à parois simples
                      du bundle sont OUVERTS ou FERMÉS
    theta:          densité surfacique des atomes de C sur les
                      parois des nanotubes SWNT du bundle.
    eps:            constante d'énergie du potentiel de
                      Lennard-Jones.
    sigma:          rayon caractéristique du potentiel de
                      Lennard-Jones entre un atome de C et une
                      molécule d'adsorbat.

*****************************************************************/

#include "traitement.h"

    /* Variables externes, mais qui restent définies seulement à
    l'intérieur de ce fichier source.  Ces variables apparaissent
    dans l'integrand_Bas comme des constantes lors de l'intégration
    sur le plan XY. */

static double T, R, theta, eps, sigma;
static int N_reseau, flag_fermer;
static double** MatriceReseau;



double integrand_Bas(double x, double y) {

        /* Pour une position (x, y) donnée, scanner tous les nanotubes pour
        voir si (x, y) est à l'intérieur d'un nanotube FERMÉ ou si (x, y)
        est très près d'une paroi d'un des nanotubes (i.e. que (x, y) est à
```

```
        l'intérieur de la "distance d'exclusion" où on considère que
        exp(-V_total/kT) ---> 0). */

    for (int j = 1; j <= N_reseau; j++) {

            /* Position du point du réseau considéré: */

        double Xj = MatriceReseau[1 -1][j -1];
        double Yj = MatriceReseau[2 -1][j -1];

            /* Distance entre (x, y) et le point "j" du réseau considéré
            (centre du nanotube). */

        double r = sqrt((x - Xj)*(x - Xj) + (y - Yj)*(y - Yj));

        if ((r < R) && (flag_fermer != 0)) {

                /* Le point (x, y) est alors à l'intérieur d'un SWNT
                FERMÉ. */

            return 0.0;
        }

        if (fabs(r - R) < DISTANCE_REDUITE_EXCLUSION * sigma) {

                /* Le point (x, y) est alors très près de la paroi
                du nanotube j. */

            return -1.0;
        }

    }

        /* Si on est rendu ici, c'est que le point (x, y) considéré
        est ni dans un SWNT fermé, ni trop près de la paroi d'un des
        nanotubes.  L'integrand_Bas sera alors calculé au moyen de sa
        forme exacte:  exp(-V_total/kT) - 1. */

    double V_total = 0;

    for (j = 1; j <= N_reseau; j++) {

            /* Position du point du réseau considéré: */

        double Xj = MatriceReseau[1 -1][j -1];
        double Yj = MatriceReseau[2 -1][j -1];

            /* Distance entre (x, y) et le point "j" du réseau considéré
            (centre du nanotube). */

        double r = sqrt((x - Xj)*(x - Xj) + (y - Yj)*(y - Yj));

            /* Addition de la contribution du potentiel du nanotube j
            au potentiel total au point (x, y). */

        V_total = V_total + potentiel_SWNT(r, R, theta, eps, sigma);

    }

        /* Retourner la valeur de l'integrand_Bas au point (x, y)
        considéré. */

    return (exp(-V_total/(k*T)) - 1.0);

}
```

```
void Fixer_ctes_integrand_Bas(double T_, double R_,
                              double** MatriceReseau_, int N_reseau_,
                              int flag_fermer_, double theta_,
                              double eps_, double sigma_) {

        /* Fonction servant à fixer les constantes apparaissant dans
        la fonction integrand_Bas() et qui sont gardées fixes lors de
        l'intégration sur le plan XY. */

    T = T_;
    R = R_;
    MatriceReseau = MatriceReseau_;
    N_reseau = N_reseau_;
    flag_fermer = flag_fermer_;
    theta = theta_;
    eps = eps_;
    sigma = sigma_;
}
```

```
/*****************************************************************

  Fonctions: frontiere_demi_cercle_gauche()
             frontiere_demi_cercle_droit()

  Ces fonctions décrivent les parties gauche et droite du cercle
  qui englobe tout le bundle de nanotubes à parois simples.
  Le cercle a un rayon R_cercle.  R_cercle doit être pris
  infiniment grand en théorie, puisque l'adsorption a lieu dans
  tout l'espace entourant le bundle.  Mais en pratique, R_cercle
  aura une grande valeur de R_reseau + quelques sigma.

  Appel: x = frontiere_demi_cercle_gauche(y)
         x = frontiere_demi_cercle_droit(y)

    x: variable sur laquelle a lieu l'intégration interne.
    y: variable variant de -R_cercle à +R_cercle et sur laquelle
       a lieu l'intégration externe. (y est maintenu fixe
       lors de l'intégration interne).



  Le rayon R_cercle du cercle à l'intérieur duquel on intègre
  (exp(-V_total/kT) - 1) et qui est gardé fixe lors de
  l'intégration, sera fixé par une autre procédure
  Fixer_R_cercle()

  Appel: Fixer_R_cercle(R_cercle)

    R_cercle: le rayon du cercle qui englobe tout le bundle de
              nanotubes et sur lequel on effectue l'intégration
              afin de trouver le second coefficient du viriel
              Bas.

*****************************************************************/

#include "traitement.h"

static double R_cercle;     /* Variable externe, mais qui reste
                               définie seulement à l'intérieur de ce
                               fichier source. */


double frontiere_demi_cercle_gauche(double y) {

    return -sqrt(R_cercle*R_cercle - y*y);

}


double frontiere_demi_cercle_droit(double y) {

    return sqrt(R_cercle*R_cercle - y*y);

}


void Fixer_R_cercle(double R_cercle_) {

        /* Fonction servant à fixer la constante R_cercle
        apparaissant dans les fonctions
        frontiere_demi_cercle_gauche(y) et
        frontiere_demi_cercle_droit(y) et qui demeure fixée
        lors de l'intégration sur la région circulaire englobant
        le bundle. */
```

```
    R_cercle = R_cercle_;
}
```

```
/*****************************************************************

  Fichier d'en-tête trapeze_2D.h

  Contient les prototypes des fonctions, les fichier d'en-tête
  des librairies, et les définitions des constantes utilisées
  exclusivement par la fonction trapeze_2D() (servant à
  effectuer l'intégrale double d'une fonction f(x, y) sur une
  région A du plan XY).

*****************************************************************/

#ifndef TRAPEZE_2D_H        /* Sert à éviter l'inclusion multiple
                               de trapeze_2D.h */
#define TRAPEZE_2D_H


#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#define ERREUR_RELATIVE_TRAPEZE_INTERNE 1e-3

                            /* erreur (relative) tolérée sur le résultat
                               de l'intégrale "interne" calculée, par rapport
                               à la variable x, avec la méthode du
                               trapèze. */

#define ERREUR_RELATIVE_TRAPEZE_EXTERNE 1e-2

                            /* erreur (relative) tolérée sur le résultat
                               de l'intégrale "externe" calculée, par rapport
                               à la variable y, avec la méthode du
                               trapèze. */

#define N_MAX__TRAP_INTERNE 40

                            /* Le rang MAXIMAL pour la dernière intégrale
                               du trapèze (de l'intégration interne, par
                               rapport à x) calculée.  Il est tel que
                               2^N_MAX__TRAP_INTERNE est le nombre MAXIMAL de
                               sous-intervalles qu'on peut avoir pour
                               faire l'intégration. */

#define N_MAX__TRAP_EXTERNE 25

                            /* Le rang MAXIMAL pour la dernière intégrale
                               du trapèze (de l'intégration externe, par
                               rapport à y) calculée.  Il est tel que
                               2^N_MAX__TRAP_EXTERNE est le nombre MAXIMAL de
                               sous-intervalles qu'on peut avoir pour
                               faire l'intégration. */

#define PRINT_INT_INTERNE 0

                            /* Constante de déboguage permettant d'imprimer
                               les résultats intermédiaires de l'intégration
                               interne sur la variable x, si désiré par le
                               programmeur (1: les imprimer ; 0: ne pas les
                               imprimer). */

#define PRINT_INT_EXTERNE 0

                            /* Constante de déboguage permettant d'imprimer
                               les résultats intermédiaires de l'intégration
```

```
                        externe sur la variable y, si désiré par le
                        programmeur (1: les imprimer ; 0: ne pas les
                        imprimer). */


    /* Prototype des fonctions servant à effectuer l'intégrale
    double de f(x, y) sur A. */

double trapeze_2D_interne(double (*f) (double x, double y),
                          double a, double b, double y);

double trapeze_2D(double (*f) (double x, double y),
                  double (*frontiere_gauche)(double y),
                  double (*frontiere_droite)(double y),
                  double y_min, double y_max);


#endif
```

```
/*****************************************************************

  Fonction: trapeze_2D()

  Cette fonction calcule l'intégrale double d'une fonction
  f(x, y) au moyen de 2 règles du trapèze imbriquées, sur
  la région d'intégration A. La région d'intégration A est
  spécifiée par les fonctions frontiere_gauche(),
  frontiere_droite() et les droites y = y_min, y = y_max.

  La procédure commence par calculer un premier estimé de
  l'intégrale (extérieure, sur la variable y), en supposant
  qu'on a un seul gros trapèze sur l'intervalle d'intégration
  [y_min, y_max].  Puis, lors des approximations suivantes,
  elle divise chacun des sous-intervalles en deux pour
  calculer un nouvel estimé de l'intégrale.  La nouvelle
  estimation de l'intégrale est faite au moyen de son
  estimation précédente, pour ne pas perdre le bénéfice du
  travail déjà fait.

  La fonction continue à calculer les estimés successifs de
  l'intégrale (externe sur y) jusqu'à ce que la précision
  désirée (ERREUR_RELATIVE_TRAPEZE_EXTERNE) soit atteinte
  (ou que l'on ait atteint le rang maximal
  N_MAX__TRAP_EXTERNE de la dernière intégrale du trapèze
  permise, même si l'on a pas réussi à atteindre la précision
  ERREUR_RELATIVE_TRAPEZE_EXTERNE demandée).


  Appel: integrale_2D = trapeze_2D(&f, &frontiere_gauche,
                                   &frontiere_droite,
                                   y_min, y_max);

        f: f(x, y) la fonction à intégrer sur la région A.
        frontiere_gauche: frontiere_gauche(y) est la
           fonction représentant la frontiere gauche de
           la région A d'intégration.
        frontiere_droite: frontiere_droite(y) est la
           fonction représentant la frontiere droite de
           la région A d'intégration.
        y_min: droite inférieure délimitant la région A.
        y_max: droite supérieure délimitant la région A.


  Référence: B. M. Ayyub; R. H. McCuen; "Numerical methods for
             Engineers"; p. 198-201.

*****************************************************************/

#include "trapeze_2D.h"

#define FUNC(y) (trapeze_2D_interne(f, (*frontiere_gauche)(y),\
                 (*frontiere_droite)(y), y))

                                /* Macro permettant de simplifier
                                l'écriture des appels de fonctions
                                dans la procédure.

                                FUNC(y) est en fait le résultat
                                de l'intégrale interne, pour un
                                certain y donné. */


double trapeze_2D(double (*f) (double x, double y),
```

```
                    double (*frontiere_gauche)(double y),
                    double (*frontiere_droite)(double y),
                    double y_min, double y_max) {

    double new_trapeze, old_trapeze = 0.0;
                                        /* variables contenant
                                        la valeur de l'estimation
                                        courante et antérieure de
                                        l'intégrale (externe sur y),
                                        obtenues avec la règle du
                                        trapèze. */


        /* Calculer les approximations successives de l'intégrale
        (externe, sur y) au moyen de la règle du trapèze, en
        divisant par 2 les sous-intervalles servant à son calcul,
        pour chaque nouvel estimé de son résultat. */

    for (int i = 0; i <= N_MAX__TRAP_EXTERNE; i++) {

        if (i == 0) {

                /* Calculer le premier estimé de l'intégrale (externe,
                sur y) au moyen de la règle du trapèze (obtenu avec
                un seul gros trapèze). */

            new_trapeze = 0.5*(y_max - y_min)*(FUNC(y_min) + FUNC(y_max));

                /* Imprimer la valeur courante (intermédiaire) de l'intégrale,
                si désiré par le programmeur (pour fin de déboguage). */

#if PRINT_INT_EXTERNE
            printf("\t\tINTEGRALE EXTERNE[%3d] = %-25.20lg\n", i, new_trapeze);
#endif

        }

        else {

                /* La valeur courante de l'intégrale (de rang i) est
                obtenue par une règle du trapèze utilisant 2^i
                sous-intervalles. Le résultat de la règle du trapèze
                courante (de rang i) peut s'obtenir en utilisant le
                travail déjà fait pour le calcul de la règle du trapèze
                précédente (de rang i - 1). */


                  /* calcul de la somme de f aux nouveaux points
                  ajoutés. */

            double somme = 0;
            for (int k = 1; k <= ((1 << i) - 1); k = k + 2) {

                        /* Où on a utilisé le fait que 2^i = 1 << i.
                        I.e. que dans la représentation binaire
                        du nombre 1, on peut le multiplier par 2
                        simplement en lui ajoutant un zéro à sa droite.
                        (Ce qui est plus rapide que d'utiliser
                        pow(2, i)). */

                somme = somme + FUNC( y_min + (k*(y_max - y_min))/(1 << i) );
            }

                /* calcul de l'estimé courant (de rang i) de l'intégrale: */
```

```
            new_trapeze = 0.5*old_trapeze + ((y_max - y_min)/(1 << i))*somme;

                /* Imprimer la valeur courante (intermédiaire) de l'intégrale,
                si désiré par le programmeur (pour fin de déboguage). */

#if PRINT_INT_EXTERNE
            printf("\t\tINTEGRALE EXTERNE[%3d] = %-25.20lg\n", i, new_trapeze);
#endif

                /* Si la tolérance en erreur est atteinte, retourner
                la valeur de l'intégrale courante. */

            if (i > 5) {

                    /* la condition i > 5 sert à éviter la possibilité
                    d'une (fausse) convergence trop hâtive. */

                if (  (fabs(new_trapeze - old_trapeze) < ERREUR_RELATIVE_TRAPEZE_EXTE
RNE * fabs(old_trapeze))
                      || ((new_trapeze == 0) && (old_trapeze == 0))  ) {

                    return new_trapeze;
                }
            }
        }

            /* La valeur courante (de rang i) de l'intégrale deviendra
            sa valeur antérieure (de rang i - 1) lors de l'itération
            suivante: */

        old_trapeze = new_trapeze;

    }
        /* Fin de la boucle "for" calculant les approximations
        successives de l'intégrale. */



        /* Si l'on est rendu ici, c'est que l'on a atteint le
        nombre maximal de sous-intervalles de [y_min, y_max] permis,
        sans avoir réussi à rencontrer le critère de convergence.

        Soit que le nombre de sous-intervalles 2^N_MAX__TRAP_EXTERNE
        est trop petit pour atteindre la précision voulue, ou que
        l'ERREUR_RELATIVE_TRAPEZE_EXTERNE tolérée est trop petite
        par rapport aux erreurs commises sur l'intégrale interne
        (qui a lieu sur la variable x). */

    printf("\nL'erreur relative toleree sur l'integrale externe\n");
    printf("faite par la procedure trapeze_2D() n'a pas pu\n");
    printf("etre atteinte.  Modifiez N_MAX__TRAP_EXTERNE\n");
    printf("et/ou ERREUR_RELATIVE_TRAPEZE_EXTERNE\n");

    exit(1);
    return 0.0;     /* Retourner une valeur double bidon car la
                    procédure doit quand même retourner un double. */

}
```

```
/*************************************************************

  Fonction: trapeze_2D_interne()

  Cette fonction calcule l'intégrale d'une fonction f(x, y) au
  moyen de la méthode du trapèze (par rapport à x sur
  l'intervalle [a, b]).

  La procédure commence par calculer un premier estimé de
  l'intégrale, en supposant qu'on a un seul gros trapèze sur
  l'intervalle d'intégration [a, b].  Puis, lors des
  approximations suivantes, elle divise chacun des
  sous-intervalles en deux pour calculer un nouvel estimé
  de l'intégrale.  La nouvelle estimation de l'intégrale est
  faite au moyen de son estimation précédente, pour ne pas
  perdre le bénéfice du travail déjà fait.

  La fonction continue à calculer les estimés successifs de
  l'intégrale jusqu'à ce que la précision désirée
  (ERREUR_RELATIVE_TRAPEZE_INTERNE) soit atteinte (ou que
  l'on ait atteint le rang maximal N_MAX__TRAP_INTERNE
  de la dernière intégrale du trapèze permise, même si l'on
  a pas réussi à atteindre la précision
  ERREUR_RELATIVE_TRAPEZE_INTERNE demandée).


  Appel: integrale = trapeze_2D_interne(&f, a, b, y);

        f: f(x, y) la fonction à intégrer par rapport à x.
        a: borne gauche de l'intervalle d'intégration.
        b: borne droite de l'intervalle d'intégration.
        y: deuxième argument de la fonction f() et qui est
               gardé fixe lors de l'intégration sur x.


  Référence: B. M. Ayyub; R. H. McCuen; "Numerical methods for
             Engineers"; p. 198-201.

*************************************************************/

#include "trapeze_2D.h"

#define FUNC(x, y) ((*f)(x, y))          /* Macro permettant de
                                            simplifier l'écriture
                                            des appels de fonctions
                                            dans la procédure. */

double trapeze_2D_interne(double (*f) (double x, double y),
                          double a, double b, double y) {

    double new_trapeze, old_trapeze = 0.0;
                                         /* variables contenant
                                            la valeur de l'estimation
                                            courante et antérieure de
                                            l'intégrale, obtenues avec
                                            la règle du trapèze. */


        /* Calculer les approximations successives de 1'intégrale
        au moyen de la règle du trapèze, en divisant par 2 les
        sous-intervalles servant à son calcul, pour chaque nouvel
        estimé de son résultat. */

    for (int i = 0; i <= N_MAX__TRAP_INTERNE; i++) {
```

```
        if (i == 0) {

                /* Calculer le premier estimé de l'intégrale au moyen de la
                règle du trapèze (obtenu avec un seul gros trapèze). */

            new_trapeze = 0.5*(b - a)*(FUNC(a, y) + FUNC(b, y));

                /* Imprimer la valeur courante (intermédiaire) de l'intégrale,
                si désiré par le programmeur (pour fin de déboguage). */

#if PRINT_INT_INTERNE
            printf("Integrale interne[%3d] = %-25.20lg\t\ty = %-lf\n", i, new_trapeze
, y);
#endif

        }

        else {

                /* La valeur courante de l'intégrale (de rang i) est
                obtenue par une règle du trapèze utilisant 2^i
                sous-intervalles. Le résultat de la règle du trapèze
                courante (de rang i) peut s'obtenir en utilisant le
                travail déjà fait pour le calcul de la règle du trapèze
                précédente (de rang i - 1). */


                  /* calcul de la somme de f aux nouveaux points
                  ajoutés. */

            double somme = 0;
            for (int k = 1; k <= ((1 << i) - 1); k = k + 2) {

                        /* Où on a utilisé le fait que 2^i = 1 << i.
                        I.e. que dans la représentation binaire
                        du nombre 1, on peut le multiplier par 2
                        simplement en lui ajoutant un zéro à sa droite.
                        (Ce qui est plus rapide que d'utiliser
                        pow(2, i)). */

                somme = somme + FUNC( a + (k*(b - a))/(1 << i) , y );
            }

                /* calcul de l'estimé courant (de rang i) de l'intégrale: */

            new_trapeze = 0.5*old_trapeze + ((b - a)/(1 << i))*somme;

                /* Imprimer la valeur courante (intermédiaire) de l'intégrale,
                si désiré par le programmeur (pour fin de déboguage). */

#if PRINT_INT_INTERNE
            printf("Integrale interne[%3d] = %-25.20lg\t\ty = %-lf\n", i, new_trapeze
, y);
#endif

                /* Si la tolérance en erreur est atteinte, retourner
                la valeur de l'intégrale courante. */

            if (i > 5) {

                    /* la condition i > 5 sert à éviter la possibilité
                    d'une (fausse) convergence trop hâtive. */

                if (  (fabs(new_trapeze - old_trapeze) < ERREUR_RELATIVE_TRAPEZE_INTE
RNE * fabs(old_trapeze))
```

```
                    || ((new_trapeze == 0) && (old_trapeze == 0))  ) {

                return new_trapeze;
            }
        }
    }

        /* La valeur courante (de rang i) de l'intégrale deviendra
        sa valeur antérieure (de rang i - 1) lors de l'itération
        suivante: */

    old_trapeze = new_trapeze;

}
    /* Fin de la boucle "for" calculant les approximations
    successives de l'intégrale. */



    /* Si l'on est rendu ici, c'est que l'on a atteint le
    nombre maximal de sous-intervalles de [a, b] permis,
    sans avoir réussi à rencontrer le critère de convergence.

    Soit que le nombre de sous-intervalles 2^N_MAX__TRAP_INTERNE
    est trop petit pour atteindre la précision voulue, ou que
    l'ERREUR_RELATIVE_TRAPEZE_INTERNE tolérée est trop petite
    par rapport aux erreurs d'arrondis dûs à l'ordinateur. */

printf("\nL'erreur relative toleree sur l'integrale faite\n");
printf("par la procedure trapeze_2D_interne() n'a pas pu\n");
printf("etre atteinte.  Modifiez N_MAX__TRAP_INTERNE\n");
printf("et/ou ERREUR_RELATIVE_TRAPEZE_INTERNE\n");

exit(1);
return 0.0;     /* Retourner une valeur double bidon car la
                procédure doit quand même retourner un double. */

}
```

```
/*****************************************************************************

  Fonction: potentiel_SWNT()

  Cette fonction calcule la valeur du potentiel de Stan et Cole pour un
  nanotube à paroi simple en fonction de la distance par rapport à l'axe
  du nanotube.

  Appel: potentiel_SWNT(r, R, theta, eps, sigma);

  r: distance par rapport à l'axe, du point où on mesure le potentiel.
  R: rayon du nanotube.
  theta: densité surfacique des atomes de C sur la paroi du nanotube.
  eps: constante d'énergie du potentiel de Lennard-Jones
  sigma: rayon caractéristique du potentiel de Lennard-Jones entre un
        atome de C et une molécule d'adsorbat.

*****************************************************************************/

#include "traitement.h"

double potentiel_SWNT(double r, double R, double theta,
                      double eps, double sigma) {

    return 3*PI*theta*eps*sigma*sigma *
           ((21.0/32.0)*(pow(sigma/R, 10))*M(r/R, 11)
           - (pow(sigma/R, 4))*M(r/R, 5));

}
```

```
/*****************************************************************

  Fichier d'en-tête:  Mn.h

  Contient les prototypes des fonctions et les définitions des
  constantes utilisées exclusivement par la fonction M(x, n).

  Il contient aussi les fichiers d'en-tête utilisés par la
  fonction M(x, n) et ses sous-fonctions, pour garder les modules
  calculant M(x, n) indépendants du reste du programme principal
  (pour fin de modularité).

*****************************************************************/

#ifndef MN_H        /* Sert à éviter l'inclusion multiple de Mn.h */
#define MN_H

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
                    /* Même si ces fichier .h standards sont
                    inclus plusieurs fois dans un fichier, ils
                    seront en fait inclus qu'une seule fois
                    par le préprocesseur. */


#define PI_Mn 3.14159265358979323846264338327950288419716939937

#define ERREUR_RELATIVE_Mn 1e-6
                /* erreur relative maximale tolérée sur la
                valeur de M(x, n). */

#define JMAX_Mn 40          /* est le nombre de raffinages MAXIMUM qu'on
                            fait sur l'intégrale de départ.
                            2^(JMAX_Mn-1) est le nombre total (maximum)
                            de sous-intervalles pour l'intégration.
                            */



    /* Constantes utilisées pour l'approximation des M(x, n)
    au moyen des séries.

    Les vecteurs de coefficients des séries ne peuvent
    pas être mis ici, car on aurait alors des
    définitions "multiples" lors de l'inclusion de
    Mn.h dans les différents fichiers sources. */

#define APPROX_SERIES_GAUCHE_DE_1 1     /* = 1 si on veut utiliser les
                                        séries pour approximer les M(x, n)
                                        sur l'intervalle [0, 1[;  = 0 sinon.
                                        */

#define APPROX_SERIES_DROITE_DE_1 1     /* = 1 si on veut utiliser les
                                        séries pour approximer les M(x, n)
                                        sur l'intervalle ]1, oo[;  = 0 sinon.
                                        */


#define X_PRECISION_DOUBLE_SERIE_M5_GAUCHE_1  0.24
                      /* i.e. où la série de M_5(x) nous donne
                      au moins 15 digits de précision pour x < 1. */
#define X_MAX_SERIE_M5_A_GAUCHE_DE_1 0.77
#define ERREUR_RELATIVE_MAX_SERIE_M5_GAUCHE_1 8.60e-8
```

```
#define X_MIN_SERIE_M5_A_DROITE_DE_1 1.30
#define ERREUR_RELATIVE_MAX_SERIE_M5_DROITE_1 8.47e-8
#define X_PRECISION_DOUBLE_SERIE_M5_DROITE_1  4.5
                      /* i.e. où la série de M_5(x) nous donne
                      au moins 15 digits de précision pour x > 1. */

#define X_PRECISION_DOUBLE_SERIE_M11_GAUCHE_1  0.28
                      /* i.e. où la série de M_11(x) nous donne
                      au moins 15 digits de précision pour x < 1. */
#define X_MAX_SERIE_M11_A_GAUCHE_DE_1 0.9999
#define ERREUR_RELATIVE_MAX_SERIE_M11_GAUCHE_1 1.70e-8
#define X_MIN_SERIE_M11_A_DROITE_DE_1 1.0001
#define ERREUR_RELATIVE_MAX_SERIE_M11_DROITE_1 1.70e-8
#define X_PRECISION_DOUBLE_SERIE_M11_DROITE_1  4.0
                      /* i.e. où la série de M_11(x) nous donne
                      au moins 15 digits de précision pour x > 1. */



    /* fonctions utilisées par M() */

double f(double phi, double x, int n);
double M(double x, int n);
double trapeze_etape_N_Mn(double (*f)(double x, double y, int n),
                          double a, double b, int N, double y, int n);
double simpson_raffinee_Mn(double (*f)(double x, double y, int n),
                           double a, double b, double y, int n);


#endif
```

```
/*****************************************************************************

  Fonction M()

  Cette fonction calcule la valeur des M(x, n) servant au calcul du
  potentiel de Stan et Cole.


  Ce module "Mn.cpp" contient la version améliorée de la fonction
  M().  Elle utilise des développements en série AMÉLIORÉS de M(x, n),
  pour éviter le plus possible l'évaluation de M(x, n) au moyen
  d'intégrations.  (DANS LE BUT DE SAUVER LE PLUS POSSIBLE DU
  TEMPS MACHINE!)

  Néanmoins, lorsque la précision demandée par l'usager est trop
  élevée pour pouvoir utiliser les développements en série de M(),
  la forme "intégrale" exacte sera utilisée.

  appel: M(x, n)

  x: distance réduite ( = r/R) du centre du nanotube
     (à paroi simple).
  n: exposant.


  NOTES:
  ------
  On a défini dans le code de cette fonction 2 constantes:
  APPROX_SERIES_GAUCHE_DE_1 et APPROX_SERIES_DROITE_DE_1
  permettant d'ignorer la représentation de M(x, n) au moyen
  des séries sur l'intervalle 0 <= x < 1 et x > 1, si désiré
  par le programmeur.  (Cela est fait au moyen de la compilation
  conditionnelle).

  Les séries, qui sont en fait des formes polynômiales,
  sont évaluées au moyen d'une méthode plus efficace que
  la méthode consistant à utiliser toujours la fonction
  pow(x, m) pour évaluer les différentes puissances de x.
  Référence:  Numerical Recipes in C, 2nd edition, p. 173.

*****************************************************************************/

#include "Mn.h"

    /* Voici les suites de coefficients des séries utilisées
    dans le calcul des M(x, n).  Étant placées à l'extérieur
    de la fonction, elles ne sont définies qu'une seule fois.
    Mais les déclarations "static" les gardent locales à ce
    fichier source. */

static double Coeff_Series_M5[] = {PI_Mn,
                                   (9.0/4.0)*PI_Mn,
                                   (9.0/64.0)*PI_Mn,
                                   (1.0/256.0)*PI_Mn,
                                   (9.0/16384.0)*PI_Mn,
                                   (9.0/65536.0)*PI_Mn,
                                   (49.0/1048576.0)*PI_Mn,
                                   (81.0/4194304.0)*PI_Mn};

static double Coeff_Series_M11[] = {PI_Mn,
                                    (81.0/4.0)*PI_Mn,
                                    (3969.0/64.0)*PI_Mn,
                                    (11025.0/256.0)*PI_Mn,
                                    (99225.0/16384.0)*PI_Mn,
                                    (3969.0/65536.0)*PI_Mn,
```

```
                                    (441.0/1048576.0)*PI_Mn,
                                    (81.0/4194304.0)*PI_Mn};


double M(double x, int n) {

    if (n == 5) {

            /* commencer par essayer de trouver M_5 au moyen
            du développement en série pour 0 <= x < 1, si
            c'est possible. */

#if APPROX_SERIES_GAUCHE_DE_1

        if (x < X_PRECISION_DOUBLE_SERIE_M5_GAUCHE_1) {

            double X2 = x*x, X_ = 1 - X2, X_2 = X_*X_, X_4 = X_2*X_2;
            return  (((((((Coeff_Series_M5[7])
                    *X2 + Coeff_Series_M5[6])
                    *X2 + Coeff_Series_M5[5])
                    *X2 + Coeff_Series_M5[4])
                    *X2 + Coeff_Series_M5[3])
                    *X2 + Coeff_Series_M5[2])
                    *X2 + Coeff_Series_M5[1])
                    *X2 + Coeff_Series_M5[0])/X_4;

        }

        if ((x < X_MAX_SERIE_M5_A_GAUCHE_DE_1) && (ERREUR_RELATIVE_Mn > ERREUR_RELATI
VE_MAX_SERIE_M5_GAUCHE_1)) {

            double X2 = x*x, X_ = 1 - X2, X_2 = X_*X_, X_4 = X_2*X_2;
            return  (((((((Coeff_Series_M5[7])
                    *X2 + Coeff_Series_M5[6])
                    *X2 + Coeff_Series_M5[5])
                    *X2 + Coeff_Series_M5[4])
                    *X2 + Coeff_Series_M5[3])
                    *X2 + Coeff_Series_M5[2])
                    *X2 + Coeff_Series_M5[1])
                    *X2 + Coeff_Series_M5[0])/X_4;

        }

#endif

            /* essayer de trouver M_5 au moyen du développement
            en série pour x > 1, si c'est possible. */

#if APPROX_SERIES_DROITE_DE_1

        if (x > X_PRECISION_DOUBLE_SERIE_M5_DROITE_1) {

            double X2 = x*x, X_ = x/(1 - X2), X_2 = X_*X_, X_4 = X_2*X_2;
            return  (X_4/x) * (((((((Coeff_Series_M5[7])
                              /X2 + Coeff_Series_M5[6])
                              /X2 + Coeff_Series_M5[5])
                              /X2 + Coeff_Series_M5[4])
                              /X2 + Coeff_Series_M5[3])
                              /X2 + Coeff_Series_M5[2])
                              /X2 + Coeff_Series_M5[1])
                              /X2 + Coeff_Series_M5[0]);

        }

        if ((x > X_MIN_SERIE_M5_A_DROITE_DE_1) && (ERREUR_RELATIVE_Mn > ERREUR_RELATI
```

```
VE_MAX_SERIE_M5_DROITE_1)) {

            double X2 = x*x, X_ = x/(1 - X2), X_2 = X_*X_, X_4 = X_2*X_2;
            return  (X_4/x) * (((((((Coeff_Series_M5[7])
                                /X2 + Coeff_Series_M5[6])
                                /X2 + Coeff_Series_M5[5])
                                /X2 + Coeff_Series_M5[4])
                                /X2 + Coeff_Series_M5[3])
                                /X2 + Coeff_Series_M5[2])
                                /X2 + Coeff_Series_M5[1])
                                /X2 + Coeff_Series_M5[0]);

        }

#endif

            /* si les développements en séries de M_5 ne sont pas
            applicables, on doit se rabattre sur le calcul de sa
            forme intégrale (plus longue à calculer). */

        return simpson_raffinee_Mn(&f, 0, PI_Mn, x, n);
    }


    if (n == 11) {

            /* commencer par essayer de trouver M_11 au moyen
            du développement en série pour 0 <= x < 1, si
            c'est possible. */

#if APPROX_SERIES_GAUCHE_DE_1

        if (x < X_PRECISION_DOUBLE_SERIE_M11_GAUCHE_1) {

            double X2 = x*x, X_ = 1 - X2, X_2 = X_*X_, X_4 = X_2*X_2,
                   X_10 = X_4*X_4*X_2;
            return  (((((((Coeff_Series_M11[7])
                    *X2 + Coeff_Series_M11[6])
                    *X2 + Coeff_Series_M11[5])
                    *X2 + Coeff_Series_M11[4])
                    *X2 + Coeff_Series_M11[3])
                    *X2 + Coeff_Series_M11[2])
                    *X2 + Coeff_Series_M11[1])
                    *X2 + Coeff_Series_M11[0])/X_10;

        }

        if ((x < X_MAX_SERIE_M11_A_GAUCHE_DE_1) && (ERREUR_RELATIVE_Mn > ERREUR_RELAT
IVE_MAX_SERIE_M11_GAUCHE_1)) {

            double X2 = x*x, X_ = 1 - X2, X_2 = X_*X_, X_4 = X_2*X_2,
                   X_10 = X_4*X_4*X_2;
            return  (((((((Coeff_Series_M11[7])
                    *X2 + Coeff_Series_M11[6])
                    *X2 + Coeff_Series_M11[5])
                    *X2 + Coeff_Series_M11[4])
                    *X2 + Coeff_Series_M11[3])
                    *X2 + Coeff_Series_M11[2])
                    *X2 + Coeff_Series_M11[1])
                    *X2 + Coeff_Series_M11[0])/X_10;

        }

#endif
```

```
            /* essayer de trouver M_11 au moyen du développement
            en série pour x > 1, si c'est possible. */

#if APPROX_SERIES_DROITE_DE_1

        if (x > X_PRECISION_DOUBLE_SERIE_M11_DROITE_1) {

            double X2 = x*x, X_ = x/(1 - X2), X_2 = X_*X_, X_4 = X_2*X_2,
                   X_10 = X_4*X_4*X_2;
            return  (X_10/x) * (((((((Coeff_Series_M11[7])
                                /X2 + Coeff_Series_M11[6])
                                /X2 + Coeff_Series_M11[5])
                                /X2 + Coeff_Series_M11[4])
                                /X2 + Coeff_Series_M11[3])
                                /X2 + Coeff_Series_M11[2])
                                /X2 + Coeff_Series_M11[1])
                                /X2 + Coeff_Series_M11[0]);

        }

        if ((x > X_MIN_SERIE_M11_A_DROITE_DE_1) && (ERREUR_RELATIVE_Mn > ERREUR_RELAT
IVE_MAX_SERIE_M11_DROITE_1)) {

            double X2 = x*x, X_ = x/(1 - X2), X_2 = X_*X_, X_4 = X_2*X_2,
                   X_10 = X_4*X_4*X_2;
            return  (X_10/x) * (((((((Coeff_Series_M11[7])
                                /X2 + Coeff_Series_M11[6])
                                /X2 + Coeff_Series_M11[5])
                                /X2 + Coeff_Series_M11[4])
                                /X2 + Coeff_Series_M11[3])
                                /X2 + Coeff_Series_M11[2])
                                /X2 + Coeff_Series_M11[1])
                                /X2 + Coeff_Series_M11[0]);

        }

#endif

            /* si les développements en séries de M_11 ne sont pas
            applicables, on doit se rabattre sur le calcul de sa
            forme intégrale (plus longue à calculer). */

        return simpson_raffinee_Mn(&f, 0, PI_Mn, x, n);
    }

        /* Si l'on est rendu ici, c'est que la valeur de n rentrée par l'usager
        est mauvaise:  ce M(x, n) n'apparaît pas dans l'expression du potentiel
        de Stan & Cole. */

    printf("\nM(x, %d) n'est pas utiliser dans le potentiel de Stan et Cole.", n);
    printf("\nPrenez n = 5 ou 11.\n");
    exit(1);
    return 0.0;

}
```

```
/*************************************************************************

  Fonction: f(phi, x, n)

  Cette fonction est la fonction dont on cherche la valeur de l'intégrale
  M(x, n) utilisée pour calculer le potentiel de Stan et Cole d'un
  nanotube à paroi simple (SWNT).

  L'INTÉGRAND POSSÈDE UNE SINGULARITÉ À x=1 et phi = 0.  Donc M(x, n)
  est non définie pour x=1. i.e. que le potentiel diverge à l'infini sur
  le nanotube.  On ne peut pas calculer V(r, R) pour r = R.

  Appel: F = f(phi, x, n);

    phi: variable d'intégration variant sur l'intervalle [0, PI]
    x: = r/R distance "réduite" de l'axe du nanotube (SWNT).
    n: exposant apparaissant dans l'intégrand.

*************************************************************************/

#include "Mn.h"

double f(double phi, double x, int n) {

    return pow((1 + x*x - 2*x*cos(phi)) , -(n/2.0));

}
```

```
/*****************************************************************************

  Fonction: simpson_raffinee_Mn()

  Cette fonction utilise la fonction trapeze_etape_N_Mn() pour calculer
  l'intégrale de f(x, y, n) par rapport à x sur l'intervalle [a, b].
  L'erreur relative permise sur l'intégrale est de ERREUR_RELATIVE_Mn.

  Cette fonction calcule l'intégrale par améliorations successives.
  Elle calcule une valeur améliorée à partir de la valeur précédente
  (qui est moins exacte).  JMAX_Mn est le nombre maximal d'améliorations
  qu'on permet pour le calcul de l'intégrale finale.  Donc 2^(JMAX_Mn - 1)
  est le nombre total (maximal) de sous-intervalles qu'on peut avoir
  pour le calcul de l'intégrale.

  utilisant l'équation (voir Numerical Recipes):
        S = (4/3)*S[2N] - (1/3)*S[N]
  on peut calculer l'intégrale de Simpson à partir d'intégrales "trapèze"
  sans complications supplémentaires.

  Appel: integrale = simpson_raffinee_Mn(f, a, b, y, n);

        f: f(x, y, n) la fonction à intégrer par rapport à x.
        a: borne gauche de l'intervalle d'intégration.
        b: borne droite de l'intervalle d'intégration.
        y: deuxième variable gardée fixe.
        n: exposant apparaissant dans f (gardé fixe).

*****************************************************************************/

#include "Mn.h"

double simpson_raffinee_Mn(double (*f)(double x, double y, int n),
                           double a, double b, double y, int n) {

    double integrale, old_integrale, integrale_trapeze,
           old_integrale_trapeze;
                           /* contient les integrales (actuelles et
                           anciennes) de Simpson et du trapèze. */

    old_integrale_trapeze = old_integrale = -1.0e30;
                           /* valeurs impossibles pour les vieilles
                           intégrales.  Ça permet de commencer la
                           boucle des "améliorations". */

    for (int j = 1; j <= JMAX_Mn; j++) {
            /* faire un nombre d'améliorations <= JMAX_Mn. */
        integrale_trapeze = trapeze_etape_N_Mn(f, a, b, j, y, n);
            /* calculer la jème approximation de l'intégrale de Simpson. */
        integrale = (4.0*integrale_trapeze - old_integrale_trapeze)/3.0;

        if (j > 5) {
                /* la condition j > 5 évite une (fausse) convergence
                trop hâtive. */
            if ((fabs(integrale - old_integrale)  < ERREUR_RELATIVE_Mn * fabs(old_int
egrale))
                || (integrale == 0.0 && old_integrale == 0.0)) {
                    /* on a obtenu une valeur assez précise
                    de l'intégrale. */
                return integrale;
            }
        }

                /* les intégrales actuelles seront les "vieilles"
                intégrales de la prochaine approximation. */
```

```
            old_integrale = integrale;
            old_integrale_trapeze = integrale_trapeze;
        }

            /* Si on est rendu ici, c'est qu'on a pas réussi à converger.
            Ça peut être parce que le nombre d'étapes JMAX_Mn est trop petit
            (i.e. que le nombre d'améliorations maximal permis n'est pas
            assez grand pour atteindre la précision voulue). Ou que
            l'ERREUR_RELATIVE_Mn tolérée  est trop petite par rapport aux
            erreurs d'arrondis dûs à l'ordinateur (cause plus probable). */
        printf("\nL'erreur relative toleree sur l'integrale faite\n");
        printf("par la procedure simpson_raffinee_Mn() est trop\n");
        printf("petite pour avoir convergence.\n");
        exit(1);
        return 0.0; /* retourner une valeur double bidon car la fonction
                       doit retourner quand même un double. */

}
```

```
/*************************************************************************

  Fonction: trapeze_etape_N_Mn()

    Cette fonction retourne la Nème valeur raffinée de l'intégrale
    de f(x, y, n) sur [a, b] à partir de la (N-1)ème valeur (i.e.
    à partir de sa valeur précédente.  L'intégration est faite par
    rapport à x, tandis que y et n sont gardées fixes.  La procédure
    a pour principe la règle du trapèze étendu (voir Numerical Recipes).

    Le premier appel calcule une intégrale grossière: un seul gros
    trapèze sur tout l'intervalle [a, b].  Les itérations suivantes
    N = 2, 3, ... rajoutent 2^(N-2) points de calculs au résultat
    précédent.

    Cette procédure a l'avantage de ne pas "jeter à la poubelle" le
    résultat précédent pour calculer une intégrale plus précise, mais
    d'utiliser le travail déjà fait.

  Appel: integrale = trapeze_etape_N_Mn(f, a, b, N, y, n);

        f: f(x, y, n) la fonction à intégrer par rapport à x.
        a: borne gauche de l'intervalle d'intégration.
        b: borne droite de l'intervalle d'intégration.
        N: indique qu'on est à la Nème étape de "raffinement".
        x: variable d'intégration.
        y: deuxième variable gardée fixe.
        n: exposant apparaissant dans f (gardé fixe).

*************************************************************************/

#include "Mn.h"

double trapeze_etape_N_Mn(double (*f)(double x, double y, int n),
                          double a, double b, int N, double y, int n) {

    double x, somme, delta;     /* somme est la somme de la nouvelle
                                   série de points à ajouter.
                                   delta est l'espace entre 2 de ces points.
                                   x est la position (abscisse) du point
                                   intermédiaire qu'on considère.
                                   */
    static double integrale;    /* contient présentement la (N-1)ème
                                   approximation de l'intégrale. (contiendra
                                   la Nème approximation à la fin de la
                                   procédure).
                                   */

    if (N == 1) {
            /* on calcule alors la première approximation très grossière
               donnée par un seul gros trapèze englobant tout l'intervalle
               [a, b]. */
        integrale = 0.5*(b-a)*((*f)(a, y, n) + (*f)(b, y, n));
        return integrale;
    }

    else {
            /* calculer la Nème approximation à partir de l'approximation
               précédente. */
        delta = (fabs(b - a))/(1 << (N-2));
                /* Car pow(2, N-2) = 1 << (N-2).  Cette dernière forme
                   étant plus rapide à calculer. */
        x = a + delta/2;    /* la position du premier point intermédiaire
                               à ajouter. */
```

```
            /* additionner les valeurs de la fonction f prises aux points
               intermédiaires x[j] = x + (j-1)*delta. j allant de 1 à 2^(N-2). */
            somme = 0;
            for (int j = 1; j <= (1 << (N-2)); j++) {
                somme = somme + (*f)(x, y, n);
                x = x + delta;  /* passer au prochain point intermédiaire. */
            }

            /* calculer la Nème approximation de l'intégrale à partir
               de la (N-1)ème approximation. */
            integrale = 0.5*(integrale + delta*somme);
            return integrale;
        }
    }
```

```
/*************************************************************

  Fonction:  matrice_Bas()

  Cette fonction génère une matriceBas[2][Nbre_x] contenant
  Nbre_x valeurs de second coefficient du viriel Bas,
  pour chaque valeur de x.  Ici, x pourra être soit un vecteur
  contenant une série de températures ou un vecteur contenant
  une série de rayons R des parois des SWNTs ou un vecteur
  contenant une série de pas d du réseau de nanotubes.

  Appel:  matriceBas = matrice_Bas(x_min, x_max, Nbre_x,
                           T, R, MatriceReseau, R_reseau,
                           N_reseau, L, theta, eps, sigma,
                           T_ou_R_ou_d, flag_fermer);


    matriceBas:     matrice contenant les Nbre_x valeurs x[i] et
                        les valeurs Bas[i] correspondantes.
    x_min:          la valeur minimale de x (peut être T_min
                        ou R_min ou d_min).
    x_max:          la valeur maximale de x.
    Nbre_x:         le nombre de valeurs de x pour lesquelles
                        on calcule les Bas correspondants.
    T:              température du gaz (en K).
    R:              rayon des nanotubes SWNT du bundle
                        (en Angstrom).
    MatriceReseau:  matrice contenant les points du réseau
                        (centres des nanotubes).
    R_reseau:       le rayon du plus petit cercle englobant
                        tous les points du réseau.
    N_reseau:       le nombre de points du réseau.
    L:              longueur des nanotubes SWNT formant le
                        bundle (en Angstrom).
    theta:          densité surfacique des atomes de C sur les
                        parois des nanotubes SWNT du bundle.
    eps:            constante d'énergie du potentiel de
                        Lennard-Jones.
    sigma:          rayon caractéristique du potentiel de
                        Lennard-Jones entre un atome de C et une
                        molécule d'adsorbat.
    T_ou_R_ou_d:    variable (= 1 pour T, 2 pour R, 3 pour d)
                        disant si x est un vecteur de
                        températures ou un vecteur de rayons R
                        ou un vecteur de pas d du réseau.
    flag_fermer:    flag disant si les nanotubes à parois simples
                        du bundle sont OUVERTS ou FERMÉS
                        ( = 0 pour OUVERTS; != 0 pour FERMÉS).


  NOTE:
  -----
    Cette fonction utilise une constante symbolique
    PRINT_PROGRESSION_CALCUL_Bas permettant au programmeur
    d'afficher à l'écran (si il le désire) le nombre de
    valeurs de Bas qui ont déjà été calculées.  Cela
    permet de savoir à quel point la matriceBas est remplie
    à un instant donné (si PRINT_PROGRESSION_CALCUL_Bas != 0,
    imprimer la progression; si  = 0, ne pas imprimer la
    progression du calcul).

*************************************************************/

#include "traitement.h"
```

```
double** matrice_Bas(double x_min, double x_max, int Nbre_x,
                double T, double R, double** MatriceReseau,
                double R_reseau, int N_reseau, double L,
                double theta, double eps, double sigma,
                int T_ou_R_ou_d, int flag_fermer) {

        /* espacement entre 2 valeurs x[i] */

    double delta_x = (fabs(x_max - x_min))/(Nbre_x - 1);

        /* formation de la matriceBas contenant les Bas */

    double** matriceBas = (double**) malloc(2 * sizeof(double*));
    if (matriceBas == NULL) {
        printf("\nErreur d'allocation de memoire\n");
        exit(1);
    }

    for (int i = 1; i <= 2; i++) {

            /* allouer la mémoire pour les 2 lignes de la
            matriceBas:  x et Bas. */

        matriceBas[i -1] = (double*) malloc(Nbre_x * sizeof(double));
        if (matriceBas[i -1] == NULL) {
            printf("\nErreur d'allocation de memoire\n");
            exit(1);
        }
    }


        /* remplir la matriceBas des valeurs de x et de Bas
        correspondantes (pour un bundle de SWNTs OUVERTS
        ou FERMÉS aux extrémités). */

    for (i = 1; i <= Nbre_x; i++) {

#if PRINT_PROGRESSION_CALCUL_Bas
        if (i == 1) {
            printf("\nNombre de Bas deja calcules: ");
        }
#endif

        double x = x_min + (i - 1)*delta_x;
        double Bas;
        if (T_ou_R_ou_d == 1) {

                /* T est alors variable. */

                /* Calculer le second coefficient du viriel.
                Tous les autres paramètres (fixés) nécessaires
                à son calcul ont été passés comme arguments à
                la procédure matrice_Bas(). */

            Bas = Bas_bundle_SWNT(x, R, L, MatriceReseau,
                            N_reseau, R_reseau, flag_fermer,
                            theta, eps, sigma);

        }
        else {

            if (T_ou_R_ou_d == 2) {

                    /* R est alors variable. */
```

```
                    /* Calculer le second coefficient du viriel.
                    Tous les autres paramètres (fixés) nécessaires
                    à son calcul ont été passés comme arguments à
                    la procédure matrice_Bas(). */

                Bas = Bas_bundle_SWNT(T, x, L, MatriceReseau,
                                N_reseau, R_reseau, flag_fermer,
                                theta, eps, sigma);
            }

            else {

                    /* le pas d du réseau est alors variable. */

                    /* Générer le réseau de pas d (des centres des
                    nanotubes SWNT du bundle). */

                MatriceReseau = GenererReseau(x, N_reseau, &R_reseau);

                    /* Calculer le second coefficient du viriel.
                    Tous les autres paramètres (fixés) nécessaires
                    à son calcul ont été passés comme arguments à
                    la procédure matrice_Bas(). */

                Bas = Bas_bundle_SWNT(T, R, L, MatriceReseau,
                                N_reseau, R_reseau, flag_fermer,
                                theta, eps, sigma);

                    /* Libérer la mémoire de la MatriceReseau des
                    points du réseau (variable) de pas d = x avant
                    de calculer Bas pour une prochaine valeur de d. */

                liberermatrice(MatriceReseau);

            }
        }

        matriceBas[l -1][i -1] = x;
        matriceBas[2 -1][i -l] = Bas;

                /* Je préfère compter les indices à partir de
                1, mais le C compte à partir de 0.  Je fais
                donc toujours une correction systématique de
                -1 aux indices des vecteurs et matrices. */

#if PRINT_PROGRESSION_CALCUL_Bas
        printf("%d  ", i);
#endif

    }

#if PRINT_PROGRESSION_CALCUL_Bas
    printf("\n");
#endif

    return matriceBas;
}
```

```
/*************************************************************

  Fonction: Ecrire_Bas()

  Cette fonction écrit dans un fichier
  "Bas_bundle_SWNTs_OUVERT_T.txt" ou
  "Bas_bundle_SWNTs_FERMER_T.txt" ou
  "Bas_bundle_SWNTs_OUVERT_R.txt" ou
  "Bas_bundle_SWNTs_FERMER_R.txt" ou
  "Bas_bundle_SWNTs_OUVERT_d.txt" ou
  "Bas_bundle_SWNTs_FERMER_d.txt" les valeurs contenues dans la
  matriceBas pour les seconds coefficients du viriel Bas
  calculés aux différentes valeurs de x (i.e T ou R ou d).
  Les nanotubes SWNT du bundle peuvent être OUVERTS ou
  FERMÉS à leurs extrémités.  Le fichier contient également
  les autres paramètres pour lesquels ces Bas ont été calculés
  (i.e. N_reseau; L; theta; eps; sigma; et deux des paramètres
  suivants selon le cas: R; T ; d.)


  Appel:  Ecrire_Bas(matriceBas, Nbre_x, T, R, d, R_reseau,
                     N_reseau, L, theta, eps, sigma,
                     T_ou_R_ou_d, flag_fermer);


    matriceBas:     matrice contenant les Nbre_x valeurs x[i] et
                        les valeurs Bas[i] correspondantes.
    Nbre_x:         le nombre de valeurs de x pour lesquelles
                        on calcule les Bas correspondants.
    T:              température du gaz (en K).
    R:              rayon des nanotubes SWNT du bundle
                        (en Angstrom).
    d:              distance (en Angstroms) séparant 2 points
                        du réseau (centre des nanotubes à
                        parois simples).
    R_reseau:       le rayon du plus petit cercle englobant
                        tous les points du réseau.
    N_reseau:       le nombre de points du réseau. (i.e. le
                        nombre de SWNTs formant le bundle).
    L:              longueur des nanotubes SWNT formant le
                        bundle (en Angstrom).
    theta:          densité surfacique des atomes de C sur les
                        parois des nanotubes SWNT du bundle.
    eps:            constante d'énergie du potentiel de
                        Lennard-Jones.
    sigma:          rayon caractéristique du potentiel de
                        Lennard-Jones entre un atome de C et une
                        molécule d'adsorbat.
    T_ou_R_ou_d:    variable (= 1 pour T, 2 pour R, 3 pour d)
                        disant si x est un vecteur de
                        températures ou un vecteur de rayons R
                        ou un vecteur de pas d du réseau.
    flag_fermer:    flag disant si les nanotubes à parois simples
                        du bundle sont OUVERTS ou FERMÉS
                        ( = 0 pour OUVERTS; != 0 pour FERMÉS).

*************************************************************/

#include "traitement.h"

void Ecrire_Bas(double** matriceBas, int Nbre_x, double T,
                double R, double d, double R_reseau, int N_reseau,
                double L, double theta, double eps, double sigma,
                int T_ou_R_ou_d, int flag_fermer) {
```

```
    /* définition d'une chaîne OUV_FERM contenant le mot
    OUVERT ou FERMER selon que les nanotubes (à parois
    simples) du bundle sont ouverts ou fermés aux extrémités.
    Ce mot "variable" sera substitué dans les noms de
    fichiers et dans le texte contenu dans ceux-ci. */

char* OUV_FERM;
if (flag_fermer == 0) {
        /* les nanotubes sont ouverts aux extrémités. */

    OUV_FERM = "OUVERT";
}
else {
    OUV_FERM = "FERMER";
}


    /* Fabrication de la chaîne de caractères contenant
    le nom du fichier, selon que R ou T ou d est le
    paramètre variable en fonction duquel on calcule Bas
    et selon que les nanotubes du bundle sont OUVERTS ou
    FERMÉS aux extrémités. */

char file_name[35] = "Bas_bundle_SWNTs_";

        /* file_name doit être assez longue pour contenir
        la chaîne contenant le nom de fichier final. */

if (T_ou_R_ou_d == 1) {
    strcat(strcat(file_name, OUV_FERM), "_T.txt");
}
else if (T_ou_R_ou_d == 2) {
    strcat(strcat(file_name, OUV_FERM), "_R.txt");
}
else {
    strcat(strcat(file_name, OUV_FERM), "_d.txt");
}


    /* Création du fichier contenant les valeurs de Bas pour
    les différentes valeurs de x (dont le nom a été formé
    ci-haut). */

FILE* fptr;
fptr = fopen(file_name, "w");
if (fptr == NULL) {
    printf("\nErreur d'ouverture de fichier\n");
    exit(1);
}


fprintf(fptr, "Fichier %c%s%c donnant le second\n", 34, file_name, 34);
fprintf(fptr, "coefficient du viriel Bas, pour un bundle de SWNTs\n");
fprintf(fptr, "%s a leurs extremites.\n\n\n", OUV_FERM);


    /*  Mettre la valeur des paramètres physiques qui sont
    gardés fixes au début du fichier, avant de mettre les
    valeurs de Bas.

    On commence par écrire les paramètres
    qui sont TOUJOURS fixes, peu importe que ce soit R, T,
    ou d qui varie. */
```

```
fprintf(fptr, "Nombre de nanotubes a parois simples formant le bundle:\n");
fprintf(fptr, "N_reseau = %d\n\n", N_reseau);
fprintf(fptr, "Longueur des nanotubes du bundle:\n");
fprintf(fptr, "L (angstrom) = %.10lg\n\n", L);

fprintf(fptr, "Densite surfacique des atomes formant les parois:\n");
fprintf(fptr, "theta (angstrom^-2) = %lg\n\n", theta);

fprintf(fptr, "Valeurs des parametres du potentiel de Lennard-Jones:\n");
fprintf(fptr, "epsilon (K) = %lg\t\tsigma (angstrom) = %lg\n\n", eps, sigma);

    /* Écrire dans le fichier les deux paramètres parmis
    les suivants, qui sont gardés fixes lors du calcul des Bas:
    R, T, (d et R_reseau). */

if (T_ou_R_ou_d != 1) {

    fprintf(fptr, "Temperature de l'adsorbat (K):\n");
    fprintf(fptr, "T = %.10lg\n\n", T);
}

if (T_ou_R_ou_d != 2) {

    fprintf(fptr, "Rayon des SWNTs du bundle (angstrom):\n");
    fprintf(fptr, "R = %.10lg\n\n", R);

}

if ((T_ou_R_ou_d == 1) || (T_ou_R_ou_d == 2)) {

    fprintf(fptr, "Distance entre les centres des nanotubes (angstrom):\n");
    fprintf(fptr, "d = %.15lg\n\n", d);
    fprintf(fptr, "Rayon du plus petit cercle englobant tous les centres\n");
    fprintf(fptr, "des nanotubes (angstrom):\n");
    fprintf(fptr, "R_reseau = %.15lg\n\n", R_reseau);

}


    /* Mettre la valeur de Bas[i] correspondant à chaque valeur
    x[i] dans le fichier.  (i.e. mettre les éléments de matriceBas
    dans le fichier, en faisant une validation de l'écriture dans le
    fichier pour chaque valeur de Bas ajoutée.
    */

fprintf(fptr, "Nombre de valeurs de Bas calculees: %d\n\n\n", Nbre_x);

fprintf(fptr, "Valeurs du second coefficient du viriel Bas pour chaque\n");

if (T_ou_R_ou_d == 1) {
    fprintf(fptr, "temperature de la plage consideree:\n");
    fprintf(fptr, "T (K)\t\t\tBas (angstrom^3)\n");
}

else {

    if (T_ou_R_ou_d == 2) {
        fprintf(fptr, "valeur du rayon R des SWNTs de la plage consideree:\n");
        fprintf(fptr, "R (angstrom)\t\tBas (angstrom^3)\n");
    }

    else {
        fprintf(fptr, "valeur de pas d du reseau dans la plage consideree:\n");
        fprintf(fptr, "d (angstrom)\t\tBas (angstrom^3)\n");
    }
```

```
    }

    for (int i = 1; i <= Nbre_x; i++) {
        if (fprintf(fptr, "%-13.6lg                %-13.6lg\n",
            matriceBas[1 -1][i -1], matriceBas[2 -1][i -1]) < 0) {

            printf("\nErreur d'ecriture dans le fichier\n");
            exit(1);
        }
    }

        /* Fermer le fichier avant de quitter la fonction. */

    fclose(fptr);
}
```

```
/*************************************************************************

  Fonction: GenererReseau()

  Cette fonction génère un réseau hexagonal (constitué des centres des
  nanotubes) comportant "N_reseau" nanotubes.  Elle met les points
  (X[i], Y[i]) calculés dans une matrice à 2 rangées
  MatriceReseau[2][N_reseau].  Elle retourne également la valeur du
  rayon "R_reseau" du plus petit cercle englobant tous les centres des
  nanotubes.

  L'idée de la procédure est de tourner dans le sens antihoraire sur
  chaque couche hexagonale pour générer le réseau, jusqu'à ce que l'on
  aie plus de points à placer.   i.e. que l'on fait en gros une spirale
  dans le sens antihoraire.

  Appel: MatriceReseau = GenererReseau(d, N_reseau, &R_reseau);

  d:        distance séparant 2 points du réseau (centre des nanotubes).
  N_reseau: nombre de points à placer pour faire le réseau.
  R_reseau: le rayon du plus petit cercle englobant tous les points
            du réseau.


  ------------------------------------------------------------------
  Explication des différents indices utilisés dans la fonction:

    i: compte le nombre de couches (i = 1 à infini).
    j: compte le nombre de points du réseau déjà placés.
    c: compte le nombre de cotés remplis (pour une couche donnée).
    k1: compte le nombre de points mis sur le coté c.
  ------------------------------------------------------------------


*************************************************************************/

#include "traitement.h"

double** GenererReseau(double d, int N_reseau, double *R_reseau) {

        /* Allouer de la mémoire dynamique pour les vecteurs X et Y contenant
        les points du réseau. */

    double* X = (double*) malloc(N_reseau * sizeof(double));
    if (X == NULL) {
        printf("\nErreur d'allocation de memoire\n");
        exit(1);
    }

    double* Y = (double*) malloc(N_reseau * sizeof(double));
    if (Y == NULL) {
        printf("\nErreur d'allocation de memoire\n");
        exit(1);
    }

        /* Déclarer la matrice contenant les points du réseau et y "attacher"
        les 2 vecteurs X et Y. */

    double** MatriceReseau = (double**) malloc(2 * sizeof(double*));
    if (MatriceReseau == NULL) {
        printf("\nErreur d'allocation de memoire\n");
        exit(1);
    }

    MatriceReseau[1 -1] = X;
```

```
MatriceReseau[2 -1] = Y;
            /* Apporter une correction systématique de -1 aux indices
            des vecteurs et matrices, car on numérote de 1 à n dans
            notre tête tandis que le langage C numérote de 0 à n-1. */


    /* Placer le point central (0, 0) du réseau manuellement. */
int j = 1;
X[1 -1] = 0;
Y[1 -1] = 0;

    /* Ajuster le rayon R_reseau du plus petit cercle englobant
    le réseau. */
*R_reseau = 0.0;

if (j == N_reseau) {
        /* tous les points sont alors placés, sortir de la fonction. */
    return MatriceReseau;
}



    /* Remplir chaque couche tant qu'il y a des points à placer (i.e.
    tant que j < N_reseau). */

for (int i = 1; ; i++) {
                /* i n'arrête pas la boucle, mais j le fait. */

        /* incrémenter j de 1, avant de placer le point suivant. */
    j++;

        /* Placer le point de départ de la couche i,
        le point (i*d, 0). */
    X[j -1] = i * d;
    Y[j -1] = 0;

        /* Ajuster le rayon R_reseau du plus petit cercle englobant
        le réseau. */
    *R_reseau = X[j -1];

        /* Si tous les pts sont placés, quitter la fonction. */
    if (j == N_reseau) return MatriceReseau;


        /* Placer i points sur chacun des 5 premiers côtés et i-1 points
        sur le 6ème côté de l'hexagone. */
    for (int c = 1; c <= 6; c++) {

            /* Ajuster l'angle theta donnant la direction du parcours,
            selon le côté considéré. */
        double theta = PI/3 + c*(PI/3);

            /* calculer le nombre de points à mettre sur le côté c. */
        int NbrePtsSurC;
        if (c <= 5) NbrePtsSurC = i;
        else NbrePtsSurC = i-1;

            /* Remplir le côté c avec des points du réseau. */
        for (int k1 = 1; k1 <= NbrePtsSurC; k1++) {

                /* Augmenter j de 1 avant d'ajouter le point suivant au
                réseau. */
            j++;

                /* Ajouter un déplacement au point du réseau précédent
```

```
                    afin de placer le point "j" courant. */
                X[j -1] = X[j-1 -1] + d*cos(theta);
                Y[j -1] = Y[j-1 -1] + d*sin(theta);

                if (j == N_reseau) return MatriceReseau;

            }

        }
    }
}
```

```
/*************************************************************************

  Fonction: ImprimerReseau()

  Cette fonction prend la matrice MatriceReseau[2][N_reseau]
  contenant les points (X[i], Y[i]) du réseau et les mets dans un
  fichier "PointsReseau.txt".  Cela permet de faire un graphique pour
  afficher les points du réseau.

  Appel: ImprimerReseau(MatriceReseau, N_reseau, d, R_reseau,
                        flag_reseau_variable);

  MatriceReseau: matrice contenant les points du réseau (centres des
                 nanotubes).
  N_reseau:      le nombre de points du réseau.
  d:             distance entre les centres des nanotubes.
  R_reseau:      le rayon du plus petit cercle englobant tous les points
                 du réseau.
  flag_reseau_variable:  flag indiquant à cette procédure d'impression
                 que les paramètres déterminant le réseau (d et/ou
                 N_reseau) sont variables.  Dans ce cas là, il n'y a
                 aucun intérêt à imprimer les points du réseau.

*************************************************************************/

#include "traitement.h"

void ImprimerReseau(double** MatriceReseau, int N_reseau, double d,
                    double R_reseau, int flag_reseau_variable) {

    FILE* fptr = fopen("PointsReseau.txt", "w");

    if (fptr == NULL) {
        printf("\nErreur d'ouverture de fichier\n");
        exit(1);
    }

    if (flag_reseau_variable != 0) {
        fprintf(fptr, "Lors du dernier cas calculer, les parametres determinant\n");
        fprintf(fptr, "le reseau etaient variables.  Il n'y avait donc pas de reseau\
n");
        fprintf(fptr, "unique qu'on pouvait faire imprimer.\n");
    }

    else {
        fprintf(fptr, "Fichier \"PointsReseau.txt\" contenant les positions [X, Y]\n"
);
        fprintf(fptr, "des points du reseau.\n\n");
        fprintf(fptr, "Reseau de %d nanotubes separes d'une distance de %lg angstroms
\n\n",
                N_reseau, d);

        fprintf(fptr, "Le rayon \"R_reseau\" du plus petit cercle englobant tous les
points\n");
        fprintf(fptr, "du reseau est de %lg angstroms\n\n", R_reseau);

        fprintf(fptr, "Coordonnees [X, Y] des points du reseau:\n");

            /* Mettre les points du réseau dans le fichier. */
        for (int i = 1; i <= N_reseau; i++) {

                /* Impression des points avec validation de l'écriture */
            if (fprintf(fptr, "%-13lg\t%-13lg\n", MatriceReseau[1 -1][i -1],
                MatriceReseau[2 -1][i -1]) < 0) {
```

```
                printf("\nErreur d'ecriture dans le fichier\n");
                exit(1);

            }
        }
    }

        /* Fermer le fichier avant de quitter la fonction. */
    fclose(fptr);
}
```

```
/*************************************************************************

  Fonction: liberermatrice()

  Cette fonction libère la mémoire d'une matrice à 2 dimensions contenant
  les points (X, Y)

  Appel: liberermatrice(matrice);

  matrice: la matrice[2][nbrePts dans les vecteurs X et Y] à libérer

*************************************************************************/

#include "traitement.h"

void liberermatrice(double** matrice) {

        /* libérer les 2 lignes */
    free(matrice[1 -1]);
    free(matrice[2 -1]);

        /* libérer la matrice contenant l'adresse des 2 lignes déjà
        libérées */
    free(matrice);

}
```