# Poly de TP pour LU3PY126 PAD “Projet numérique”, version C++

13 janvier 2026

Sorbonne Université - L3 Physique

# Table des matières

# TP 0

# Marches aléatoires

## 0.1 Nombres aléatoires

Pour simuler informatiquement un processus aléatoire, il est nécessaire d'utiliser un générateur qui fournit une suite de nombres s'approchant le plus possible d'un «vrai» hasard. Par vrai hasard on entend par exemple le fait que la suite produite ne doit pas être périodique et que les nombres produits doivent être uniformement répartis.

On utilise en pratique des algorithmes qui produisent une suite périodique, mais avec un période très grande : par exemple $m = 2^{31} - 1$ pour la fonction `rand()` que nous utiliserons. Un exemple d'algorithme utilisé pour générer une suite $X_n$ de nombres est :

$$X_{n+1} = (aX_n + c) \bmod m$$

où le modulo $m$ détermine la période maximale de la suite et $X_0$ est la «graine», le premier terme de la suite.

Un exemple d'appel à la fonction s'écrit :

```
#include <cstdlib> // bibliothèque nécessaire pour utiliser rand()
int n;
srand(12345); // initialise la suite avec la graine X0=12345
n = rand(); // renvoie un entier pseudo−aléatoire avec 0 ⩽ n ⩽ RAND_MAX
            // sous linux, RAND_MAX = 2^31 − 1 = 2 147 483 647
n = rand() % 2; // n vaut 0 ou 1 avec des probabilités égales (0.5)
n = rand() % 6; // n vaut 0,1,2,3,4 ou 5 avec des probabilités quasi−égales (1/6)
```

**1.** La fonction `rand()` renvoie la même suite de nombres à chaque exécution si on ne change pas la graine $X_0$. Essayez de comprendre pourquoi il est souhaitable et important, en programmation scientifique, que les séquences de nombres aléatoires ne changent pas d'une exécution à l'autre du programme.

En général, plutôt que de tirer des nombres entiers, on a très souvent besoin d'une fonction qui fournisse un nombre aléatoire flottant compris dans l'intervalle $[0; 1[$ (zéro inclus, un exclu). La fonction `alea()` définie ci-dessous remplit ce rôle :

```
double alea(void) {
  double x = rand()/(RAND_MAX+1.0); // x contient un flottant aléatoire avec 0 ⩽ x < 1
  return x;
}
```

À l'aide de la fonction `alea()`, il est facile d'obtenir un nombre flottant ou entier dans l'intervalle que l'on souhaite :

```
double a;
a = alea ();              // a contient un flottant aléatoire avec 0 ⩽ a < 1

double b;
b = 3.*alea () + 6.5;  // b contient un flottant aléatoire avec 6.5 ⩽ b < 9.5

int c;
c = (int)(100. * alea ());    // c contient un entier avec 0 ⩽ c ⩽ 99
```

**2.** Comment feriez-vous pour tirer au sort un nombre flottant dans un intervalle donné $[min, max[$ ? et un nombre entier dans le même intervalle ? *(Pour se familiariser avec le concept de fonction, essayez d'ecrire la réponse sous la forme d'une fonction).*

## 0.2 Marches aléatoires

Les marches aléatoires sont le point de départ de nombreuses modélisations physiques décrivant le mouvement brownien, la diffusion, les polymères, etc. Elles ont été introduites en physique en 1905 par Albert Einstein.

### 0.2.1 Marches aléatoires à pas discrets

Le modèle le plus simple d'une marche aléatoire a une dimension est celui d'une particule pouvant se déplacer par pas discrets ($\pm 1$) qui s'effectuent au hasard à droite ou à gauche. La particule part de l'origine ($x = 0$) et la marche s'arrête au bout d'un nombre $n$ de pas. La marche est dite isotrope si la probabilité d'aller à droite est la meme $p = q = 1/2$ que celle d'aller à gauche.

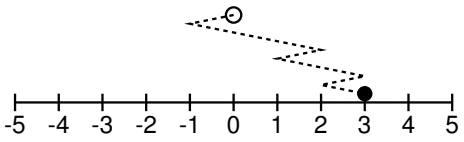



FIGURE 0.1 – Une marche aléatoire discrète a 1D.

Afin de caractériser les propriétés statistiques de ce modèle, il est nécessaire d'effectuer un grand nombre de marches, toutes du même nombre $n$ de pas.

**3.** Écrivez un programme qui fait la simulation d'**une** marche de 10 pas, en affichant la position en fonction du temps, ainsi que la position finale.

**4.** Écrire le même programme en définissant une **fonction** `int marche(int npas)` qui effectue une marche aléatoire de `npas` pas et renvoie la position finale.

**5.** Généralisez ce programme pour effectuer un nombre `NMARCHES` de marches de 10 pas chacune. Chaque marche est indépendante de la précédente, et redémarre donc depuis l'origine. Calculez la moyenne sur toutes ces `NMARCHES` marches de la position d'arrivée du marcheur, ainsi que son écart type. On pourra utiliser les definitions suivantes de la moyenne et de l'écart type :

$$< x >= \frac{1}{N} \sum_{i=1}^{N} x_i \qquad \sigma = \sqrt{< x^2 > - < x >^2}$$

**6.** Faîtes les mêmes simulations pour un nombre de pas variant de $n = 50$ jusqu'a $n = 6400$ en doublant le nombre de pas à chaque fois : c'est-à-dire effectuez `NMARCHES` de $n = 50$ pas, en calculant les mêmes quantités statistiques qu'au point précédent, puis `NMARCHES` de $n = 100$ pas, etc...

**7.** Adaptez éventuellement le nombre `NMARCHES` de marches pour chaque $n$ afin d'avoir une bonne statistique.
Utilisez gnuplot pour tracer ces quantités en fonction du nombre de pas $n$ et essayez de deviner leur loi de variation.

Il est relativement simple de prédire la loi de variation de la position $x_n$ au bout de n pas : on peut considérer que $x_n$ est une somme de $n$ variables aléatoires $u$ qui chacune peuvent prendre la valeur $\pm 1$ avec une probabilité égale $p = 1/2$. La variable aléatoire $u$ a une valeur moyenne nulle

$$< u >= \sum_i p_i u_i = 1/2 \times -1 + 1/2 \times 1 = 0$$

et une variance

$$\sigma_u^2 =< u^2 > - < u >^2 = \sum_i p_i u_i^2 = 1.$$

La position moyenne $< x_n >$ après $n$ pas peut se calculer en utilisant la linéarité de la moyenne :

$$< x_n > = < \sum_{i=1}^{n} u_i > = \sum_{i=1}^{n} < u_i > = 0.$$

De la même façon, sachant que $< x_n >= 0$ et que les $u_i$ sont indépendant entre eux, on a pour la variance de $x$ :

$$\sigma_n^2 =< x^2 > - < x >^2 = < (\sum_{i=1}^{n} u_i)^2 > = < \sum_{i=1}^{n} u_i^2 > + < \sum_i \sum_{j \neq i} u_i u_j > = n\sigma_u^2 = n.$$

On trouve donc que la position moyenne au bout de $n$ pas est nulle, et que son écart type, soit en quelque sorte la largeur typique de la distribution, vaut $\sqrt{n}$.

On souhaite maintenant évaluer numériquement la loi de probabilité $p(n, x)$ suivie par la variable $x_n$, indiquant la probabilité de se trouver en $x_n = x$ au bout de $n$ pas.

**8.** Quelle sont les positions $(x_{min}, x_{max})$ minimales et maximales possibles pour une marche de $n$ pas ?

**9.** Si on souhaite associer à chaque valeur possible de $x_n$ une probabilité, combien d'éléments `nval` doit posséder le tableau `double proba[nval]` qui servira à les stocker ?

Pour estimer la loi $p(n, x)$, il faut modifier légèrement votre programme obtenu aux questions précédentes : on souhaite maintenant stocker le nombre de fois où on a obtenu la position finale $x_n$ dans un histogramme comme sur la figure 0.2. Pour cela, il faut utiliser un tableau `double proba[nval]` qui sera initialisé à la valeur 0.

A chaque valeur de $x_n$ obtenue par la fonction `marche`, il faut trouver quel est l'élément du tableau qui lui correspond et y ajouter 1.. Par exemple l'élément `proba[0]` du tableau correspond à la position $x_n = -npas$, l'élément `proba[1]` à $x_n = -npas + 1$ etc. jusqu'à `proba[nval-1]` qui réprésente la positon $x_n = npas$.

Apres avoir répété `NMARCHES` fois cette procedure, il faut diviser chaque élement du tableau par `NMARCHES` pour obtenir la fréquence d'apparition de chaque valeur de $x$.

**10.** Ecrivez le programme qui calcule la probabilité $p(n, x)$ sous forme d'histogramme en faisant `NMARCHES`= 10000 marches aléatoires. Pour visualiser le résultat, vous écrirez le contenu du tableau `proba` dans un fichier sur 2 colonnes : $x$ et $p(x)$ pour pouvoir le tracer avec gnuplot (utiliser l'option `with histep` pour les histogrammes).

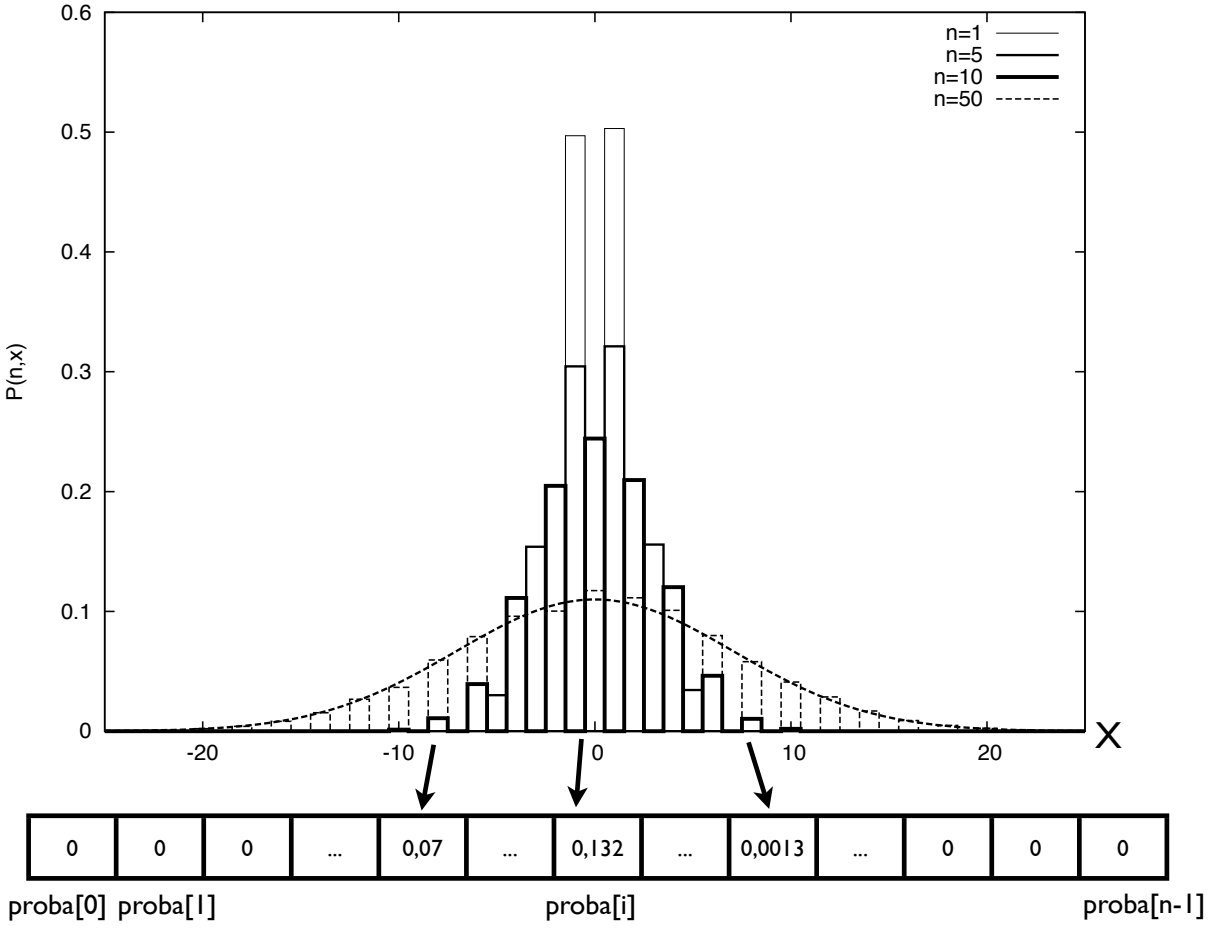



FIGURE 0.2 – Probabilité $p(n, x)$ de se trouver en $x_n = x$ au bout de $n$ pas calculée pour différentes valeurs de $n$.

**11.** Tracer les distributions obtenues pour différentes valeurs de $n$ comme sur la figure 0.2. Vers quelle distribution tend $p(n, x)$ quand $n$ augmente ?

Pour faire le lien avec la physique, la marche aléatoire que nous avons étudié est en lien avec le processus de diffusion. On peut calculer par exemple comment évolue la densité $n(x, t)$ de molécules d'un certain type au cours du temps a partir d'une valeur initiale ou toutes les molécules sont en $x = 0$ (soit $n(x, 0) = \delta(x)$). L'équation de la diffusion, valable à une échelle macroscopique, est donnée par

$$\frac{dn}{dt} = D\frac{d^2n}{dx^2}$$

où la constante $D$ est le coefficient de diffusion. La solution pour une injection ponctuelle en $x = 0$ de toutes les molécules est donnée par :

$$n(x, t) = \frac{1}{\sqrt{4\pi Dt}} e^{-x^2/4Dt},$$

et est équivalente à la distribution limite à grand $n$ que l'on obtient pour la marche aléatoire en faisant la correspondance $D = 1/2$ et $t = n$.

### 0.2.2 Marches aléatoires à pas continus

On considère désormais des marches aléatoires dont le pas n'est plus discrétisé ($\Delta x \pm 1$), mais peut varier de façon continue selon une loi de distribution donnée

$f(\Delta x)$, figure 0.3. On considère d'abord une marche aléatoire avec des pas $\Delta x$ variant continuement entre -1 et +1 et suivant une distribution uniforme dans cet intervalle (en haut à droite de la figure 0.3).

**12.** Ecrivez un programme simulant une telle marche aléatoire. Tracez avec gnuplot une trajectoire d'une marche de 100 pas pour vérifier que tout marche bien.

**13.** Tracer la moyenne et l'écart-type de la position finale en fonction du nombre de pas $n$. Comparer avec les résultats obtenus pour une marche avec pas discret.

On considère à présent que $\Delta x$ suit une loi de distribution gaussienne (en bas à gauche de la figure 0.3) comme c'est souvent le cas dans un grand nombre de phénomènes aléatoires :

$$f(\Delta x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{\Delta x^2}{2}} \tag{1}$$

(la distribution de$\Delta x$ est ici de moyenne nulle et de variance 1.) La génération de ces pas peut être obtenu par l'algorithme de Box-Muller :

```
for ( i = 1 ; i <= NSTEPS/2 ; i++ ){
    proba = alea() ;
    v = sqrt( -2 * log(proba) ) ;
    phi = 2 * M_PI * alea() ;
    step = v * cos(phi) ;
    x = x + step ;
    step = v * sin(phi) ;
    x = x + step ;
}
```

Notez bien que par cet algorithme, deux pas sont générés par le même tirage, ce qui divise par deux le nombre de boucles (`NSTEPS` correspond au nombre de pas $n$ d'une marche donnée).

**14.** Etudiez comme précédemment cette marche gaussienne en traçant la moyenne et l'écart type de la position finale en fonction de $n$ et en comparant aux résultats obtenus précédemment.

Une loi de distribution qu'on rencontre également souvent en physique, est la distribution de Lorentz (figure 0.3 en bas à droite) :

$$f(\Delta x) = \frac{1}{\pi} \frac{b}{(\Delta x - a)^2 + b^2} \tag{2}$$

où $a$ est le centre de la distribution, et $b$ sa largeur à mi-hauteur. On considerera ici le cas $b = 1$ et $a = 0$. Pour générer des pas $\Delta x$ distribués selon une lorentzienne, on peut montrer qu'il faut utiliser la relation $\Delta x = \tan(\pi z)$ (ne pas insérer cela dans la fonction de la distribution de Lorentz, équation 2) ou $z$ est un nombre aléatoire uniformément distribué dans l'intervalle $[0; 1[$ (donné par la fonction `alea()`). Le pas $\Delta x$ obtenu ainsi est distribué selon la distribution de Lorentz.

**15.** Tracez avec gnuplot une trajectoire d'une marche lorentzienne de 1000 pas, et comparez-la aux marches obtenues par les autres lois de distribution. Que remarquez-vous ? Calculer l'écart-type de la position finale pour un nombre de pas donné. Que remarquez-vous ? Pourquoi ?

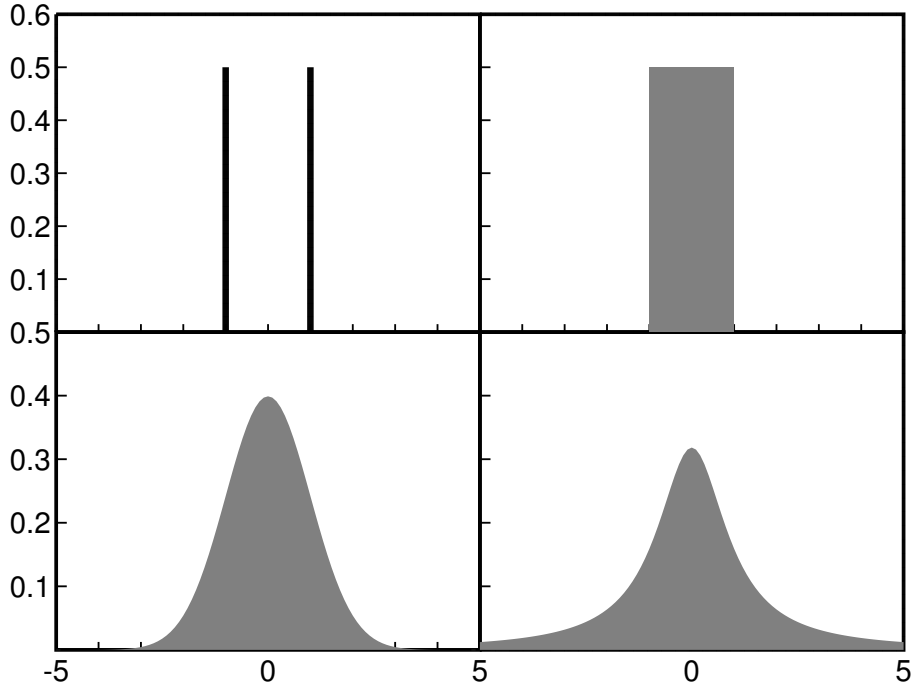



FIGURE 0.3 – Distributions pour le pas, $f(\Delta x)$. *En haut à gauche* : pas discret de ±1, *en haut à droite* : pas continue avec probabilité uniforme entre -1 et +1, *en bas à gauche* : distribution de Gauss, *en bas à droite* : distribution de Lorentz

## 0.3 Pour aller plus loin

On souhaite calculer numériquement la distribution $p(n, x)$ dans le cas d'une distribution continue de $\Delta x$. Par rapport à la méthode décrite aux questions 8-10, il faut modifier la façon d'utiliser le tableau `proba[]` : chaque case du tableau représente maintenant la probabilité que $x_n \in [x; x + \delta]$ où $\delta = (x_{\max} - x_{\min})/\mathrm{n}_{\mathrm{val}}$. Pour déterminer la case du tableau correspondant à une valeur $x$, vous pourrez utiliser la relation :

```
int c=(int) ((x-xmin)/delta);
```

où `delta` désigne la largeur d'un intervalle $\delta$ défini précédemment.

**16.** Implémentez cette nouvelle utilisation du tableau et calculez la distribution $p(n, x)$ obtenue pour la marche continue et uniforme sur $[-1; 1]$. Tracez les histogrammes pour différentes valeurs de $n$ et commentez. Serait-il possible possible d'obtenir un tel histogramme dans le cas d'une distribution lorentzienne de $\Delta x$ ?

**17.** On revient aux pas discrets mais pour simuler une marche en deux dimensions : la particule est sur un réseau carré, sa position est donnée par deux coordonnées entières $(x, y)$ et la particule a quatre directions possibles (haut, bas, gauche, droite) pour chaque déplacement. Étudiez les distributions au bout de $n$ pas de la position $x, y$ et de la distance $r = \sqrt{x^2 + y^2}$ a l'origine.

## 0.4 Annexes

### 0.4.1 Calcul de l'écart-type

Pour rappel, l'écart-type $\sigma_X$ d'une grandeur $X$ est définie comme :

$$\sigma_X^2 = E\left[(X - E(X))^2\right] = \frac{1}{N}\sum_{i=0}^{N}(x_i - < X >)^2$$

où $E(X)$ est l'espérance de $X$, c'est-à-dire la moyenne d'une grandeur $X$. On pourrait utiliser cette formule pour calculer l'écart-type, mais ceci consommerait beaucoup de mémoire vive, car on devrait garder en mémoire toutes les valeurs $x_i$ de $X$ pour calculer la différence entre ces valeurs et $< X >$. Comme déjà pour le calcul d'une moyenne, on peut aussi éviter de garder tous les $x_i$ en mémoire pour le calcul de l'écart-type. Pour cela on transforme la formule de l'écart-type comme ceci :

$$\begin{aligned}\sigma_X^2 &= E\left[X^2 - 2 \cdot X \cdot E(X) + (E(X))^2\right] \\ &= E(X^2) - 2 \cdot E(X) \cdot E(X) + (E(X))^2 \\ &= E(X^2) - (E(X))^2\end{aligned}$$

En plus de la moyenne de $X$, il suffit alors de calculer en même temps la moyenne de $X^2$, ce qui donne alors :

$$\sigma_X^2 = \frac{1}{N}\sum_{i=0}^{N} x_i^2 - \left(\frac{1}{N}\sum_{i=0}^{N} x_i\right)^2$$

# TP 1

# Résolution numérique d'équations différentielles ordinaires - 1ère partie : la méthode d'Euler et rk4

## 1.1 La méthode d'Euler

Une équation différentielle du premier ordre satisfaite par une fonction $y(t)$ est de la forme,

$$\frac{dy(t)}{dt} = f\Big(y(t),\, t\Big) \tag{1.1}$$

où $f$ est une fonction connue qui dépend du problème considéré. La méthode la plus simple pour résoudre numériquement une telle équation est la méthode d'Euler, figure 1.1 . La fonction $y$ est calculée aux points $t_k = k\Delta t$ avec $k$ entier, séparés par un petit intervalle $\Delta t$. La valeur de $y$ au point $t_{k+1} = t_k + \Delta t$ est obtenue à partir de la valeur au point $t_k$ par la formule,

$$y(t_{k+1}) = y(t_k) + f\left(y(t_k),\, t_k\right)) \times \Delta t \tag{1.2}$$

A partir de la donnée d'une condition initiale $y(t_0)$, on peut ainsi calculer de proche en proche $y$ pour des valeurs successives de $t$. Les questions 1-3 proposent des applications de cette méthodes à des cas très simples.

**1.** L'évolution dans le temps de la densité $x(t)$ d'un élément radioactif $X$ subissant une réaction de désintégration $X \to Y$ obéit à l'équation différentielle

$$\frac{dx}{dt} = -kx \tag{1.3}$$

où $k$ est le taux de désintégration. Écrire un petit programme qui calcule par la méthode d'Euler et enregistre $x$ pour des valeurs de $t$ entre 0 et $Tmax$. On prendra $k = 1$ et vous fixerez $Tmax$, $\Delta t$ et $x(0)$. Tracer $x(t)$ et comparer avec la solution analytique de l'équation 1.3.

**2.** Supposons maintenant que l'élément $Y$, dont la densité est notée $y(t)$, se désintègre en un élément $Z$ avec le tau $k_2$. On a alors,

$$\begin{aligned} \frac{dx}{dt} &= -kx \\ \frac{dy}{dt} &= kx - k_2 y \end{aligned} \tag{1.4}$$

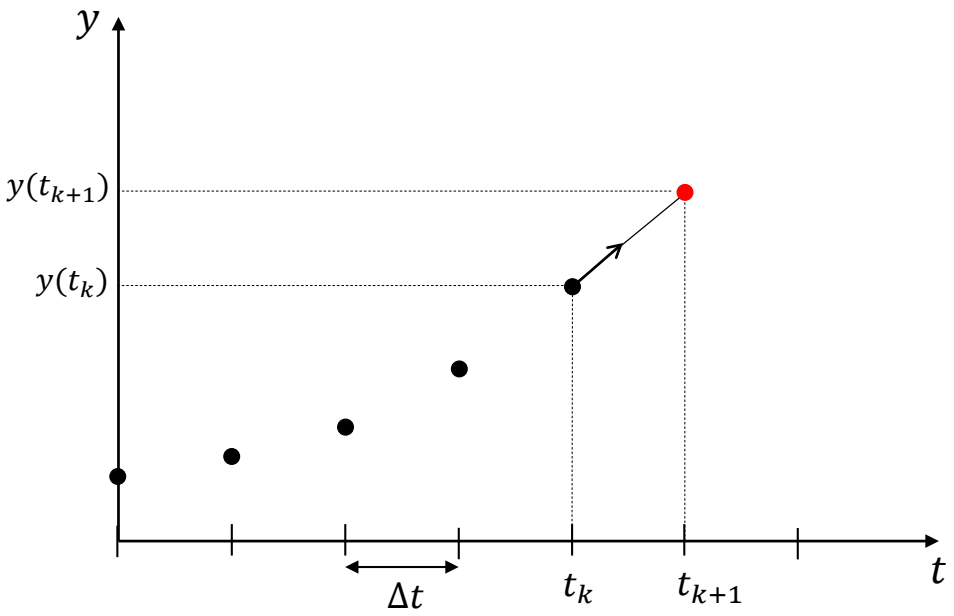



FIGURE 1.1 – Schéma méthode d'Euler.

Écrire un programme analogue au précédent qui calcule et enregistre $x(t)$ et $y(t)$ de $t = 0$ à $Tmax$. On prendra $k_2 = 0.1k$, $x(0) = 1$ et $y(0) = 0$. Tracer $x(t)$ et $y(t)$.

**3.** L'évolution d'un oscillateur harmonique est décrite par une équation du second ordre

$$\frac{d^2x}{dt^2} = -\omega_0^2 x \tag{1.5}$$

Une équation différentielle du second ordre peut être écrite sous forme de deux équations du premier ordre. Écrire ces deux équations et écrire un programme calculant $x(t)$. Vous fixerez les paramètres et les conditions initiales. Essayer plusieurs pas de temps $\Delta t$, qu'est-ce que vous observez sur un grand nombre de périodes ?

De manière générale, une équation différentielle d'ordre supérieur à un peut être transformée en un système d'équations du premier ordre.

Le but des quatre questions suivantes est de construire une fonction `euler` qui implémente l'algorithme d'Euler sur un pas et qui puisse être utilisée sans modification dans tout programme nécessitant de résoudre un système d'équations différentielles. Pour un ensemble de $n$ équations différentielles du premier ordre satisfaites par $n$ fonction $y_i(t)$ avec $i = 0, 1, .., n-1$,

$$\frac{dy_i(t)}{dt} = f_i\left(\{y_j(t)\}, t\right), \tag{1.6}$$

l'algorithme d'Euler pour le calcul des $y_i$ au point $t + \Delta t$ à partir des valeurs en $t$ peut être découpé en deux étapes :

1. le calcul de la dérivée de chaque fonction $y_i$ au point $t$, donnée par $y_i'(t) = f_i\left(\{y_j(t)\}, t\right)$.
2. le calcul des $y_i$ au point $t + dt$ avec la formule $y_i(t + \Delta t) = y_i(t) + y_i'(t) \times \Delta t$ .

Les valeurs des fonctions $y_i$ au point $t$ sont stockées dans un tableau y[n]. L'étape 1 qui dépend de l'équation considérée (de la forme des $f_i$), est réalisée par une fonction,

```
deriv(int n, double t, double y[], double dy[])
```

qui à partir des $y_i(t)$ calcule les $y_i'(t)$ et les stocke dans le tableau dy[]. Notons que `deriv` prend aussi comme argument $n$ et $t$ qui peuvent être nécessaire dans certain cas pour calculer les dérivées.

**4.** Écrire la fonction `deriv` dans le cas de l'équation 1.4. (les paramètres physiques de l'équation différentielle sont déclarés en variables globales).

**5.** Écrire la fonction

```
euler(int n, double t, double y[], double dt)
```

qui reçoit en entrée les $y_i(t)$ dans le tableau y[], ainsi que les valeurs de $n$ , $t$ et $\Delta t$, calcule les $y_i(t + \Delta t)$ et les stocke dans le tableau y[] à la place des $y_i(t)$. Il faudra dans cette fonction déclarer un tableau dy[] et appeler la fonction `deriv` pour remplir ce tableau avec les $y_i'(t)$. Inclure les fonctions `deriv` et `euler` dans un programme complet permettant de traiter la question 2.

**6.** Créer une fonction `deriv3` permettant de résoudre l'équation 5 et modifier `euler` pour traiter la question 3.

**7.** Pour que la fonction `euler` puisse être utilisée sans aucune modification en toute circonstance, il faudrait pouvoir lui spécifier lorsqu'on l'appelle quelle fonction choisir pour calculer les $y_i'$ (par exemple `deriv` ou `deriv3` ou autre). Autrement dit, il faut pouvoir passer en argument la fonction à utiliser pour calculer les $y_i'$ . La fonction `euler` doit alors être de la forme,

```
euler(int n, double t, double y[], double dt, void f(int , double , double[], double[])
```

Écrire la fonction euler. Modifier la fonction `main` en conséquence et tester ce programme sur la question 2 et la question 3.

**8.** Écrire un programme utilisant la fonction euler pour calculer numériquement la trajectoire dans le plan $x - y$ d'une particule de masse $m$ et charge $q$ dans des champs électrique et magnétique uniformes, $\overrightarrow{E} = E\overrightarrow{u_x}$ et $\overrightarrow{B} = B\overrightarrow{u}_z$. On rappelle que l'équation du mouvement est

$$m\frac{d\overrightarrow{v}}{dt} = q\left(\overrightarrow{E} + \overrightarrow{v} \times \overrightarrow{B}\right) \tag{1.7}$$

Les unités sont choisies de telle sorte que $q/m = E = B = 1$. Quel est le nombre $n$ d'équations différentielles ? Écrire ces équations et la fonction `deriv` correspondante.

## 1.2 Méthode de Runge-Kutta

Pour $y(t + \Delta t)$ à partir de $y(t)$, les méthodes de types Runge-Kutta estime la valeur de la dérivée $y'$ au milieu de l'intervalle entre les points $t$ et $t + \Delta t$. La version la plus simple pour résoudre l'équation 1.1 procède ainsi (voir figure 1.2) :

$$\begin{aligned} d_1 &= f(y(t), t) \\ y_p &= y(t) + d_1 \times \Delta t/2 \\ d_2 &= f(y_p, t + \Delta t/2) \\ y(t + \Delta t) &= y + d_2 \times \Delta t \end{aligned} \tag{1.8}$$

Cette méthode s'appelle Runge-Kutta d'ordre 2 car on peut montrer que les erreurs sont d'ordre $(\Delta t)^3$.

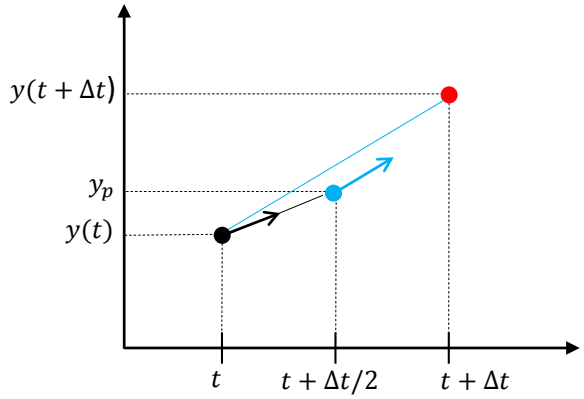



FIGURE 1.2 – Schéma méthode RK2

La méthode la plus couramment utilisée est Runge-Kutta d'ordre 4 (les erreurs sont d'ordres $(\Delta t)^5$ ) qui utilise 4 évaluations de la dérivée. La fonction est donnée dans le fichier rk4.cpp (voir sur moodle) et vous pouvez l'utiliser directement en la copiant dans votre programme.

**9.** Résoudre l'équation $\frac{d^2x}{dt^2} = -\omega_0^2 x$ en utilisant la fonction `euler` avec $\Delta t = 0.01s$. Tracer $x(t) - x_{th}(t)$ où $x(t)$ est la solution numérique et $x_{th}(t)$ est la solution analytique exacte. Que constatez-vous ? Diminuer le $\Delta t$. Le problème est-il résolu ?

**10.** Refaire la question précédente en utilisant à présent la fonction `rk4`. Que constatez-vous ?

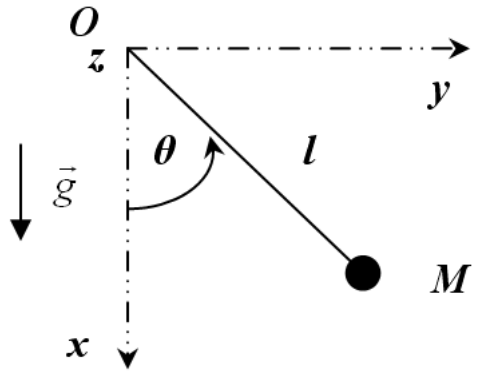
## 1.3 *Pour aller plus loin :* implémenter rk2()

**11.** Ecrivez une fonction rk2() qui implémente l'algorithme RK2 décrit plus haut.

**12.** Comparer cette méthode avec la méthode d'Euler et de rk4 sur l'oscillateur harmonique.

# TP 2

# Résolution numérique d'équations différentielles ordinaires - 2ème partie : application au pendule chaotique




## 2.1 Étude du mouvement d'un pendule avec l'approximation des petits angles

On considère le pendule simple de la figure , dont l'équation du mouvement libre s'écrit :

$$\frac{d^2\theta}{dt^2} + q\frac{d\theta}{dt} + \Omega^2 \sin\theta = 0 \qquad \text{avec } \sin\theta \simeq \theta \quad \rightarrow \quad \frac{d^2\theta}{dt^2} + q\frac{d\theta}{dt} + \Omega^2\theta = 0$$

où $\theta$ est l'angle que fait le pendule par rapport à la verticale, $\Omega = \sqrt{g/l}$ est la pulsation propre et $q$ est le terme de frottement fluide. On utilisera par commodité la valeur suivante : $\Omega = 1\text{rad.s}^{-1}$.

**13.** Résolvez cette équation linéarisée ($\frac{d^2\theta}{dt^2} + q\frac{d\theta}{dt} + \Omega^2\theta = 0$) avec la méthode RK4 pour différentes valeurs de l'amortissement : $q = 1$, $q = 2$, $q = 5\ s^{-1}$ et tracez *sur un même graphe* l'évolution de $\theta(t)$ dans ces régimes respectivement pseudo-périodique, critique et apériodique. On prendra comme conditions initiales $\theta(t = 0) = 10°$ (à convertir en radian) et $\frac{d\theta}{dt}(t = 0) = 0$ et un pas de temps $dt = 0.05$s pour $t$ allant de 0 à 20s. Il est utile ici de mettre la boucle sur le temps qui calcule $\theta(t)$ dans une fonction avec comme paramètre la valeur de l'amortissement $q$. Cette fonction devra remplir deux tableaux : celui qui contient les valeurs du temps $t$ et celui qui contient les valeurs de $\theta(t)$.

On ajoute maintenant une force d'excitation au pendule de sorte que l'équation du mouvement s'écrive :

$$\frac{d^2\theta}{dt^2} + q\frac{d\theta}{dt} + \Omega^2\theta = F_e \sin(\Omega_e t).$$

**14.** Résolvez cette nouvelle équation avec la méthode RK4 pour une force excitatrice d'intensité $F_e = 1\text{rad.s}^{-2}$ et de pulsation $\Omega_e = 2\Omega/3$. Tracez *sur un même graphe* la trajectoire dans l'espace des phase $(\theta, \frac{d\theta}{dt})$ pour le pendule libre ($q = 0$ et $F_e = 0$) , amorti (q=1 et $F_e = 0$), et amorti avec excitation (q=1 et $F_e = 1$). On prendra toujours comme conditions initiales $\theta(t = 0) = 10°$ (à convertir en radian) et $\frac{d\theta}{dt}(t = 0) = 0$. Commentez la forme des trajectoires que vous observez.

## 2.2 Mouvement chaotique

Lorsque l'on ne fait plus l'hypothèse des petits angles ($\sin\theta \simeq \theta$), on obtient une équation différentielle d'ordre 2 qui n'est pas linéaire :

$$\frac{d^2\theta}{dt^2} + q\frac{d\theta}{dt} + \Omega^2 \sin\theta = F_e \sin(\Omega_e t).$$

Pour certaines valeurs des paramètres physiques, le comportement du pendule sera chaotique. Afin d'illustrer ce comportement, on se placera dans les conditions suivantes : $\theta(t = 0) = 10°$ (à convertir en radian) et $\frac{d\theta}{dt}(t = 0) = 0$, $\Omega_e = 2\Omega/3$, $q = 0.5s^{-1}$ .

**15.** Résolvez l'équation du mouvement non-linéaire avec la méthode RK4 pour les valeurs suivantes de l'amplitude d'excitation $F_e = \{1.4, 1.44, 1.465, 1.5\}$ rad.$s^{-2}$ et tracez $\theta(t)$ sur un temps de 100s (choisir le nombre de plots nécessaires pour bien distinguer le comportement). Ajouter deux tests **if** dans la boucle après l'appel à rk4 pour maintenir l'angle $\theta$ dans l'interval $[-\pi; \pi]$. Que constatez-vous au sujet de la période du pendule ? (attention, périodique≠sinusoidal...)

**16.** Dans le cas $F_e = 1.5$ rad.$s^{-2}$, calculez l'évolution de $\theta(t)$ pour deux conditions initiales très proches l'une de l'autre : $\theta(t = 0) = 10°$ et $\theta(t = 0) = 9.999°$. Tracez les deux trajectoires sur un même plot. Au bout de combien temps est-ce qu'un écart entre les deux courbes devient visible ? On voudrait maintenant quantifier comment et avec quelle la vitesse cet écart croit. On essaye de vérifier une croissance exponentielle, c'est-à-dire la valeur absolue de cet écart croit comme $e^{\lambda t}$ où $\lambda$ est « l'exposant de Lyapounov » qui caractérise la vitesse à laquelle deux systèmes quasiment identiques divergent. Essayer de déterminer l'exposant de Lyapounov en traçant le logarithme de cet écart absolu.

Puis en y superposant une droite à la main, vérifiant ainsi qu'il s'agit d'une croissance exponentielle. La pente de cette droite est la valeur de l'exposant de Lyapounov $\lambda$. Qu'en conclure sur vos capacités de prédiction de l'état futur du système ?

## 2.3 *Pour aller plus loin :* diagramme de bifurcation.

Afin de mettre en évidence la transition du système vers le régime chaotique, on peut calculer l'état du système uniquement à des instants $t_n$ tels que $t_n = \frac{2\pi n}{\Omega_e}$ où $n$ est un entier, ce qui veut dire que l'on observe en phase avec la fréquence excitatrice. Comme dans le cas où on observe un mouvement avec un stroboscope, si le pendule est dans un régime stationnaire et oscille à la fréquence d'excitation, alors les valeurs de $\theta_n$ que l'on calcule ont la même valeur, c'est comme si on avait figé le mouvement. En faisant varier

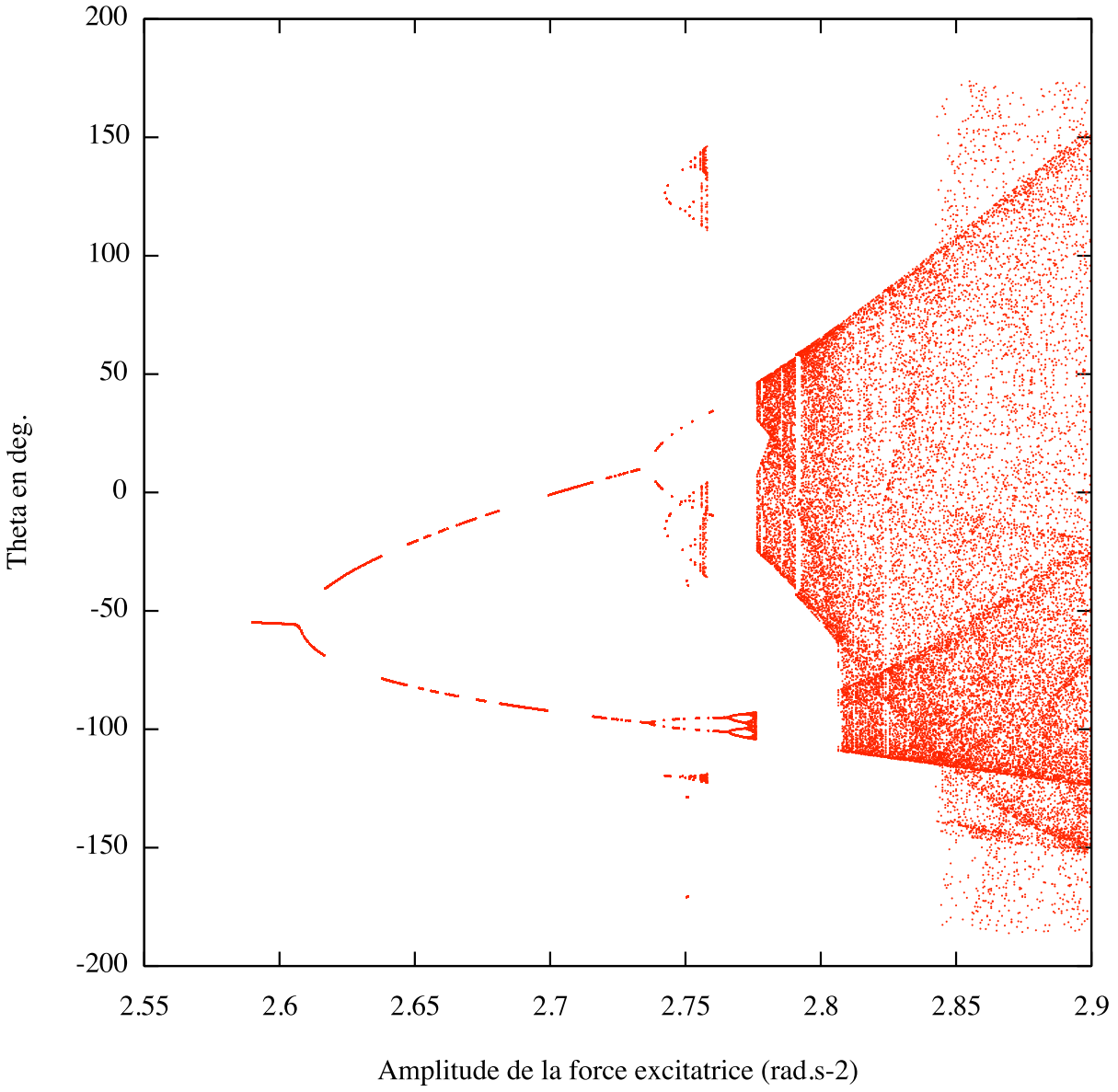



FIGURE 2.1 – Diagramme de bifurcation

la valeur de la force d'excitation, vous pourrez étudier les changements de régime du pendule.

**17.** Pour tracer le diagramme de bifurcation, il faut faire évoluer le système suffisamment longtemps pour éliminer le régime transitoire, et afficher par exemple 50 ou 100 valeurs de $\theta$ pour des temps multiples entiers de la période d'excitation. On choisira un $\delta t$ adapté pour faciliter les calculs, par exemple :$\delta t = \frac{2\pi}{200\Omega_e}$. Pour obtenir de beaux exemples, vous pourrez faire varier $F_e$ entre 1.41 et 1.5 rad.$s^{-2}$ par pas de 0.005, ou entre 2.55 et 2.85. Vous pourrez obtenir un graphe comme montré dans la figure 2.1.

# TP 3

# Adsorption de particules sur une surface

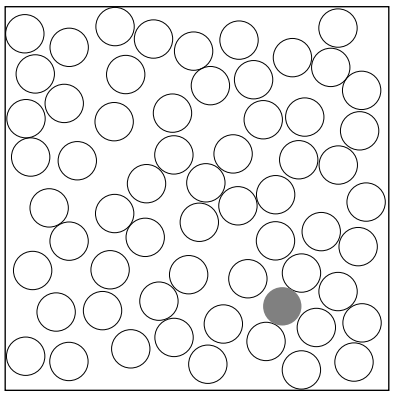
## 3.1 Surface homogène

L'exposition d'une surface cristalline à un gaz donne lieu à des *phénomènes d'adsorption* : les particules du gaz sont piégées sur la surface du cristal. Ce phénomène a de nombreuses applications, en particulier dans la réalisation de catalyseurs.

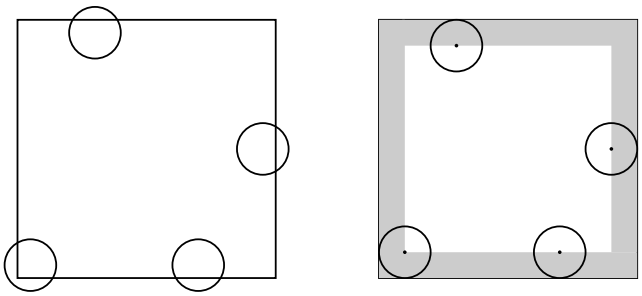
Pour modéliser ce problème, on fait les hypothèses préliminaires suivantes :

- La surface cristalline est un carré de côté $L$.
- Les particules de gaz adsorbées sont modélisées par des disques de rayon $R$ avec $R \ll L$. Ces disques sont impénétrables, c'est-à-dire que deux particules ne peuvent pas se chevaucher. (Voir la figure 3.1a.)
- Une fois qu'une molécule a été adsorbée, elle ne bouge plus et ne quitte plus la surface du cristal.

La simulation fonctionne ainsi : on part d'une surface vide et, à chaque pas de temps, on essaye de rajouter une particule de gaz. La nouvelle particule arrive à un endroit aléatoire si elle ne chevauche aucune particule déjà présente, on la garde, sinon rien



(a) Un exemple de particules sphériques adsorbées aléatoirement sur une surface. La particule indiquée en gris est la dernière que l'on a pu caser.



(b) À gauche : configuration interdite car les particules débordent. À droite : configuration valide. Les particules sont à l'extrême limite de la zone autorisée.

FIGURE 3.1 – Surface homogène

ne se passe et le système n'est pas modifié. On fait ainsi de nombreux essais jusqu'à ce que l'on n'arrive plus à caser de nouvelles particules et on s'intéresse aux propriétés de l'état final, en particulier au nombre de particules que l'on a réussi à placer.

Une particule de gaz adsorbée doit-être entièrement contenue dans le carré (voir figure 3.1b). Si on note $(x, y)$ les coordonnées du centre de la particule, les valeurs autorisées pour $x$ et $y$ en fonction de $L$ et de $R$ sont comprises dans l'intervalle $[R, L-R]$.

Deux particules de coordonnées $(x, y)$ et $(x', y')$ se chevauchent si leur distance est inférieure à $2R$, c'est à dire :

$$\sqrt{(x-x')^2 + (y-y')^2} < 2R$$

On peut donner une *borne supérieure* au nombre de particules de rayon $R$ que l'on peut espérer caser sans chevauchement dans un carré de côté $L$, en prenant le rapport entre la surface du carré et la surface d'un cercle :

$$\text{N\_MAX} = \frac{L^2}{\pi R^2}$$

Sans chevauchement cette borne ne sera évidemment jamais atteint, car il restera toujours des surfaces non-couvertes par des cercles. `N_MAX` sera utile pour réserver à l'avance un nombre suffisant de cases dans un tableau qui contiendra les coordonnées des cercles.

**1.** Écrire un programme capable de simuler ce phénomène d'adsorption. Le programme doit essayer de placer successivement des particules dans le système jusqu'à ce qu'il y ait eu `MAX_TRIES` échecs consécutifs, c'est-à-dire qu'après `MAX_TRIES` d'essais pour placer la dernière particule, il n'a trouvé aucun emplacement libre.

A la fin le programme devra afficher le nombre de particules placées ainsi que le rapport entre la surface totale occupée par toutes les particules adsorbées et la surface du carré. Les constantes $L$ et $R$ et `MAX_TRIES` seront des constantes et l'on pourra prendre, par exemple

```
const double L = 20.0;
const double R = 0.4;
const int  MAX_TRIES = 1000;
```

## 3.1.1 Conseils pour la mise en place du programme

Pour réaliser votre programme de la question précédente, voici un certain nombre de conseils sur les ingrédients à mettre en place. Ces conseils sont là pour vous aider à une mise-en-place rapide et robuste par rapport aux "bugs", mais vous êtes libre de les suivre ou pas et de faire vos propres choix des éléments de langage du C/C++ qui s'offrent ici.

### 3.1.1.1 La structure de données pour représenter les cercles

Comme tous les cercles ont le même rayon $R$, la seule chose qu'il faut enregistrer dans la mémoire vive sont les coordonnées $(x, y)$ des centres des cercles. Pour cela soit on utilise deux tableaux, un pour $x$ et un pour $y$, soit on assemble ces deux valeurs réelles dans un "struct" du C ou encore une "classe" du C++, puis on fait un seul tableau de ce nouveau type "coordonnes". Pour en savoir plus sur les "struct" et les "class", voir le poly de cours C++ 3P002, chapitre 14.2 "struct du C" et chapitre 14.5 "Exemple : TP5 avec C++".

Lors de l'adsorption les cercles seront ajoutés successivement à la surface et devront alors également être enregistrés dans la mémoire vive successivement. Comme le processus est aléatoire, il est impossible de savoir l'avance combien de cercles seront enregistrés à la fin d'un remplissage du carré. On sait par contre que ce nombre n'atteindra jamais `N_MAX` (voir plus haut). La structure de données adaptée ici est un "tableau dynamique", c'est à dire un tableau qui peut changer de taille, comme ici croître d'une case avec chaque ajout de cercle.

On vous propose ici deux manières alternatives pour enregistrer tous les cercles de la surface :

1. Avec la classe vector de la STL (standard template library)
2. Avec les tableaux classiques du C

La première variante avec "vector" a l'avantage qu'elle implémente déjà le tableau dynamique, vous n'avez pas besoin de l'implémenter par vous même. Puis cela peut éviter des bugs de débordement ou mauvaise indexation d'un tableau classique en C.

La deuxième variante a l'avantage d'utiliser ce que vous connaissez déjà : les tableaux classiques en C. Comme ces tableaux ne sont pas dynamiques - on ne peut pas changer leur taille lors de l'exécution - il faudra par contre implémenter par vous même cette fonctionnalité de tableau dynamique sans faire d'erreur au niveau des indices des tableaux.

Les deux variantes sont aussi rapides au niveau de l'exécution du programme, si on respecte les consignes données ci-dessous.

Voir plus bas pour savoir comment faire, puis à vous de choisir ce que vous préférez.

**L'enregistrement des cercles avec la classe vector de la STL** La classe "vector" de la STL (standard template library) est introduite dans les slides `coursTD4_2020.pdf` sur moodle (dossier Ressources). Ici est rappelé tout dont on a besoin pour ce TP, si on choisit d'utiliser "vector" pour enregistrer les cercles.

Pour commencer il faut importer la classe vector : **#include** <vector>. Comme la STL est fournit avec chaque compilateur C++, il n'y a pas besoin de l'installer pour utiliser la classe vector.

Ensuite, dans la fonction gérant l'évolution de notre système, on crée un tableau vide (ici pour l'exemple de la coordonné $x$) : vector<**double**> x; Comme la classe vector permet de stocker des variables et des objets de n'importe quel type, il faut indiquer ici le type des éléments du vector via <**double**>. Si vous utilisez par exemple la classe "Coordinate" du chapitre 14.5 du poly de cours C++ 3P002, alors il aurait fallu écrire : vector<Coordinate> c.

A chaque fois qu'un cercle est adsorbé sur la surface, on ajoute ces coordonnées `x_new` et `y_new` à la fin du vector avec la fonction push_back() : x.push_back(x_new). Cette fonction agrandit le tableau vector d'une case et copie&colle la valeur fournit (ici `x_new`) dans cette nouvelle case.

Le nombre de cercles enregistrés est donné par la fonction size() de la classe vector : x.size(). Pour parcourir les éléments d'un vector, c'est comme pour un tableau classique en C, sauf que la taille donnée par x.size() est toujours correcte :

```
for (int i=0; i < x.size(); i++){
    cout << x[i] << endl;
}
```

Alternativement on peut faire aussi ceci pour lire tous les éléments d'un vector :

```
for (auto elem: x){
```

```
    cout << elem << endl;
}
```

Pour ceux qui ont déjà fait du python, cette notation du C++ moderne (C++11) est l'équivalent du "for elem in x :".

Si vous souhaitez passer le vector à une autre fonction `func`, alors le mieux serait d'utiliser une "référence", indiqué avec le symbole & dans l'entête de la fonction : **void** func(vector<**double**>& v). Cela évite que toutes les cases du vector soient copiées&collées à chaque appel de la fonction. Voir la partie 12.3.2 "Passer une matrice à une autre fonction" du poly de cours C++ 3P002.

Pour optimiser le temps d'exécution, on peut réserver à l'avance suffisamment de cases dans la mémoire vive avec la fonction reserve(), ici alors x.reserve(N_MAX). Cela ne change pas la taille du vector donnée par size(), donc le vector reste un tableau dynamique. Puis lors de la compilation l'ajout de l'option `-O2` (grand O, comme Optimisation 2ème niveau) permet ici de gagner un facteur 10 à 20 environ en vitesse d'exécution.

**L'enregistrement des cercles avec les tableaux classiques du C** Ici le tableau dynamique est implémenté avec un tableau statique du C de taille `N_MAX` et grâce à l'utilisation d'une variable `n_at` qui représente le nombre de cercles déjà présents dans le système (l'équivalent du `x.size()` de la classe vector).

On définira, dans la fonction gérant l'évolution de notre système, trois variables pour indiquer le nombre et la position des cercles adsorbées :

```
int n_at = 0;          // nombre de cercles déjà présents dans le système
double x[N_MAX]; // x[i] est l'abscisse du i-ème cercle présent
double y[N_MAX]; // y[i] est l'ordonnée du i-ème cercle présent
```

Ces variables seront ensuite transmises par argument (pointeurs) aux différentes fonctions qui les utilisent et/ou les mettent à jour. Il faut noter que `x[i]` et `y[i]` ne sont définis que pour $0 \leq i < n\_at$, donc pour parcourir les coordonnées des cercles déjà adsorbés, on fait :

```
for (int i=0; i < n_at; i++){
    cout << x[i] << "  " << y[i] << endl;
}
```

Au départ on a `n_at␣=␣0`, c'est cette variable qui détermine l'emplacement des deux tableaux où on doit ajouter un nouveau cercle : x[n_at] = x_new. Après chaque ajout `n_at` est incrémenté de un. Par construction `n_at` doit toujours rester bien inférieur à `N_MAX`, sinon il s'agit d'un bug. Pour vider la surface il suffit de faire `n_at=0`, pas besoin d'initialiser les valeurs des tableaux `x` et `y` ici.

#### 3.1.1.2 L'organisation du code

Une bonne manière d'écrire le programme est d'utiliser les fonctions suivantes :

- Une fonction **double** coord(**void**) qui renvoie une valeur aléatoire dans l'intervalle autorisé pour la coordonnée $x$ ou $y$ de la nouvelle particule qu'on essaye de placer. On n'a besoin d'écrire qu'une seule fonction **double** coord(**void**) pour les deux coordonnées $x$ et $y$, comme il s'agit d'une surface carrée. Pour obtenir les deux nouvelles coordonnées `x_new` et `y_new`, on appellera coord() deux fois.
- Une fonction **int** remplissage(**int** MAX_TRIES) qui gère l'évolution du système pour arrive à un remplissage de la surface, jusqu'à on a `MAX_TRIES` échecs consécutifs. Elle renvoie le nombre de cercles adsorbés.

La fonction remplissage doit effectuer plusieurs tâches :

```
int remplissage(int MAX_TRIES){
    ... // déclaration tableaux C ou vector pour enregistrer les cercles
    int echecs = 0;
    while(echecs < MAX_TRIES){
        double x_new = coord(); double y_new = coord(); // tirage aléatoire de coordonnées
        // Test si l'emplacement est libre :
        int libre = 1; // supposer au départ que oui
        for (...) // parcourir tous les cercles déjà adsorbés
            if (...) // pour voir s'il y a un qui chevauche
                libre = 0; // si oui, alors la place n'est pas libre
        // Ajout de cercle si emplacement libre, sinon incrémenter echecs
        if ( libre == 1)
            ... // ajout de cercle dans les tableaux C ou vector
            echecs = 0; // remettre le compteur à zéro pour le prochain cercle
        else
            echecs++;
    }
    return ... // retourner le nombre de cercles adsorbés
}
```

### 3.1.2 Analyse des résultats

**2.** Utilisez gnuplot pour visualiser la configuration finale. Pour cela modifier votre programme afin qu'il enregistre un fichier avec trois colonnes "x y R" et une ligne par cercle :

1. x : les positions en x des cercles
2. y : les positions en y des cercles
3. R : rayons des cercles (ici toujours la même valeur pour tous les cercles)

Sous gnuplot il faudra tracer ces données de cette façon :

```
set size square
plot 'fichier.res' with circles
```

Vérifier bien qu'il n'y a pas de chevauchement entre les cercles et qu'aucun cercle dépasse les bords du carré. Une fois que vous avez ainsi graphiquement vérifié que votre programme fonctionne bien, enlever pour la suite les lignes de code qui servent à enregistrer un fichier pour gnuplot.

**3.** Modifier la fonction main() pour que l'expérience soit faite `M` = 1000 fois et que la fonction main() affiche la moyenne sur toutes ces expériences du nombre de particules adsorbées et de fraction de la surface du carré qu'elles occupent. Ici bien sûr on ne visualise pas les configurations obtenues.

Pour vérifier si votre programme fournit un bon résultat, voici les valeurs qu'on devrait obtenir approximativement :

- La moyenne du nombre de particules adsorbées : < n_at >= 373
- La moyenne de la fraction de surface : $\eta =< \text{n\_at} > \cdot \pi R^2/L^2 = 0,47$

**4.** Comment se comparent les valeurs obtenues à la fraction qu'on pourrait idéalement occuper d'une façon ordonnée ?

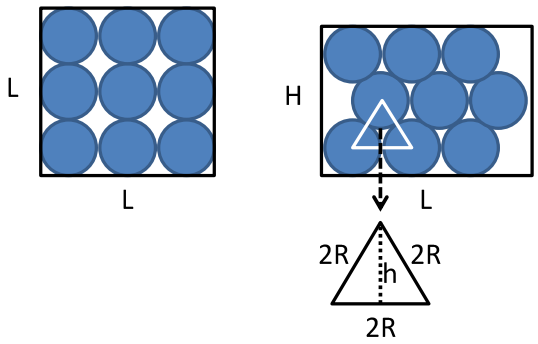



FIGURE 3.2 – Empilement ordonné, *à gauche :* empilement carré, *à droite :* empilement hexagonal

Voici deux exemples d'empilement ordonné : l'empilement carré et hexagonal (voir figure 3.2). Pour l'empilement carré d'une surface avec $L = n \cdot 2R$ la fraction de surface occupée est simplement :

$$\eta = \frac{\pi R^2}{4R^2} = \frac{\pi}{4} = 0,785$$

L'empilement hexagonal est l'empilement le plus compact possible pour des cercles de même taille, ce qui a été démontré par Joseph Louis Lagrange en 1773. La fraction de surface occupée se calcule par exemple comme ceci :

On suppose une surface rectangulaire $L\,x\,H$, avec $L = n \cdot 2R$ et $m$ étant le nombre de lignes de cercles. Le nombre de cercles pour une surface infinie ($L \to \infty$) est alors égale à $m \cdot n$. La distance entre deux lignes est égale à $h = \sqrt{3}R = 1,73R$, comme les centres de trois cercles sont reliés par un triangle équilatéral tel qu'on voit dans la figure 3.2, ce qui donne la relation $R^2 + h^2 = 4R^2$. Donc la hauteur $H$ de la surface peut être exprimé comme ceci, si on suppose un très grand nombre de lignes $m$ :

$$H = 2R + (m-1)h \approx m \cdot h = m\sqrt{3}R$$

Au final on obtient pour l'empilement hexagonal d'une surface $L\,x\,H$ très grande par rapport à $R$ :

$$\eta = \frac{m \cdot n \cdot \pi R^2}{L \cdot H} = \frac{m \cdot n \cdot \pi R^2}{n \cdot 2R \cdot m\sqrt{3}R} = \frac{\pi}{2\sqrt{3}} = \frac{1}{6}\pi\sqrt{3} = 0,907$$

Voir en annexe 3.4.1 de ce TP pour de la lecture intéressante sur l'empilement de bobines ou encore des bonbons M&Ms !

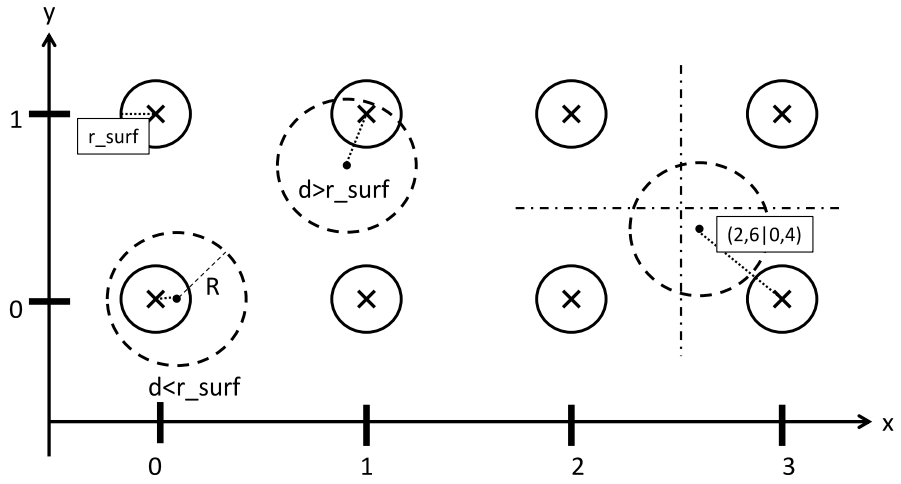



FIGURE 3.4 – Modèle d'adsorption sur un réseau d'atomes, ici en cercles pleins avec croix. Les particules de gaz, ici en cercles avec tirets, peuvent soit s'adsorber dans tous les cas, si $d <$ r_surf ou alors seulement avec une certaine probabilité, si $d >$ r_surf. Comme les atomes ont des coordonnées entières (0,1,2,...), il est très facile de trouver l'atome le plus proche à partir des coordonnées de la particule de gaz : Il suffit de les arrondir vers l'entier le plus proche, ici par exemple : (2,6|0,4) => (3|0).

## 3.2 Surface structurée

On a jusque-là modélisé ce problème en considérant la surface d'adsorption comme parfaitement homogène. En fait, il y a des interactions entre les atomes de la surface d'adsorption et les particules du gaz, et il se trouve que les particules du gaz s'adsorbent plus facilement à proximité des atomes de la surface d'adsorption.

Pour prendre le cas le plus simple, on suppose que les atomes de la surface d'adoption sont organisés selon un réseau cristallin carré de côté 1 (voir figure 3.3) : il y a un atome de la surface en tous les points de coordonnées $(x, y)$ où $x$ et $y$ sont des entiers compris entre 0 et $L$.

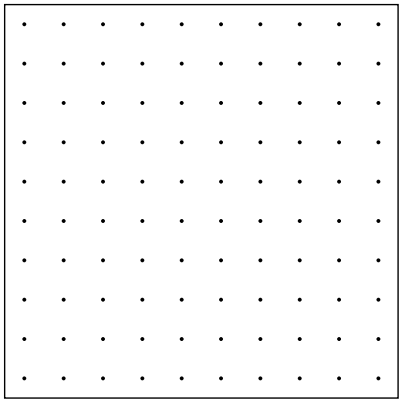


FIGURE 3.3 – Surface d'adsorption avec sa structure atomique, les atomes étant organisés en réseau carré.

Décrire précisément ces interactions est difficile, mais on obtient de bons résultats avec le modèle simplifié suivant (voir figure 3.4) :

- On suppose qu'il existe une distance caractéristique `r_surf` qui décrit la portée de l'interaction entre les atomes de la surface et les particules du gaz.
- Si une particule du gaz essaye de s'adsorber à une distance inférieure à `r_surf` de l'un des atomes de la surface, l'adsorption a toujours lieu.
- Si, au contraire, une particule du gaz essaye de s'adsorber à une distance supérieure à `r_surf` de tous les atomes de la surface, l'adsorption n'a lieu qu'avec une probabilité $\exp(-U/T)$ où $U > 0$ représente le « coût » énergétique à se placer loin

des particules de la surface et où $T$ représente la température. (Pour être parfaitement précis, $T$ est en fait la température en Kelvin multipliée par la constante de Boltzmann $k_B$.)

(Le modèle qu'on vient de décrire est une variante de « l'algorithme de Métropolis ». L'algorithme de Métropolis est très répandu en physique et ailleurs. Voir l'article original de Nicholas Metropolis et al. (1953) sur moodle.)

**5.** Pour quelle valeur de $T$ retrouve-t-on le modèle de la première partie ? Comment pourrait-on décrire en quelques mots le modèle obtenu en prenant $T = 0$ ?

### 3.2.1 Mise en place du programme

**6.** Modifier votre programme pour prendre en compte les atomes de la surface et la température. La distance `r_surf` et l'énergie $U$ seront définies par des constantes et l'on pourra prendre, par exemple,

```
const double r_surf = 0.05;
const double U = 10.0;
```

Il faudra

- Définir une fonction **double** dist_latt(**double** x_new, **double** y_new) qui renvoie la distance du centre de la particule de gaz de coordonnées `(x_new, y_new)` à l'atome le plus proche de la surface d'adsorption. On pourra utiliser la fonction **double** rint(**double**) de la librairie mathématique qui renvoie l'entier le plus proche de son argument (voir figure 3.4 pourquoi cela est utile).
- Modifier la fonction remplissage. Cette fonction prend désormais en argument la température en plus : **int** remplissage (..., **double** T) et doit implémenter l'algorithme décrit.
- Modifier la fonction main() pour simuler le modèle à une température donnée (par exemple, $T = 0$ avec `MAX_TRIES = 10000`).

Quelques aides :

- Pour accepter l'adsorption seulement avec la probabilité $p = \exp(-U/T)$ on pourra utiliser notre fonction `alea()` du TP0 qui fournit un nombre (pseudo-)aléatoire entre zéro et un d'une manière uniforme. La probabilité que la condition `alea() < p` est vraie est alors égale à $p$, si $p \in [0, 1]$. Comme $U > 0$ et $T \geq 0$, on a bien $\exp(-U/T) \in [0, 1]$ et on peut alors utiliser `alea() < exp(-U/T)` comme condition pour accepter l'adsorption pour les cas $d >$ r_surf.
- Pour pouvoir inclure ici le cas $T = 0$, il faut transformer cette condition en : `T*log(alea()) < -U`

### 3.2.2 Dépendance de la température

**7.** Comme avant utiliser gnuplot pour visualiser séparément les configurations finales obtenues pour ces températures $T$ : 0, 1, 2, 5 et 10, (voir au début de la section 3.1.2). Décrivez qualitativement ce que vous observez. Surtout pour des températures proche de zéro il faut utiliser `MAX_TRIES = 10000` pour éviter d'avoir trop de trous à cause d'une recherche pas assez profonde.

**8.** Modifier votre programme pour faire `M=100` simulations indépendantes pour chaque température entre 0 et 10 d'un pas de $\Delta T = 0,5$, et tracer le graphe de la fraction moyenne de la surface occupée en fonction de la température. Utiliser ici un `MAX_TRIES = 10000`.

**9.** En ayant analysé les configurations avec gnuplot (question plus haut), comment peut-on expliquer la courbe de la question précédente ? Peut-on définir une « température critique » ? Avez vous une prédiction théorique pour la valeur de la fraction à basse température ?

## 3.3 *Pour aller plus loin :* Influence de MAX_TRIES sur les résultats

Revenons sur la partie avec une **surface homogène** :

**10.** Répéter pour plusieurs valeurs de `MAX_TRIES` le calcul de la fraction moyenne (taux) de surface occupée par les atomes adsorbés ainsi que l'écart-type. Augmenter `MAX_TRIES` avec un facteur deux entre chaque simulation, comme ceci : 1000, 2000, 4000, ..., 32000. Utiliser `M=100` répétitions seulement pour limiter le temps de calcul. Tracer la fraction moyenne de surface occupée en fonction de MAX_TRIES en échelle semi-log.

Commentez le graphe.

**11.** Essayer d'étendre la courbe jusqu'à `MAX_TRIES` = 512.000 et avec `M=100` répétitions. Pour cela n'oubliez pas de compiler votre programme C++ avec l'option "-O2" (O comme Optimisation). Commentez le nouveau graphe.

## 3.4 Annexe

### 3.4.1 Liens sur l'empilement compact

En sciences on peut aussi faire des expériences amusantes et les publier dans un journal important comme "Science". Comme cette expérience avec laquelle les chercheurs ont démontrés que l'empilement au hasard des bonbons M&Ms dans une sphère donne un empilement presque aussi dense que l'empilement ordonné le plus compact de sphères de taille égale (empilement cubique à faces centrées ou empilement hexagonal compact) :

`http://www.ncbi.nlm.nih.gov/pubmed/14963324`

(voir aussi sur moodle pour avoir l'article)

Cette bonne performance des bonbons M&Ms est dû à leur forme ellipsoïdale, qui permet un meilleur empilement.

# TP 4

# Valeurs propres et vecteurs propres : Résolution de l'équation de Schrödinger par un calcul de différences finies

L'équation de Schrödinger $H\psi = E\psi$ est l'équation maîtresse de la mécanique quantique : résoudre un problème de physique quantique revient en général à résoudre cette équation dans le cas considéré. Malheureusement, même dans les cas simples, on se heurte à des problèmes techniques souvent ardus quand on cherche à le faire ; fort souvent même, il n'y a pas de solution analytique du tout ! Dans le présent exercice, on étudiera une méthode simple, peu exigeante sur le plan de la programmation (il s'agit essentiellement de faire appel à un sous-programme de bibliothèque déjà existant) pour tenter de résoudre numériquement des problèmes dont la solution est connue (puits infini, oscillateur harmonique) afin de contrôler la méthode, puis dans des cas où la solution n'est pas connue. Ce sera aussi l'occasion de se familiariser à peu de frais avec cet objet étrange qu'est la fonction d'onde $\psi$.

On se restreint ici à l'équation de Schrödinger stationnaire (ou non-dépendante du temps) à une particule et à une dimension. Il s'agit donc de chercher les états propres d'une particule dans un potentiel stationnaire [1].

Dans ces conditions, l'équation de Schrödinger s'écrit :

$$-\frac{\hbar^2}{2m}\frac{d^2\psi(x)}{dx^2} + V(x)\,\psi(x) = E\,\psi(x) \tag{4.1}$$

où $V$ et $\psi$ sont des fonctions de l'abscisse $x$, $V(x)$ est l'énergie potentielle de la particule au point d'abscisse $x$, $\psi(x)$ la fonction d'onde qui décrit l'état de la particule et $E$ l'énergie associée à cet état ; $m$ est la masse de la particule et $\hbar$ la constante de Plank. Résoudre l'équation (4.1) revient à trouver les fonctions d'ondes $\psi(x)$ et les énergies $E$ pour lesquelles l'équation est vérifiée : il s'agit d'une équation différentielle du second ordre dont on connaît les solutions analytiques dans certains cas ($V = Cst$, $V = \frac{1}{2}kx^2$, ...), mais reste insoluble dans bien des cas d'où la recherche de solutions numériques.

Afin de simplifier les notations, on utilisera un système d'unités *ad hoc* dans lequel $\frac{\hbar^2}{2m} = 1$, donc l'équation (4.1) devient :

$$-\frac{d^2\psi(x)}{dx^2} + V(x)\,\psi(x) = E\,\psi(x) \tag{4.2}$$

1. voir le cours de physique quantique.

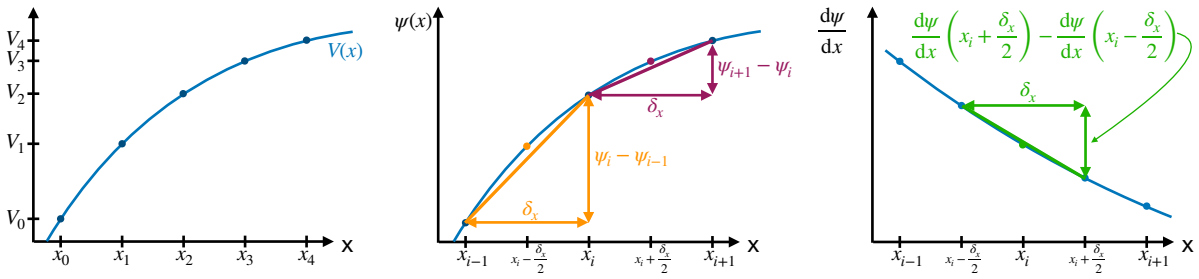



FIGURE 4.1 – Gauche : Illustration de la discrétisation de $x$ ainsi que des conséquences pour le potentiel $V$ et la fonction d'onde $\psi$. Centre : La dérivée première de $\psi$ est approchée à $x_i \pm \delta_x/2$. Droite : La dérivée seconde de $\psi$ est approchée à $x_i$.

## 4.1 La méthode des différences finies.

Le problème est légèrement différent de celui de la résolution d'une (ou plusieurs) équation(s) du mouvement où l'on calcule de proche en proche les valeurs que prend la ou les fonctions recherchées : ici, on veut simultanément *toutes* les valeurs de $\psi(x)$ et de surcroît $E$ est aussi une inconnue. Les méthodes de type Runge-Kutta ne conviennent donc pas, il faut en trouver d'autres.

La méthode des différences finies en est une possible. Elle consiste en une discrétisation du problème, c'est-à-dire que l'on fait l'approximation de remplacer la variable continue $x$ par une variable discrète $x_i$ :

$$x_i = -L/2 + i \cdot \delta_x, \; i \in [0, n-1]$$

où $i$ est un indice entier. La droite infinie est évidemment remplacée par un segment fini de longueur $L = (n-1)\delta_x$, centré autour de zéro, on couvre alors l'intervalle $[-L/2, L/2]$.

Cela entraîne que $\delta$ doit être "petit", c'est-à-dire sensiblement plus petit qu'une distance "typique" sur laquelle la fonction d'onde varie sensiblement. Par ailleurs, $L$ doit être "presque" infini, soit, plus grand que le domaine dans lequel la particule peut se déplacer. Autrement dit, $n$ doit être "grand".

Notre objectif est maintenant de discrétiser l'équation de Schrödinger, c'est-à-dire en particulier la fonction d'onde ainsi que sa dérivée seconde. On pose :

$$\begin{aligned} \psi_i &= \psi(x_i) \\ V_i &= V(x_i) \end{aligned}$$

pour la fonction d'onde et le potentiel discrétisés. Pour la dérivée première de $\psi$, prise un demi-intervalle au delà de $x_i$, on peut trouver une expression approchée :

$$\frac{d\psi}{dx}\left(x_i + \frac{\delta_x}{2}\right) \sim \frac{\psi_{i+1} - \psi_i}{\delta_x}$$

De même on trouve pour la dérivée prise un demi-intervalle en deçà de $x_i$ :

$$\frac{d\psi}{dx}\left(x_i - \frac{\delta_x}{2}\right) \sim \frac{\psi_i - \psi_{i-1}}{\delta_x}$$

En utilisant ces expressions, nous pouvons ensuite en déduire une expression approchée pour la dérivée seconde de $\psi$ prise en $x_i$ :

$$\frac{d^2\psi}{dx^2}(x_i) \sim \frac{\frac{d\psi}{dx}\left(x_i + \frac{\delta_x}{2}\right) - \frac{d\psi}{dx}\left(x_i - \frac{\delta_x}{2}\right)}{\delta_x} = \frac{\psi_{i+1} - 2\psi_i + \psi_{i-1}}{\delta_x^2}$$

Muni de ces expressions approchées, nous pouvons établir une forme discrétisée de l'équation de Schrödinger :

$$-\frac{\psi_{i+1} - 2\psi_i + \psi_{i-1}}{\delta_x^2} + V_i\psi_i = E\psi_i \tag{4.3}$$

Dans la suite on va étudier trois problèmes qui correspondent à des potentiels V(x) différents :

1. le puits carré infini : Il suffit de fixer la valeur du potentiel à zéro partout sur l'intervalle, $V(x) = 0$.
2. le potentiel harmonique : $V(x) = \frac{m}{2}\omega^2 x^2$. Ici on prendra $\omega = \sqrt{\frac{1}{m}}$ (voir ci-dessous).
3. un double puits, paramétré par $V(x) = a(x - r_1)(x - r_2)(x - r_3)(x - r_4)$, soit un polynôme de degré 4 dont les racines sont $r_1, r_2, r_3, r_4$.

**1.** Écrire trois fonctions (une par problème) qui rendent la valeur $V(x)$. Lorsqu'on voudra ensuite changer la forme du potentiel il suffira de modifier l'appel de la fonction, à l'exclusion du reste du programme. Ces fonctions sont alors de cette forme :

```
1 double potentiel_puits(double x){
2     return ...
3 }
```

⇒ Rendre un seul script python qui contient les résultats des éxercices 1 et 2.

**2.** Définir un intervalle $[-L/2, L/2]$, le diviser en $n$ segments de longueur $\delta_x$ et remplir les tableaux $x$ (contenant les $x_i$) et $V$ (contenant les valeurs de potentiel $V_i$). On peut utiliser des tableaux numpy classiques (structure de données `ndarray`). Tracer ensuite les trois potentiels définis dans l'exercise précédent dans une seule figure. Choisir $L = 5, n = 100$ et $a = 1, r_1 = -r_4 = 2$ , $r_2 = -r_3 = 0.5$.
⇒ Rendre une seule figure avec l'inscription des axes et la légende.

## 4.2 Valeurs propres-vecteurs propres.

On considère maintenant que la fonction d'onde discrétisée est un vecteur :

$$\psi = \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_i \\ \vdots \\ \psi_{n-2} \\ \psi_{n-1} \end{pmatrix}$$

On peut écrire l'expression de la matrice $n \times n$ notée $\mathbf{H}$ telle que l'équation de Schrödinger discrétisée s'écrive maintenant sous forme matricielle :

$$\mathbf{H}\psi = E\psi$$

On appellera **H** l'Hamiltonien discrétisé : ce n'est plus un opérateur comme dans l'équation (4.1), mais une matrice [2].

Autrement dit, l'équation de Schrödinger discrétisée s'écrit pour tous les $\psi_i$ :

$$\sum_{j=0}^{n-1} H_{ij} \cdot \psi_j = E \cdot \psi_i \tag{4.4}$$

Si on regroupe l'équation 4.3 par les différents $\psi_i$, on obtient :

$$\begin{aligned} -\frac{\psi_{i+1} - 2\psi_i + \psi_{i-1}}{\delta_x^2} + V_i\psi_i &= E\psi_i \\ -\frac{1}{\delta_x^2}\psi_{i-1} + \left(\frac{2}{\delta_x^2} + V_i\right)\psi_i - \frac{1}{\delta_x^2}\psi_{i+1} &= E\psi_i \end{aligned} \tag{4.5}$$

En comparant l'équation 4.4 avec l'équation 4.5 on obtient :

$$\begin{aligned} H_{ii} &= \frac{2}{\delta_x^2} + V_i \\ H_{i(i-1)} &= -\frac{1}{\delta_x^2} \\ H_{i(i+1)} &= -\frac{1}{\delta_x^2} \end{aligned}$$

Donc concrètement, la matrice **H** a cette forme :

$$\mathbf{H} = \begin{pmatrix} \frac{2}{\delta_x^2} + V_0 & -\frac{1}{\delta_x^2} & 0 & 0 & 0 & \dots & 0 \\ -\frac{1}{\delta_x^2} & \frac{2}{\delta_x^2} + V_1 & -\frac{1}{\delta_x^2} & 0 & 0 & \dots & 0 \\ 0 & -\frac{1}{\delta_x^2} & \frac{2}{\delta_x^2} + V_2 & -\frac{1}{\delta_x^2} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & 0 & -\frac{1}{\delta_x^2} & \frac{2}{\delta_x^2} + V_i & -\frac{1}{\delta_x^2} & 0 & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & 0 & -\frac{1}{\delta_x^2} & \frac{2}{\delta_x^2} + V_{n-2} & -\frac{1}{\delta_x^2} \\ 0 & \dots & 0 & 0 & 0 & -\frac{1}{\delta_x^2} & \frac{2}{\delta_x^2} + V_{n-1} \end{pmatrix}$$

C'est une matrice tridiagonale symétrique.

Les extrémités de l'intervalle, $x_0$ et $x_{n-1}$, doivent faire l'objet d'une attention particulière. On fera l'approximation supplémentaire que $\psi_{-1} = \psi_n = 0$, c'est-à-dire que la fonction d'onde « à l'infini » est nulle : En $i = 0$ on n'a pas le terme en $\psi_{i-1}$ et en $i = n-1$, le terme $\psi_{i+1}$ est absent. Cela revient à postuler que la fonction d'onde est nulle aux deux extrémités de l'intervalle fini considéré. La conséquence en est que la particule ne doit pas s'aventurer vers les extrémités de l'intervalle et que donc l'extension de sa fonction d'onde doit rester à peu près centrée et petite devant la taille de l'intervalle.

2. Pour mieux comprendre le lien entre la matrice **H** et l'équation (4.1), il est utile d'écrire l'opérateur hamiltonien en représentation de position : L'opérateur hamiltonien est défini comme $\hat{H} = \frac{\hat{P}^2}{2m} + V(\hat{X})$ avec l'opérateur impulsion $\hat{P}$ et l'opérateur de position $\hat{X}$ . On fait alors le choix de la "représentation de position $|x\rangle$" définie par $\hat{X}|x\rangle = x|x\rangle$ où x est maintenant une variable. Dans cette représentation on obtient $\langle x|P|\psi\rangle = -i\hbar\frac{d\psi(x)}{dx}$ et $\langle x|V(X)|\psi\rangle = V(x)\psi(x)$ où $\langle x|\psi\rangle = \psi(x)$ est la fonction d'onde.

3. Quel sens physique cela peut-il avoir ? Quelle est la conséquence pour le potentiel effectif qu'on simule avec cette approche ?

Le problème mathématique à résoudre est maintenant devenu une équation aux valeurs propres : les valeurs que peut prendre $E$ sont les valeurs propres de la matrice $\mathbf{H}$ et le vecteur $\psi$ est le vecteur propre asssocié à chaque valeur propre. Comme la matrice est $n \times n$, il doit y avoir $n$ valeurs propres et $n$ vecteurs propres solution de l'équation de Schrödinger discrétisée.

## 4.3 Mise en œuvre numérique.

Il n'est pas question d'écrire une fonction capable de résoudre un problème aux valeurs propres ! Il existe de nombreuses bibliothèques de programmes déjà écrits et testés : ils sont fiables et efficaces, il est inutile de réinventer la roue ! On utilisera ici la bibliothèque EIGEN, qui est spécialement écrite pour le C++ et décrite dans le poly de cours C++ de 3P002. EIGEN permet non seulement de travailler avec des vecteurs et matrices (= tableaux 2D), mais aussi de diagonaliser des matrices pour résoudre un problème aux valeurs et vecteurs propres. Pour faciliter la tâche, voici une fonction qui utilise EIGEN pour diagonaliser une matrice carrée symétrique `z` (téléchargeable sur moodle) :

```
1 void solve(MatrixXd &z, VectorXd &d, MatrixXd &v){
2    SelfAdjointEigenSolver<MatrixXd> eigensolver(z);
3    if (eigensolver.info() != Success){
4         cout << "ERROR in SelfAdjointEigenSolver" << endl;
5         abort();
6    }
7    d = eigensolver.eigenvalues();
8    v = eigensolver.eigenvectors();
9    cout << "La diagonalisation s'est bien passée" << endl;
10 }
```

Cette fonction `solve()` fournit les valeurs propres dans le vecteur (= tableau 1D) `d` et les vecteurs propres dans la matrice (= tableau 2D) `v`. On l'appelle comme ceci :

```
1 int n = 100;
2 MatrixXd z(n,n);
3 ... // remplissage de z
4 VectorXd eigenvalues;
5 MatrixXd eigenvectors;
6 solve(z, eigenvalues, eigenvectors);
```

Ici les objets `z,␣eigenvalues` et `eigenvectors` sont passés à la fonction `solve()` via des *références* comme indiqué par les & dans l'entête de la fonction `solve()`. Ceci permet à `solve()` de lire et écrire sur ces objets. Voir le chapitre sur EIGEN dans le poly de cours C++ de 3P002 pour en savoir plus.

Pour pouvoir utiliser EIGEN, il faut ajouter aussi ceci au début du programme :

```
#include "Eigen/Dense"
using namespace std;
using namespace Eigen;
```

Puis on doit encore copier ou décompresser le dossier “Eigen” dans le dossier de votre programme, voir le poly de cours C++ de 3P002.

Enfin, pour la compilation il est très utile ici d’ajouter l’option “-O2” (grand O pas zéro !) à la ligne de commande du compilateur, pour activer les optimisations automatiques du code du 2ème niveau. Ceci peut accélérer la diagonalisation de matrice d’un facteur 10 à 20 pour atteindre des performances similaires à d’autres bibliothèques très utilisées dans le domaine comme par exemple LAPACK qui est écrite pour Fortran.

Il ne reste plus qu’à remplir la matrice `z` qui représente $\mathbf{H}$ en forme discrète. Sa taille `n*n` dépend du nombre d’éléments du vecteur $\psi$. Les vecteurs propres sont enregistrés colonne par colonne dans la matrice `eigenvectors` avec la fonction `solve()`.

Pour la sortie des vecteurs propres dans un fichier, on peut profiter des fonctionnalités d’EIGEN :

```
ofstream fichier("TP6_q3.res");
MatrixXd results(n,n+1);
results << x, eigenvectors;
fichier << results;
fichier.close();
```

Comme on doit ici écrire les positions `x[i]` en première colonne, on crée d’abord une matrice “results” avec n+1 colonnes, puis on la remplit avec le vecteur de n éléments `x` et la matrice de taille n*n `eigenvectors` d’en haut. Ensuite l’écriture de “results” sur le fichier se fait très facilement, pas besoin d’une seule boucle ! Il faut juste faire attention que `x` n’est pas un tableau C classique, mais bien un objet du type `VectorXd`.

On remarquera que les fonctions d’ondes calculées ne sont pas normalisées comme les fonctions d’ondes théoriques avec

$$\int \psi^2 dx = 1$$

On peut le faire, si souhaité, avec cette fonction par exemple sur la matrice `eigenvectors` :

```
// Using the normalize() function of Eigen and vector-operations
  ↪  on each column using the colwise() function.
// The multiplication with the scalar 1/sqrt(dx) can be done for
  ↪  all elements of the matrix with a single command:
void normalize(MatrixXd &m, double dx)
{
    m.colwise().normalize(); // columns of m are already
  ↪  normalized here, but to be shure we do it again here.
    m = m / sqrt(dx);
}
```

Lors de ce TP seront générés beaucoup de fichiers *.res différents en fonction du potentiel et des paramètres `L` et `n`. Pour garder une trace de quel fichier correspond à quoi, il est utile de directement inclure les valeurs des paramètres dans le nom de fichier. Pour faire cela d'une manière automatique, on peut utiliser les "string-stream" pour définir le nom de fichier, c'est-à-dire les *flux* qui permettent d'assembler une chaîne de caractères. Voir cet exemple :

```
#include <fstream> //file-stream
#include <sstream> //string-stream

// dans main():
ostringstream filename;
filename << "q4_harm_valeurs_n" << n << "_L" << (int)L << ".res";
ofstream fichier(filename.str().c_str());
```

Ici le nom de fichier (`filename`) indique qu'il s'agit de valeurs propres de la question 4 pour un potentiel harmonique avec les paramètres `n` et `L`. Ce qui donne par exemple : `q4_harm_valeurs_n100_L20.res`. On utilise ici la même syntaxe que pour les `cout`, on a seulement remplacé l'objet `cout` par l'objet `filename` qui est du type `ostringstream` ("output string stream" => pour créer une chaîne de caractères). Ensuite pour récupérer de l'objet `filename` une chaîne de caractères classique du langage C, il faut d'abord extraire un objet du type `string` avec `filename.str()`, puis de celui-ci la chaîne de caractères classiques avec `c_str()`, ce qui donne en une seule ligne : `filename.str().c_str()`.

**4.** Écrire une fonction qui rend la valeur $V(x)$. Pour le puits infini, il suffit de mettre 0, pour le potentiel harmonique, $x^2$. Lorsqu'on voudra ensuite changer la forme du potentiel, il suffira de modifier cette fonction, à l'exclusion du reste du programme. Cette fonction est alors de cette forme :

```
double potentiel(double x){
    return ... ;
}
```

**5.** Écrire un programme principal qui :

1. définisse un intervalle $[-L/2, L/2]$, le divise en $n$ segments de longueur $\delta x$ et remplisse les tableaux `x[i]` (contenant les $x_i$) et `V[i]` (contenant les valeurs de potentiel $V_i$). On peut soit utiliser des tableaux C classiques, soit des objets du type `VectorXd` d'EIGEN. Pour faciliter l'écriture sur un fichier, il est conseillé d'utiliser la 2ème variante : `VectorXd␣x(n),␣V(n)`. L'accès aux tableaux C et les vecteurs d'EIGEN se fait de la même façon avec `x[i]` et `V[i]`. Prenez ces valeurs pour commencer : $L = 5$ et $n = 100$.
2. crée le tableau 2D `MatrixXd␣z(n,n)`, l'initialise à zéro, puis le remplisse avec les éléments diagonaux et sous-diagonaux de **H**.
3. en recherche les valeurs propres et les vecteurs propres et écrive les valeurs propres dans un fichier et quelques vecteurs propres dans un autre.

*Rappel :* La compilation de ce programme se fait avec l'option `-O2` pour optimiser le code :
`c++ -Wall -Wextra -O2 mon_prog.cpp -o mon_prog`
où `mon_prog.cpp` est évidemment le nom de *votre* programme, que vous pouvez choisir librement...

## 4.4 Étude numérique.

### 4.4.1 Instructions pour gnuplot

- Utiliser uniquement des scripts gnuplot dans ce TP, voir poly de cours C++ de 3P002. Il y aura un très grand nombre de graphes à créer et des scripts gnuplot permettent de gagner du temps et de garder une trace sur comment vous avez généré vos graphes. **Vos scripts gnuplot doivent être imprimés et joints à votre compte rendu.**
- Faites une sortie sur fichier *.ps, puis imprimer ces fichiers seulement quand vous êtes satisfait du résultat.
- Nota bene : Lors de ce TP seront générés beaucoup de fichiers *.pdf différents en fonction du potentiel. Pour garder une trace de quel fichier correspond à quoi, il est utile de directement inclure les informations saliants dans le nom de fichier, par exemple `q6_puits_valeurs_n100_L5.pdf` correspondrait aux valeurs propres de la question 6 pour un puits de potentiel avec $n = 100,\ L = 5$.
- Pour ouvrir plusieurs fenêtres en même temps avec gnuplot sous Linux, il y a la commande `set term X11 0` pour la première fenêtre, puis `set term X11 1` pour la 2ème etc. Ces commandes sont à mettre à l'endroit de votre scripts où vous voulez ouvrir une nouvelle fenêtre sans oublier d'incrémenter l'identifiant de la fenêtre.
- Pour tracer la courbe théorique de $E_p$ ou les fonctions d'ondes théoriques gnuplot permet d'entrer directement les formules mathématique. Par exemple pour $E_p$ du puits carré infini : `plot pi*pi*x*x/(L*L)` si on définit la variable `L` plus haut dans votre script gnuplot : `L=5`
- Comme pour le langage C, il faut faire attention avec gnuplot aux divisions entre valeurs entières. Par exemple pour tracer une fonction d'onde théorique, il faudra plutôt écrire `sqrt(2.0/L)` que `sqrt(2/L)` si `L` est définit en étant une variable entière avec l'affectation `L=5`. En gnuplot on n'écrit pas explicitement le type d'une variable, il dépend de la valeur affecté.

### 4.4.2 Puits carré infini.

*Rappels théoriques*[3] *:*

Si on considère une particule dans un puits carré infini de largeur $L$, les fonctions d'onde solutions de l'équation de Schrödinger indépendante du temps sont nulles en dehors du puits et ont la forme suivante dans le puits

$$\psi_p^{\text{theo}}(x) \quad = \quad \sqrt{\frac{2}{L}} \sin \frac{(p+1)\pi(x+\frac{L}{2})}{L}$$

3. Reportez-vous au cours de physique quantique.

Les énergies de ces états s'écrivent :

$$E_p^{\text{theo}} = \frac{\hbar^2}{2m}\left(\frac{\pi(p+1)}{L}\right)^2$$

Rappel : on choisit ici $\frac{\hbar^2}{2m} = 1$.

**6.** Essayer votre programme avec $n = 100$ et $L = 5$, afin de comparer le résultat obtenu avec le résultat théorique : Tracer les énergies propres $E_p$ en fonction de $p$ avec, sur le même graphe, la courbe théorique $E_p^{\text{theo}}$. Vérifier l'accord avec la courbe théorique et discuter le résultat.
⇒ Rendre une seule figure avec les deux courbes.

Ensuite, on souhaite comparer également les fonctions d'onde. On remarquera que les fonctions d'ondes calculées ne sont pas normalisées comme les fonctions d'ondes théoriques avec

$$\int \psi^2 dx = 1$$

**7.** Tracer les premières fonctions d'onde ($p \in [0,2]$) obtenues par votre programme : Pour cela tracer sur le même graphe les premières ($p \in [0,2]$) fonctions d'onde *au carré*, décalées verticalement d'une valeur proportionnelle à leur énergie $E_p$. Attention à ne pas confondre les lignes et les colonnes - les vecteurs propres se trouvent dans les **colonnes** de la matrice v. Discuter le résulat qualitativement eu égard des résultats attendus pour un puits carré infini.
⇒ Rendre une seule figure avec les trois courbes décalées ainsi qu'un code C++ qui contient votre programme complet.

*Astuces gnuplot :*

- Pour faire ce graphe complexe, on peut s'aider en utilisant la modification des données à la volée de gnuplot (voir aussi le poly de cours C++ de 3P002). Voici un exemple où il faudra encore insérer vos valeurs de $E_p$ et ajuster un facteur d'échelle. On y voit que pour faire le carré de $\psi_p$ on utilise `($2)**2` (pas possible sous C++ !). Le mieux est de mettre ces instructions gnuplot dans un script gnuplot :

```
E0 = ...  # la valeur de l'energie pour p=0
E1 = ...  # la valeur de l'energie pour p=1
E2 = ...  # la valeur de l'energie pour p=2
scale = ... # un facteur d'echelle entre l'amplitude des
            # fonctions d'ondes et l'energie
plot 'vecteurs.res' u 1:(scale*($2)**2 + E0) title 'Fondamental' w l
replot 'vecteurs.res' u 1:(scale*($3)**2 + E1) title '1er excite' w l
replot 'vecteurs.res' u 1:(scale*($4)**2 + E2) title '2nd excite' w l
```

**8.** Comparer l'accord entre les fonctions d'onde numériques et théoriques pour $p = 1$ et un ordre plus élevé dans le domaine où l'accord pour $E_p$ n'est pas bon, par exemple $p = 55$. Attention à ne pas mélanger les fonctions d'onde avec leurs valeurs au carré. Faites votre diagnostic.
⇒ Rendre deux figures avec l'inscription des axes et la légende - un graphe par valeur de $p$. Faites attention à rendre des figures avec des courbes lisibles. Si nécessaire, ajuster

l'ordonnée ou modifier l'échelle de la courbe.

Sur deux autres graphes comparer l'accord entre les fonctions d'onde numériques et théoriques pour $p = 1$ et un ordre plus élevé dans le domaine où l'accord pour $E_p$ n'est pas bon, par exemple $p = 55$ (un graphe par valeur de $p$). Faites votre diagnostic. L'amplitude et le signe de la fonction d'onde numérique peuvent être différents comparés à la fonction d'onde théorique, si on ne normalise pas la fonction d'onde numérique (voir plus haut).

Refaire cette dernière analyse de comparaison des résultats numériques et théoriques (pour $E_p$ et $\psi_p$) avec différentes valeurs de $n$ et de $L$, par exemple : $n = 200$ & $L = 5$, $n = 100$ & $L = 10$ et $n = 200$ & $L = 10$. Comparer les résultats.

### 4.4.3 Potentiel harmonique.

*Rappels théoriques :*
Le deuxième cas dont la solution est connue est le potentiel harmonique [4] :

$$V(x) = \frac{1}{2}kx^2 = \frac{m}{2}\omega^2 x^2, \quad \omega = \sqrt{\frac{k}{m}}$$

Les énergies propres s'écrivent maintenant :

$$E_p^{\text{theo}} = \left(p + \frac{1}{2}\right)\hbar\omega, \quad p = 0, 1, 2, 3, \ldots$$

L'état fondamental, pour $p = 0$ a donc une énergie $E_0 = \hbar\omega/2$.

**9.** Modifier votre programme pour étudier le potentiel harmonique : on prendra $k = 1$, soit $\hbar\omega = \sqrt{2m}\sqrt{k/m} = \sqrt{2}$. Il s'agit de la forme de potentiel que vous avez déjà mise en œuvre dans l'exercice 1. Pour rappel : Il suffit de changer l'appel de la fonction du potentiel.
Répéter les opérations de l'exercice 6 avec $n = 100$ et $L = 5$ : tracer les énergies propres numériques $E_p$ avec la courbe théorique $E_p^{\text{theo}}$. Commenter le résultat.
⇒ Rendre une seule figure avec deux courbes.

**10.** Refaire le même calcul, toujours avec $n = 100$ mais cette fois-ci avec $L = 20$. Commenter sur l'impact de $\delta_x$ et $n$ par rapport aux résultats obtenus - diagnostic.
⇒ Rendre une seule figure avec deux courbes.

**11.** Tracer sur le même graphe le potentiel $V(x)$ ainsi que les premières ($p \in [0, 2]$) fonctions d'onde au carré (numériques), décalées verticalement d'une valeur égale à leur énergie $E_p$. (On pourra multiplier l'amplitude des fonctions d'onde, avant décalage de $E_p$, par un facteur pour éviter leur chevauchement et ainsi rendre le graphe bien lisible). Qu'observe-t-on quand à la "largeur" de la fonction d'onde ? Commenter.
⇒ Rendre une seule figure avec quatre courbes - le potentiel ainsi que les trois fonctions d'onde au carré.

### 4.4.4 Double puits.

L'intérêt de cet exercice n'est bien sûr pas de chercher à résoudre un problème dont la solution analytique est connue (comme le puits infini ou l'oscillateur harmonique).

4. Se reporter de nouveau au cours de physique quantique.

Cela permet juste de tester le programme que l'on a écrit et d'illustrer le cours de physique quantique.

Toutefois, le même programme, à très peu de choses près, permet de résoudre des problèmes sensiblement plus compliqués, pour lesquels les solutions analytiques sont pour l'essentiel inaccessibles. On remplace le potentiel harmonique maintenant par :

$$V(x) = a(x - r_1)(x - r_2)(x - r_3)(x - r_4)$$

soit un polynôme de degré 4, dont les racines sont $r_1$, $r_2$, $r_3$, $r_4$.
Selon le paramètrage du potentiel, il permet de simuler un double puits symétrique ou dissymétrique. Pour la suite, prenez $n = 1000$ et $L = 20$.

**12.** Comme précédemment, tracer sur un seul graphe les solutions obtenues des premiers états pour $a = 1$ et $r_1 = -2$, $r_2 = -0.5$, $r_3 = 0.5$, $r_4 = 2$, ainsi que le potentiel $V(x)$. C'est le paramètrage de votre troisième fonction de l'éxercice 1 et correspond à un double puits symétrique. Quel est le sens physique d'un tel potentiel, quelles appliquations pourrait-il avoir ? Commenter ensuite sur la forme des premiers états propres du système. Retrouvez-vous dans vos résultats des effets que vous connaissez déjà, par exemple de la physique moléculaire ?
⇒ Rendre une seule figure avec plusieurs courbes. Vérifier que toutes les courbes sont bien lisibles, qu'elles sont déplacées proportionellement à leurs énergies et qu'elles sont étiquetées dans la légende.

### 4.4.5 *Pour aller plus loin :* Double puits particuliers

**13.** Tracer les solutions obtenues pour $a = 400$ et $r_1 = -2$, $r_2 = -0.5$, $r_3 = 0.5$, $r_4 = 2$ (double puits symétrique plus profond). Commenter et, bien sûr, comparer avec le cas précédent.

**14.** Tracer maintenant les solutions obtenues pour $a = 1$ et $r_1 = -2$, $r_2 = -0.5$, $r_3 = 0$, $r_4 = 2$, ainsi que le potentiel $V(x)$. C'est un potentiel correspondant à un double puits dissymétrique. Commenter et, bien sûr, comparer avec le double puits symétrique avec $a = 1$.