Simulations Numériques en Physique

Hand ZENIA
Université A. MIRA de Béjaïa
Faculté des Sciences Exactes
Département de Physique

9 octobre 2016

Avant-propos

Ces notes on été rédigées à partir d'un cours que j'ai enseigné au étudiants de Master 1 ("Théorique" et "Matériaux et Nano-composites") durant le deuxième semestre de l'année universitaire 2015/2016. L'intitulé du cours est "Informatique", mais ses concepteurs ont laissé quelque peu la liberté à l'enseignant de confectionner son programme. C'est comme cela que j'ai opté pour des sujets qui soient communs aux deux spécialités de Master, à savoir la mécanique quantique et la physique statistique.

Le cours s'est fait à raison d'une heure et demi par semaine, et le temps était juste assez pour aborder les sujets contenus dans ces notes. Le cours comporte également des travaux pratiques à raison d'une heure et demi par semaine. Mais là, le temps n'était pas suffisant pour écrire et exécuter de très longs programmes. Le choix a été donc fait d'opter pour un langage de programmation interprété, cela permettant de réduire énormément les temps impartis pour l'écriture et le débogage des programmes. J'ai choisi Matlab comme langage, à cause de sa simplicité, mais également parce qu'il existe un émulateur en version libre de celui-ci, à savoir Octave.

Les notes sont organisées comme suit : le premier chapitre est consacré à l'initiation au système Matlab/Octave par l'intermédiaire de quelques notions et commandes de bases. Le deuxième chapitre est un rappel de quelques méthodes numériques, mais où l'accent est plutôt mis sur leur implémentation dans le langage de Matlab/Octave. Dans le troisième chapitre l'équation de Schrödinger à une dimension est étudiée dans ces deux versions : stationnaire et dépendante du temps. Le quatrième chapitre est une introduction à la méthode de Monte Carlo. Cette méthode est illustrée par son application pour le calcul des intégrales définies, puis elle a été utilisée pour étudier un modèle magnétique simple : le modèle d'Ising.

Table des matières

Chapitre 1

Introduction à Matlab/Octave

1.1 Introduction et historique

Matlab est un langage de programmation utilisé essentiellement pour des calculs numériques. C'est un langage interprété et n'a donc pas besoin de compilation, comme c'est le cas pour le Fortran ou le C. Il a été créé initialement pour l'algèbre linéaire d'où son nom Matrix Laboratory. A l'époque, début des années 80, l'idée était de faciliter l'utilisation des puissantes librairies numériques d'algèbre linéaire LINPACK et EISPACK, écrites elles en Fortran. La librairie LAPACK y a été ajoutée par la suite en 2000. En plus de l'algèbre linéaire, Matlab contient un grand nombre d'algorithmes, tels que la résolution numérique des équation différentielles. Matlab est un logiciel propriétaire développé par la société The MathWorks. Vu la popularité de Matlab, des personnes, John W. Eaton en l'occurrence, ont décidé d'écrire un langage de programmation qui aura la même syntaxe que Matlab, mais sera un logiciel libre. Et c'est comme cela qu'est né Octave. La version 1.0 du logiciel est apparue en 1994. Octave est toujours maintenu pour le projet GNU par John W. Eaton.

1.2 Quelques commandes de base

On donne ici quelques commandes simple pour s'initier au système Matlab/Octave. Pour plus de détail l'étudiant est conseillé de consulter les manuels en-ligne de Matlab [1] et d'Octave [2].

— On déclare et définit une variable a au même temps, comme ceci

```
> a=2.5
a =  2.5000
```

On dit qu'on a affecté la valeur `2.5` à la variable `a`.

— On peut maintenant faire subir à a toute sorte d'opérations arithmétiques : on peut par exemple ajouter un nombre à a, ou bien calculer a^2. Voici ce qu'on peut faire

```
> a+3
ans =  5.5000
> a*a
ans =  6.2500
> a^2
ans =  6.2500
```

1. Voir le site https://www.mathworks.com/help/matlab/getting-started-with-matlab.html.
2. Voir le site https://www.gnu.org/software/octave/doc/interpreter/.

On remarque deux choses ici. La première est que la "réponse" du système est affichée sous forme de `ans =` . La variable `ans` porte ainsi la dernière valeur retournée ou bien calculée par le système. La deuxième remarque concerne le calcul du carré. Sous Matlab/Octave, on peut soit faire `a*a`, soit `a^2`. Le signe `*` est un opérateur signifiant multiplication, et `^` est un opérateur signifiant élever à la puissance.

— On peut affecter le résultat d'opérations sur une ou plusieurs variables à une autre variable :

```
> a=2.5
a =  2.5000
> b=-3.1
b = -3.1000
> c=a^2+b
c =  3.1500
```

— Les fonctions mathématiques sinus, cosinus, tangente, racine carrée, exponentiel, logarithme naturel ... sont appelées respectivement `sin`, `cos`, `tan`, `sqrt`, `exp`, `log` ...

```
> sqrt(25)
ans =  5
```

— Les constantes $i = \sqrt{-1}$ et π ont pour noms respectivement `i` et `pi`. Comme exemple, $e^{i\pi} + 1$ donne

```
> exp(i*pi)+1
ans =  0.0000e+00 + 1.2246e-16i
```

Le résultat $0 + 1.2246 \times 10^{-16} i$ est pratiquement zéro, puisque le plus petit nombre réel ou flottant que le système peut représenter, et qui s'appelle `eps`, est

```
> eps
ans =  2.2204e-16
```

C'est à dire $\epsilon = 2.2204 \times 10^{-16}$.

— Si on veut que le système n'affiche pas le résultat d'une instruction, on termine celle-ci par un `;`.

```
> a=log(2)
a =  0.69315
> a=log(2);
```

La deuxième instruction, qui est la même que la première, mais terminée par un `;` n'a pas produit de sortie.

1.3 Vecteurs et matrices

1.3.1 Vecteurs

— Pour déclarer un vecteur ligne :

```
> V=[1 2 3];
```

— Pour un vecteur colonne :

```
> V=[1; 2; 3]
```

— Pour accéder aux éléments individuellement : on utilise `V(i)` pour designer l'élément V_i du vecteur V.

1.3.2 Matrices

— Pour déclarer une matrice

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix},$$

on fait

```
> A=[1  2; 3  4]
A =
   1   2
   3   4
```

— Pour accéder aux éléments individuellement : on utilise `A(i,j)` pour designer l'élément A_{ij} de la matrice A.

— Les notations `A(i,:)` et `A(:,j)` désignent respectivement la i^{eme} ligne et la j^{eme} colonne de `A`.

1.3.3 Arithmétique

— Pour calculer la somme, la différence, et le produit de deux matrices, on utilise respectivement les opérateur `+`, `-`, et `*`.

— La transposée de la matrice `A` est donnée par `A.'`.

— La transposée conjuguée de `A` est donnée par `A'`.

— Si on veut opérer par élément, on précède les opérateurs d'un point. Si par exemple on veut un produit élément par élément on utilise `.*`. Pour une division élément par élément, on utilise `./`. Et pour élever chaque élément à une puissance, on utilise `.^`. Exemples

```
> A=[1  2; 3  4];
> A.^2
ans =
    1    4
    9   16
> A^2
ans =
    7   10
   15   22
```

— Pour multiplier un vecteur par une matrice on utilise l'opérateur `*`.

```
> V=[1  ; 2]
V =
   1
   2
> A=[1  2; 3  4]
A =
   1   2
   3   4
octave:7> A*V
ans =
    5
   11
```

— On peut aussi multiplier à gauche, auquel cas on utilise un vecteur ligne. Ici on utilise la transposée de `V`, et le résultat est un vecteur ligne.

```
> V'*A
ans =
    7    10
```

1.3.4 Algèbre linéaire

— Pour résoudre le système $Ax = b$, on écrit `x=A\b`.
— Pour calculer l'inverse d'une matrice `A` on utilise la fonction `inv`. Exemple : `B=inv(A)`.
— Pour trouver les valeurs propres de `A`, on écrit `D=eig(A)`. D est alors un vecteur colonne qui contient les valeurs propres de A. Dans le cas où celles-ci sont toutes réelles, elle sont données dans un ordre ascendant.
— L'instruction `[V D]=eig(A)` retourne deux matrices. La matrice `D` est diagonale et contient les valeurs propres. La matrice `V` contient les vecteurs propres. Chaque colonne `j` de `V` est un vecteur propre associé à la valeur propre `D(j,j)`. Voici un exemple

```
> A=[1  2; 2  1]
A =
   1   2
   2   1
> [V D]=eig(A)
V =
  -0.70711   0.70711
   0.70711   0.70711
D =
Diagonal Matrix
  -1   0
   0   3
```

1.4 Graphisme

1.4.1 Courbes à deux dimensions

Voici un exemple de tracé d'une courbe $y = \sin(3x)$ dans l'intervalle $[0, \pi]$

```
> x=linspace(0,pi,100);
> y=sin(3*x);
> plot(x,y,'r','LineWidth',4)
> axis([0,pi,-1,1])
> xlabel("x")
> ylabel("y")
> grid on
```

La fonction `linspace` permet de créer un vecteur de nombres réels dans l'intervalle allant de son premier à son deuxième argument, et de longueur spécifiée par le troisième argument. Ici la variable `x` est un vecteur de 100 nombres réels répartis dans l'intervalle $[0, \pi]$. La variable `y` est un autre

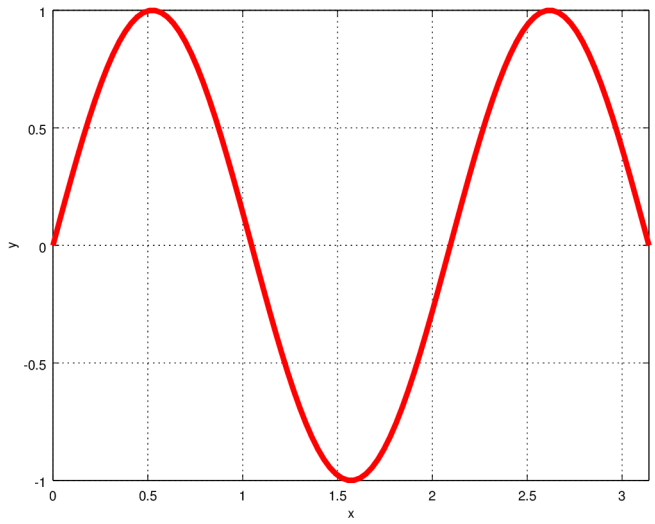

FIGURE 1.1 – *Courbe* $y = \sin(3x)$ *dans l'intervalle* $[0, \pi]$ *tracée avec la fonction* `plot` *de Matlab/Octave.*

vecteur dont les éléments sont fonctions des éléments de `x`. L'argument `'r'` de la fonction `plot` détermine la couleur (ici rouge), et `'LineWidth'`, suivi d'un nombre détermine l'épaisseur de la ligne représentant la courbe. La fonction `axis` permet de modifier les valeurs minimale et maximale sur les axes. Les fonctions `xlablel` et `ylabel` permettent d'écrire sur les axes, respectivement, horizontal et vertical. Enfin l'instruction `grid on` permet de rajouter un maillage au graphe. Après avoir exécuté les instructions, on obtient la figure 1.1.

1.4.2 Courbes à trois dimensions

On va utiliser la fonction `mesh` pour tracer une courbe à trois dimension. La courbe est donnée par

$$z = \frac{\sin(r)}{r},$$

avec $r = \sqrt{x^2 + y^2}$ et $(x, y) \in [-8, 8] \times [-8, 8]$.

```
> [x y]=meshgrid(-8:0.1:8);
> r=sqrt(x.^2+y.^2)+eps;
> z=sin(r)./r;
> mesh(x,y,z)
> xlabel("x")
> ylabel("y")
> zlabel("z")
```

La première instruction crée deux matrices : `x` et `y`. Les colonnes de `x` contiennent les valeurs allant de -8 à 8 avec un pas de 0.1. Les nombres dans une colonne sont tous identiques. La matrice `y` est la transposée de `x` dans ce cas. `r` et `z` sont deux matrices de la même taille que `x` et `y`. La courbe est représentée dans la figure 1.2.

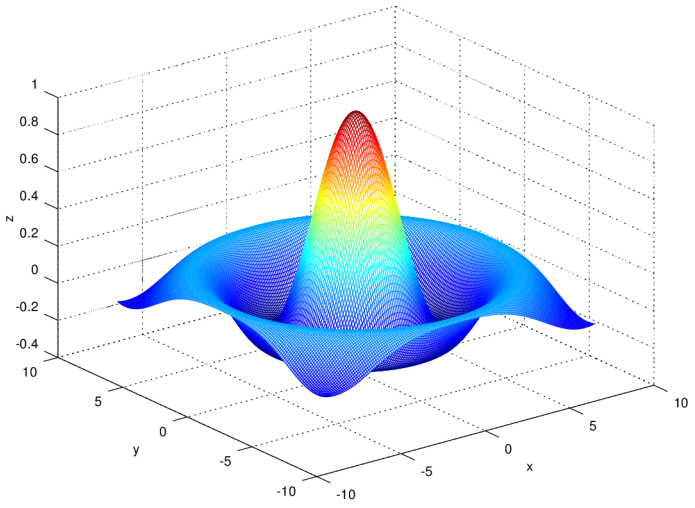

FIGURE 1.2 – *Courbe de* $z = \sin(r)/r$, *avec* $r = \sqrt{x^2 + y^2}$ *et* $(x, y) \in [-8, 8] \times [-8, 8]$, *tracée avec la fonction* `mesh` *de Matlab/Octave.*

1.5 Les scripts

Le travail en mode interactif n'est pratique que lorsqu'on exécute des instructions simples. Mais si on a besoin d'exécuter un grand nombre d'instructions à la fois, et si on veut les répéter une autre fois, alors il est possible d'écrire ces instruction dans un fichier. C'est ce qu'on appelle un script. Voici un script simple écrit dans un fichier appelé `monscript.m` :

```
a=1;
b=-3;
c=2;
delta=b^2-4*a*c;
x1=(-b-sqrt(delta))/(2*a)
x2=(-b+sqrt(delta))/(2*a)
```

Il faut sauvegarder le fichier `monscript.m` dans le répertoire de travail. Pour exécuter le script, on fait alors

```
> monscript
x1 =  1
x2 =  2
```

Remarques :

- Comme pour la ligne de commande, dans un script aussi, si une instruction n'est pas terminée par un `;`, la valeur retournée par l'instruction est affichée à l'écran au moment de l'exécution. C'est l'exemple des deux lignes donnant `x1` et `x2` dans le présent script.
- Lorsqu'on écrit des scripts, il est fortement recommandé d'utiliser des commentaires afin de faciliter la compréhension de ce qui se fait à des endroits critiques du programme. Les commentaires en Matlab/Octave peuvent être placés partout, et il sont obligatoirement précédés du signe `%`.

<	plus petit	<=	plus petit ou égal	&	et
>	plus grand	>=	plus grand ou égal	|	ou
==	égal	~=	pas égal	~	pas

Table 1.1 – *Opérateurs logiques dans Matlab/Octave.*

— Il arrive qu'une instruction est trop longue pour être contenue dans une seul ligne de texte. On peut alors écrire une instruction s'étalant sur plusieurs lignes, et pour marquer cette continuité, on utilise ... à chaque fin de ligne du texte.

1.6 Contrôle

1.6.1 La boucle for

Supposons qu'on veuille multiplier un vecteur `xvec` de `ndim` éléments par une matrice carrée `ndim x ndim` appelée `amat` pour obtenir un vecteur appelé `yvec` de `ndim` éléments :

$$\mathbf{A}\,\vec{x} = \vec{y} \longrightarrow y_i = \sum_{j=1}^{n} A_{i,j}\,x_j, \qquad i = 1, 2, ...n.$$

Pour ce faire, on ferra appel à la boucle `for` comme suit :[3]

```
for i=1:ndim
  yvec(i)=0;
  for j=1:ndim
    yvec(i) = yvec(i) + amat(i,j)*xvec(j);
  end
end
```

On a ici deux boucles, une imbriquée dans l'autre. Chaque instruction `for` doit être terminée par une instruction `end`. Le premier `end` ferme le deuxième `for`, et le deuxième `end` ferme le premier `for`.

1.6.2 L'instruction if

Avec l'instruction `if` on teste une proposition logique, en fonction de quoi on détermine les opérations à effectuer. Le plus souvent on fait appel à des opérateurs logiques qui permettent la comparaison entre deux entités. Ces opérateurs sont montrés dans le tableau 1.1. L'instruction `if` à elle seul ne donne que le choix de faire une chose[4] ou pas. Pour permettre d'avantage de choix on utilise l'instruction `else`.

Comme exemple on se propose ici de résoudre l'équation quadratique

$$a\,x^2 + b\,x + c = 0.$$

Le code suivant, en utilisant l'instruction `if` permet de résoudre l'équation en tenant compte de toutes les éventualités quant aux valeurs prises par les coefficients :

3. Ceci n'est bien sur qu'un exemple pour illustrer l'utilisation de la boucle `for`. En effet, et on l'a vu plus haut, pour effectuer cette multiplication dans Matlab/Octave, il suffit d'écrire `yvec=amat*xvec`.

4. Qui peut au être au fait une série de plusieurs instructions.

```
if a==0
  if b==0
    printf('Il n''y a pas de solution.\n')
  else
    x1=-c/b;
    printf('Solution unique : x1 = %f.\n',x1);
  end
else
  delta=b^2-4*a*c;
  if delta < 0
     printf('Il n''y a pas de solutions réelles.\n')
  else
    x1= (-b - sqrt(delta))/(2*a);
    x2= (-b + sqrt(delta))/(2*a);
    printf('Les solutions sont : x1 = %f et x2 = %f.\n',x1,x2);
  end
end
```

A remarquer ici qu'on utilise deux signes `=` pour tester l'égalité entre deux valeurs. Ainsi `a==b` retourne vrai si `a` est égal à `b`, et faux, sinon (voir tableau 1.1). Mais l'instruction `a=b` (avec un seul signe `=`) affecte la valeur de la variable `b` à la variable `a`.

1.7 Les fonctions

Une fonction est un sous-programme qui effectue une tâche particulière. Dans Matlab, une fonction est écrite dans un fichier à part, et qui porte le nom de la fonction avec une extension ".m". Voici un exemple d'une fonction qui calcule la moyenne de valeurs introduites sous forme d'un vecteur :

```
function m=moyenne(x)
  m=sum(x)/length(x);
```

Ici `x` est appelé argument de la fonction. C'est la variable qu'il faut fournir à la fonction lorsqu'on l'appelle. On le voit bien que dans cette fonction elle-même on a fait appel à deux autres fonctions, qui sont elles implémentées dans Matlab/Octave : la fonction `sum` retourne la somme des éléments d'un vecteur, et `length` retourne la longueur ou le nombre d'éléments dans un vecteur. Pour utiliser fonction `moyenne` qu'on vient d'écrire, on se place d'abord dans le dossier où se trouve le fichier `moyenne.m`.[5] Si, par exemple, on veut utiliser cette fonction pour calculer la valeur moyenne de $A = [1\,2\,3\,4]$, on fait

```
> A=[1  2  3  4];
> moyenne(A)
ans =   2.5000
```

ou bien

```
> moyenne([1  2  3  4])
ans =   2.5000
```

5. Ceci n'est pas nécessaire si le fichier se trouve dans un des répertoires où Matlab/Octave cherche les fichiers fonctions. On peut en effet rajouter des chemins vers nos répertoires personnels et Matlab/Octave cherchera alors dans ces nouveaux chemins à chaque fois qu'on appelle une fonction. Ceci se fait à l'aide de la commande `addpath`.

On peut bien sûr écrire des fonctions qui acceptent plus d'un argument et/ou qui retourne plus d'une valeur. On peut modifier la fonction `moyenne` pour retourner le nombre de valeurs dans le vecteur, en plus de la moyenne. On a alors ceci

```
function [m l]=moyenne_mod(x)
 l=length(x);
 m=sum(x)/l;
```

On a le choix maintenant d'appeler la fonction pour qu'elle retourne une seul valeur, qui sera alors la moyenne, ou bien deux valeurs, c'est à dire la moyenne et la longueur du vecteur d'entrée

```
> moyenne_mod([1 2 3 4])
ans =  2.5000
> [moy long]=moyenne_mod([1 2 3 4])
moy =  2.5000
long =  4
```

1.8 Entrées/sorties

Il existe dans Matlab/Octave un grand nombre de fonctions dédiées à la lecture et à l'écriture de données. Dans ce qui suit on verra un échantillon très réduit de fonctions qui permettent de réaliser les opérations d'écriture et de lecture, que ce soit en utilisant les entrées et sorties standards (écran et clavier) ou bien des fichiers formatés sur le disque.

1.8.1 Ecriture à l'écran

Pour écrire à l'écran on utilise la fonction `printf`. Voici un exemple de comment imprimer un nombre entier, un nombre réel (flottant), et une chaîne de caractères :

```
n=10;
a=2.3;
c='exemple';
printf('n = %i, a = %f. Ceci est un %s.\n',n,a,c);
```

On obtient

```
n = 10, a = 2.300000. Ceci est un exemple.
```

On voit que le premier argument de la fonction `printf` détermine quelle est la forme que prendra la sortie. Le signe (`%`) indique qu'une valeur doit être mise à sa position, et la lettre qui suit indique la nature de la variable : `i` pour un entier, `f` pour un flottant, et `s` pour une chaîne de caractères. Enfin, le signe `\n` veut dire tout simplement qu'il faut rajouter un retour ou une fin de ligne à la fin.

1.8.2 Lecture à partir du clavier

Pour lire à partir du clavier, on utilise la fonction `input`. Voici un petit script appelé `input_exemple.m` pour illustrer la fonction

```
n=input("Entrer un nombre ");
printf('n = %i, n^2 = %i.\n',n,n^2);
```

Et voici ce qui se passe lorsqu'on utilise ce script

```
> input_exemple
Entrer un nombre 3
n = 3, n^2 = 9.
```

1.8.3 Ecriture dans un fichier

Pour écrire dans un fichier on utilise la fonction `fprintf`. Cette commande est semblable à `printf`, à la seule différence près que `fprintf` requiert un argument de plus : l'identifiant du fichier dans lequel on veut écrire. Voici un exemple

```
n=10;
a=2.3;
c='exemple';
fid = fopen('resultats.dat','w');
fprintf(fid,'n = %i, a = %f. Ceci est un %s.\n',n,a,c);
fclose(fid);
```

Ici on utilise la fonction `fopen` pour ouvrir un fichier du nom de `resultats.dat` en écriture (d'où l'argument `'w'` pour write en anglais). Le résultat de cette ouverture est un identifiant auquel on donne le nom `fid`. Ensuite on donne cet identifiant comme premier argument à la fonction `fprintf`, suivi du format, et puis des variables à écrire. Enfin, on utilise la fonction `fclose` pour fermer le fichier. En exécutant ces commandes, un nouveau fichier `resultats.dat` est crée dans le répertoire de travail et contient ceci

```
n = 10, a = 2.300000. Ceci est un exemple.
```

1.8.4 Lecture à partir d'un un fichier

On utilise la fonction `fscanf` pour lire des données à partir d'un fichier. Supposons qu'un fichier appelé `input.dat` contienne ces données

```
10   2.3   'test'
```

Le script suivant appelé `fscanf_exemple.m` permet de lire les données et de les sauvegarder dans des variables appropriées

```
clear all
fid = fopen('input.dat','r');
taille = [1 1];
n=fscanf(fid,'%i',taille);
a=fscanf(fid,'%f',taille);
c=fscanf(fid,'%s',taille);
fclose(fid);
printf('n = %i, a = %f, c = %s.\n',n,a,c);
```

En exécutant le script on obtient

```
> fscanf_exemple
n = 10, a = 2.300000, c = 'test'.
```

On voit que la fonction `fscanf` admet en général trois arguments : l'identifiant du fichier à lire, le format, et la taille du champs à lire.

Chapitre 2

Rappels de Quelques Méthodes Numériques

2.1 Résolution des équations non-linéaires

2.1.1 Méthode de la bissection

La méthode de la bissection permet de chercher le zéro d'une fonction $f(x)$ dans un intervalle donné $[a, b]$ et avec une précision donnée ϵ. L'idée est de cerner d'abord le zéro dans un intervalle $[a, b]$. Ensuite pour avoir une meilleur idée sur la position exacte du zéro, on subdivise l'intervalle $[a, b]$ en deux. On calcule $c = (a+b)/2$. Si la fonction change de signe entre a et c , la racine se trouve dans l'intervalle $[a, c]$, qui devient alors le nouvel intervalle $[a, b]$. Sinon, elle se trouve dans l'intervalle $[c, b]$, qui devient alors le nouvel intervalle $[a, b]$[1, 2]. On continue ainsi jusqu'à ce que $|f(c)| < \epsilon$, ou bien la largeur de l'intervalle elle-même soit inférieure à ϵ. Voici un exemple d'implémentation de la méthode :

```
function [sol niter]= bissec(a,b,nitermax,tol,func)
 fa=func(a);
 fb=func(b);
 if fa*fb>0
  error('Erreur: il n''y a pas de solution dans l''intervale donne')
 end
 for iter=1:nitermax
    c=(a+b)/2;
    fc=func(c);
    if abs(fc) < tol
       sol=c;
       niter = iter;
       break;
    else
       if fa*fc < 0
         b=c;
         fb=fc;
       else
         a=c;
         fa=fc;
       end
    end
```

```
end
if iter == nitermax
   printf('Pas de solution trouvee')
end
```

Les arguments `a` et `b` sont les bornes de l'intervalle dans lequel on cherche la solution, `nitermax` et le nombre d'itérations maximal à effectuer, `tol` est la tolérance ou la précision avec laquelle on veut déterminer la solution, et enfin `func` est la fonction dont on cherche la racine. Les variables de sortie sont `sol` qui contient la racine recherchée, et `niter` qui contient le nombre d'itérations effectuées par la bissection pour trouver la racine. Comme exemple, on cherche à calculer la racine carrée de 5 par cette méthode. La fonction à résoudre est alors

$$f(x) = x^2 - 5.$$

On doit d'abord créer un fichier `mafonction.m` qui contiendra ceci

```
function res = mafonction(x)
  res = x*x-5;
end
```

Puis on fixe les autres paramètres et on appelle la bissection comme suit

```
> [sol niter] = bissec(2.2,2.3,100,1.0e-06,@mafonction)
sol =  2.2361
niter =  17
```

On peut faire usage également de la notion de fonction anonyme– sans avoir à créer un fichier fonction– et faire ceci

```
> [sol niter] = bissec(2.2,2.3,100,1.0e-06,@(x) x^2-5)
sol =  2.2361
niter =  17
```

2.1.2 Méthode de Newton-Raphson

La méthode de Newton-Raphson consiste à chercher le zéro d'une fonction $f(x)$ de manière itérative. On commence d'abord par estimer une valeur approchée de la solution x_0. La valeur approchée de la solution x_{i+1} à l'itération $i+1$ est donnée en fonction de la valeur approchée à l'itération précédente x_i par

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)},$$

où f' est la dérivée de la fonction f[1, 2]. La recherche est arrêtée si à l'itération i on a $|f(x_{i+1})| < \epsilon$, ou bien le nombre maximal d'itérations est atteint. En effet, la méthode de Newton-Raphson peut ne pas converger vers la solution recherchée. Voici une implémentation de la méthode :

```
function [sol niter] = newt(x0,nitermax,tol,fun,funp)
 c=x0;
 fc=fun(c);
 for iter=1:nitermax
    c=c-fc/funp(c);
    fc=fun(c);
```

```
    if abs(fc) < tol
      sol=c;
      niter=iter;
      break;
    end
  end
  if iter==nitermax
     printf('Pas de solution')
  end
```

L'argument `x0` est la valeur approchée initiale de la racine, `nitermax` est le nombre d'itérations maximal à effectuer, `tol` est la tolérance ou la précision sur la racine, `fun` est la fonction pour laquelle on cherche le zéro, et `funp` est la dérivée de cette fonction. Pour reprendre l'exemple donné précédemment du calcul de la racine de 5 on fait

```
> [sol niter] = newt(2,100,1.0e-06,@(x) x^2-5, @(x) 2*x)
sol =  2.2361
niter =  3
```

On voit que la méthode de Newton-Raphson converge au bout de 3 itérations, alors qu'il en a fallu 17 pour la méthode de la bissection. En effet, la méthode de Newton-Raphson converge beaucoup plus rapidement, en général, que la méthode de la bissection. Le seul inconvénient est que la méthode de Newton-Raphson ne converge pas toujours, contrairement à la méthode de la bissection.

2.2 Quadrature

2.2.1 Méthode des trapèzes

La méthode des trapèzes est une méthode pour une estimation numérique d'une intégrale définie

$$I = \int_a^b f(x)\,dx.$$

On subdivise l'intervalle $[a, b]$ en N sous-intervalles égaux de largeur $h = (b - a)/N$. La valeur de l'intégrale est alors la somme des contributions de chacun des sous-intervalles. Dans un sous-intervalle $[x_i, x_{i+1}]$ la fonction est interpolée linéairement et son intégral est estimée comme étant égale à l'aire du trapèze formé par les points $(x_i, 0)$, $(x_{i+1}, 0)$, $(x_i, f(x_i))$, et $(x_{i+1}, f(x_{i+1}))$. La méthode des trapèzes revient à estimer l'intégrale d'une fonction comme l'intégrale de son interpolation linéaire par intervalles[1, 2, 3]. Enfin, on trouve l'estimation donnée par cette méthode

$$I = \int_a^b f(x)\,dx \approx \frac{b-a}{N}\left(\frac{f(a)+f(b)}{2} + \sum_{i=1}^{N-1} f(x_i)\right),$$

où $x_i = a + i \times h$.

La méthode est facile à implémenter et en voici un exemple sous forme d'une fonction :

```
function somme = trapezes(func,a,b,npoints)
  h=(b-a)/npoints;
  somme=(func(a)+func(b))/2;
  xi=a;
```

```
  for i=1:npoints-1
      xi=xi+h;
      somme=somme+func(xi);
  end
  somme=somme*h;
```

L'argument `func` est la fonction à intégrer, `a` et `b` sont les bornes d'intégration, et `npoints` est le nombre de subdivisions ou nombre de pas d'intégration. Comme exemple, on va calculer une valeur approchée de

$$I = \int_0^{\frac{\pi}{2}} \sin(x)\, dx,$$

avec $N = 100$. Le calcul est fait comme ceci

```
> int = trapezes(@sin,0,pi/2,100)
int =  0.99998
```

La valeur exacte de l'intégrale est bien sûr $I_{ex.} = 1$.

2.2.2 Méthode de Gauss-Legendre

Les méthodes de quadrature en général consistent à choisir N points x_i dans l'intervalle d'intégration $[a\, b]$, à associer des poids w_i à chacun des points, et à écrire finalement

$$\int_a^b f(x)\, dx \approx \sum_{i=1}^{N} w_i f(x_i).$$

En inspectant la méthode des trapèzes par exemple, on voit que $x_i = a + i \times h$, et que $w_i = h$ pour les points intérieurs, et $w_i = h/2$ pour les deux points a et b. La méthode de quadrature de Gauss d'ordre N est construite de façon à donner un résultat exacte pour des polynômes de degré $2N - 1$ ou moins, avec un choix approprié des points x_i et des poids w_i pour $i = 1, 2...N$[2]. Par convention, on prend le domaine d'intégration dans la méthode de Gauss comme étant $[-1\, 1]$. En effet il est facile de ramener toute intégrale à une intégrale sur $[-1\, 1]$, y compris celles ayant l'∞ comme borne d'intégration. Dans la méthode de Gauss les points x_i sont choisis comme étant les zéros d'un polynôme appartenant à une classe de polynômes orthogonaux. Dans la méthode de Gauss-Legendre ces polynômes orthogonaux sont justement ceux de Legendre. Les abscisse x_i de la méthode d'ordre N sont les racines/zéros du polynôme de Legendre $P_N(x)$, et apparaissent de manière symétrique par rapport à zéro. Les poids w_i sont alors donnés par [1]

$$w_i = \frac{2(1 - x_i^2)}{(N+1)^2 [P_{N+1}(x_i)]^2}$$

Une fois qu'on a les abscisses et les poids correspondants, il est facile d'implémenter la méthode de Gauss-Legendre sous Matlab/Octave. En voici un exemple

```
function somme = gausslegendre(func,a,b,xxi,wwi)
  somme=0;
  for i=1:length(xxi)
    somme=somme+wwi(i)*func((b-a)*(xxi(i)+1)/2+a);
  end
  somme=somme*(b-a)/2;
```

1. Voir par exemple http://mathworld.wolfram.com/Legendre-GaussQuadrature.html.

x_i	w_i
0	$\frac{128}{225}$
$\pm\frac{1}{21}\sqrt{245-14\sqrt{70}}$	$\frac{1}{900}(322+13\sqrt{70})$
$\pm\frac{1}{21}\sqrt{245+14\sqrt{70}}$	$\frac{1}{900}(322-13\sqrt{70})$

TABLE 2.1 – *Abscisses et poids de la méthode de Gauss-Legendre d'ordre 5.*

Les arguments de la fonction `gausslegendre` sont : `func` la fonction à intégrer, `a` et `b` les bornes d'intégration, `xxi` et `wwi` les abscisses et les poids respectivement. Pour la méthode d'ordre 5, les abscisses et les poids sont montrés dans le tableau 2.1. On va les utiliser pour calculer

$$I = \int_0^{\frac{\pi}{2}} \sin(x)\, dx.$$

Voici les instructions

```
> xi=[0  sqrt(245-14*sqrt(70))/21  -sqrt(245-14*sqrt(70))/21
      -sqrt(245+14*sqrt(70))/21  sqrt(245+14*sqrt(70))/21];
> wi=[128/225  (322+13*sqrt(70))/900  (322+13*sqrt(70))/900
      (322-13*sqrt(70))/900  (322-13*sqrt(70))/900];
> int = gausslegendre(@sin,0,pi/2,xi,wi)
int =  1.0000
```

2.2.3 Intégrales dont une borne est ∞

Il ne s'agit pas de présenter ici une nouvelle méthode de quadrature, mais juste une astuce pour calculer des intégrales dont l'une ou les deux bornes sont ∞. Supposons qu'on veuille calculer

$$I = \int_0^{\infty} f(x)\, dx.$$

On peut alors diviser l'intégrale en deux contributions :

$$I_1 = \int_0^{1} f(x)\, dx,$$

et

$$I_2 = \int_1^{\infty} f(x)\, dx.$$

La première contribution est laissée comme telle, mais dans la deuxième on opère un changement de variable

$$x = 1/y,$$

ce qui nous donne

$$I_2 = \int_0^{1} \frac{f(1/y)}{y^2} dy.$$

Et enfin, en recombinant les deux contributions, on a

$$I = \int_0^{\infty} f(x)\, dx = \int_0^{1} dx \left[f(x) + \frac{f(1/x)}{x^2} \right].$$

Il est facile maintenant de généraliser cette procédure à d'autres cas similaires. Comme exemple, on va utiliser la méthode des trapèzes pour calculer

$$I = \int_0^\infty e^{-x^2} dx,$$

et comparer le résultat à la valeur exacte $I_{ex.} = \sqrt{\pi}/2$.

```
> err = trapezes(@(x) exp(-x^2)+exp(-1/x^2)/x^2,1e-06,1,100) ...
                    -sqrt(pi)/2
err =    -7.1314e-06
```

Pour éviter une division par zéro, on a pris 10^{-6} comme borne inférieure de l'intégrale transformée.

2.3 Equations différentielles ordinaires

2.3.1 Runge-Kutta pour une équation de premier ordre

On s'intéresse ici à la méthode de Runge-Kutta d'ordre 2 (RK2) et à son implémentation dans Matlab/Octave. Dans cette méthode on cherche à intégrer numériquement l'équation

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

dans l'intervalle $[a\ b]$. Pour ce faire, on subdivise l'intervalle en N sous-intervalles de largeur $h = (b-a)/N$ chacun. On obtient ainsi des points $t_i = a + i \times h$, (avec $i = 0, 1, 2, \ldots N$) auxquels on veut évaluer $y(t_i)$, qu'on appellera simplement y_i. Dans la méthode RK2, on calcule y_{i+1} en fonction de t_i et de y_i comme suit[1, 2]

$$y_{i+1} = y_i + h\, f\left(t + \frac{h}{2}, y_i + \frac{h}{2} f(t_i, y_i)\right).$$

Comme il s'agit d'une équation de premier ordre, il faut fournir la condition initiale y_0 pour pouvoir calculer le reste des valeurs y_i pour $i = 1, 2, \ldots N$. La fonction `rk2_1` suivante est une implémentation de la méthode

```
function  y = rk2_1(t,fun,y0)
 h=t(2)-t(1);
 y=zeros(1,length(t));
 y(1)=y0;
 for m=1:length(t)-1
     y(m+1) = y(m) + h*fun(t(m)+h/2,y(m)+h*fun(t(m),y(m))/2);
 end
```

Les arguments de la fonction sont : `t` est un vecteur contenant les valeurs t_i, `fun` est la fonction $f(t, y)$, et `y0` est la valeur initiale de y. On teste maintenant cette implémentation en intégrant numériquement l'équation

$$\frac{dy}{dt} = -y,$$

dans l'intervalle [0 6], avec la valeur initiale $y_0 = y(0) = 1$.

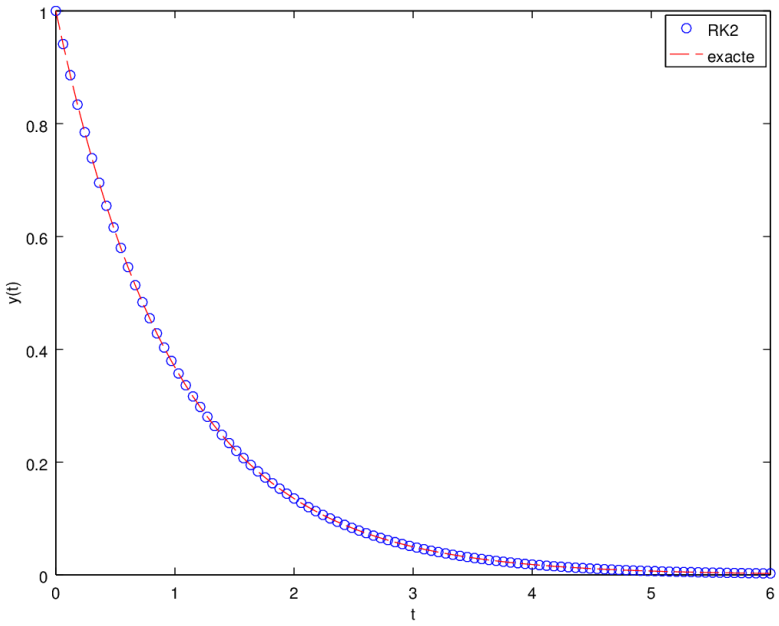

FIGURE 2.1 – *Comparaison de la solution numérique, obtenue avec la méthode RK2, avec la solution exacte de l'équation* $dy/dt = -y$.

```
clear all
Np=50;
t=linspace(0,6,100);
y0=1;
y = rk2_1(t,@fun,y0);
plot(t,y,"o;RK2;",t,exp(-t),"r--;exact;")
xlabel("t")
ylabel("y(t)")
```

La fonction `fun` est donnée par

```
function ret = fun(t,y)
 ret = -y;
```

En exécutant le script, on obtient la figure 2.1 où la solution numérique est comparée à la solution exacte $y_{ex.}(t) = e^{-t}$.

2.3.2 Runge-Kutta pour une équation de second ordre

Pour pouvoir utiliser la méthode RK2 dans le cas d'une équation de second ordre, on doit d'abord transformer celle-ci en un système de deux équations de premier ordre. L'équation qu'on veut résoudre est de la forme

$$\frac{d^2y}{dt^2} = f(t, y, y'),$$

où $y' \equiv dy/dt$. On va introduire une nouvelle variable, appelée v telle que

$$\begin{aligned}\frac{dy}{dt} &= v \\ \frac{dv}{dt} &= f(t,y,v).\end{aligned}$$

On peut écrire les deux variables dépendantes y et v sous forme d'un vecteur, et écrire l'équation sous la forme

$$\frac{d}{dt}\begin{bmatrix} y \\ v \end{bmatrix} = \begin{bmatrix} v \\ f(t,y,v) \end{bmatrix}.$$

Il est facile maintenant de voir que la méthode RK2 appliquée à cette équation de premier ordre donne

$$\begin{bmatrix} y_{i+1} \\ v_{i+1} \end{bmatrix} = \begin{bmatrix} y_i \\ v_i \end{bmatrix} + h \begin{bmatrix} v_i + \frac{h}{2} f(t_i, y_i, v_i) \\ f(t_i + \frac{h}{2}, y_i + \frac{h}{2} v_i, v_i + \frac{h}{2} f(t_i, y_i, v_i)) \end{bmatrix}$$

On voit que maintenant on a besoin de deux conditions initiales pour pouvoir intégrer l'équation : il s'agit de donner y_0 et v_0, c'est à dire la valeur de la fonction y et de sa dérivée au point initial t_0. La fonction `rk2_2` suivante est une implémentation de la méthode

```
function [y v]= rk2_2(t,fun,y0,v0)
 h=t(2)-t(1);
 y=zeros(1,length(t));
 v=zeros(1,length(t));
 y(1)=y0;
 v(1)=v0;
 for m=1:length(t)-1
  v(m+1) = v(m) + h * fun(t(m)+h/2,y(m)+h*v(m)/2, ...
                v(m)+h*fun(t(m),y(m),v(m))/2);
  y(m+1) = y(m) + h*(v(m)+h*fun(t(m),y(m),v(m))/2);
 end
```

Les arguments de la fonction sont : `t` est un vecteur contenant les valeurs t_i, `fun` est la fonction $f(t,y,v)$, et `y0` et `v0` sont respectivement les valeurs initiales de y et de v. On teste maintenant cette implémentation en intégrant numériquement l'équation

$$\frac{d^2y}{dt^2} = -y,$$

dans l'intervalle [0 10], avec les valeurs initiale $y_0 = 1$ et $v_0 = 0$.

```
clear all
Np=50;
t=linspace(0,10,100);
y0=1;
v0=0;
[y v]= rk2_2(t,@fun,y0,v0);
plot(t,y,"o;RK2;",t,cos(t),"r--;exacte;")
xlabel("t")
ylabel("y(t)")
```

La fonction `fun` est donnée par

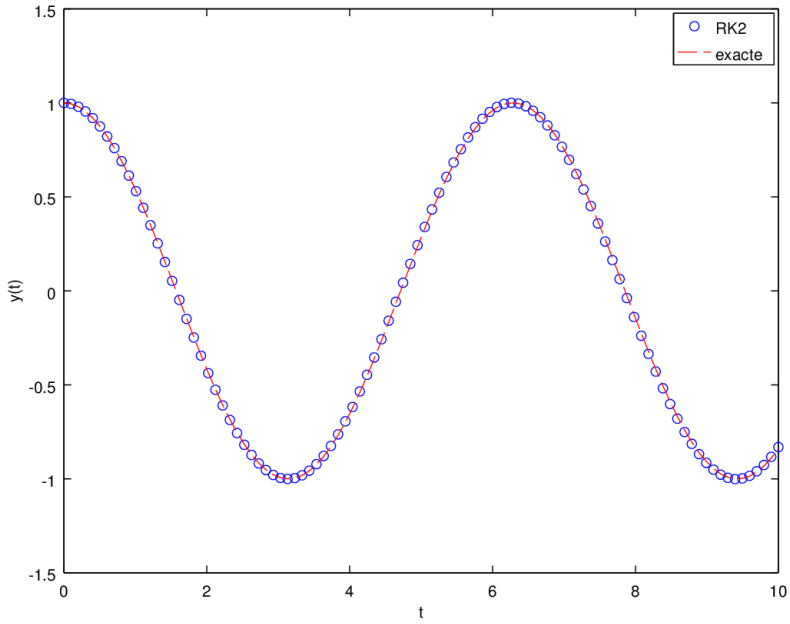

FIGURE 2.2 – *Comparaison de la solution numérique, obtenue avec la méthode RK2, avec la solution exacte de l'équation* $d^2y/dt^2 = -y$.

```
function ret = fun(t,y,v)
  ret = -y;
```

En exécutant le script, on obtient la figure 2.1 où la solution numérique est comparée à la solution exacte $y_{ex.}(t) = \cos(t)$.

Chapitre 3

Equation de Schrödinger à Une Dimension

Dans ce chapitre on va s'intéresser à l'étude de l'équation de Schrödinger, stationnaire et puis dépendante du temps, à une dimension. Dans le premier temps on utilisera différentes méthodes pour étudier le cas des états liés d'un puits de potentiel fini. Puis on verra l'équation dépendante du temps pour décrire la diffusion d'une particule à une dimension.

3.1 Méthode analytique

L'équation stationnaire de Schrödinger pour une particule de masse m dans un potentiel $V(x)$ à une dimension s'écrit

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

On va considérer le cas d'un puits de potentiel fini. Ce potentiel est donné par

$$V(x) = \begin{cases} -V_0 & |x| \leq a \\ 0 & |x| > a \end{cases},$$

où a est la moitié de la largeur du puits et V_0 est sa profondeur. A présent on va suivre le traitement du problème tel que donné dans le livre de Griffiths[4]. Pour résoudre l'équation, on partage l'espace en trois régions. Dans la première région, $x < -a$, l'équation devient

$$\frac{-\hbar^2}{2m}\frac{d^2}{dx^2}\psi(x) = E\psi(x), \qquad \text{ou} \quad \frac{d^2}{dx^2}\psi(x) = \kappa^2\psi(x),$$

avec

$$\kappa \equiv \frac{\sqrt{-2mE}}{\hbar}$$

qui est réel et positif puisque E est négatif pour les états liés. La solution générale est

$$\psi(x) = A\,e^{-\kappa x} + B\,e^{\kappa x}.$$

On voit que le premier terme devient infini quand $x \to -\infty$, et pour un état lié ceci n'est pas acceptable. La solution qui est physiquement admissible est alors

$$\psi(x) = B\,e^{\kappa x}, \qquad \text{pour } x < -a.$$

Dans la deuxième région, $-a \leq x \leq a$, l'équation de Schrödinger s'écrit

$$\frac{-\hbar^2}{2m}\frac{d^2}{dx^2}\psi(x) - V_0\psi(x) = E\psi(x), \qquad \text{ou} \quad \frac{d^2}{dx^2}\psi(x) = -l^2\psi(x),$$

avec

$$l \equiv \frac{\sqrt{2m(E+V_0)}}{\hbar},$$

qui est lui aussi réel et positif, puisque E doit être supérieur à $-V_0$. La solution générale est maintenant donnée par

$$\psi(x) = C\sin(l\,x) + D\cos(l\,x), \qquad \text{pour } -a \leq x \leq a.$$

Enfin et dans la troisième région, $x > a$, le potentiel est nul, et comme dans la première région, la solution générale s'écrit

$$\psi(x) = F\,e^{-\kappa x} + G\,e^{\kappa x}.$$

Mais la solution physiquement acceptable est cette fois-ci

$$\psi(x) = F\,e^{-\kappa x}, \qquad \text{pour } x > a.$$

Les constantes d'intégration B, C, D, et F vont être déterminées à partir des conditions aux limites. Au fait, il restera toujours une constante arbitraire, qui pourra être fixée si par exemple on veut que la fonction d'onde soit normée. Les conditions aux limites a imposer à la fonction d'onde dans ce cas sont : ψ et $d\psi/dx$ doivent être continues à $x = -a$ et à $x = a$. Le potentiel V est pair, et par conséquent on peut supposer que les fonctions d'onde sont soit paires, soit impaires. Ainsi on n'imposera de conditions aux limites que d'un côté, par exemple au point $x = a$. On commence par le cas d'une fonction paire. Elle à la forme suivante

$$\begin{cases} F\,e^{-\kappa x}, & \text{pour } x > a, \\ D\cos(l\,x), & \text{pour } 0 \leq x \leq a, \\ \psi(-x), & \text{pour } x < 0. \end{cases}$$

La continuité de $\psi(x)$ au point $x = a$, implique

$$F\,e^{-\kappa a} = D\cos(l\,a), \tag{3.1}$$

et la continuité de $d\psi/dx$ donne

$$-\kappa F\,e^{-\kappa a} = -lD\sin(l\,x). \tag{3.2}$$

En divisant les deux égalités, on trouve

$$\kappa = l\tan(la).$$

C'est une équation qui donne les énergie permises, puisque κ et l sont toutes deux fonctions de E. On peut obtenir une équation simplifiée en définissant les deux quantités suivantes

$$z \equiv l\,a, \qquad \text{et} \qquad z_0 = \frac{a}{\hbar}\sqrt{2mV_0}.$$

Puisque $(\kappa^2 + l^2) = 2mV_0/\hbar^2$, on a $\kappa\,a = \sqrt{z_0^2 - z^2}$, et l'équation précédente devient

$$\tan(z) = \sqrt{(z_0/z)^2 - 1}. \tag{3.3}$$

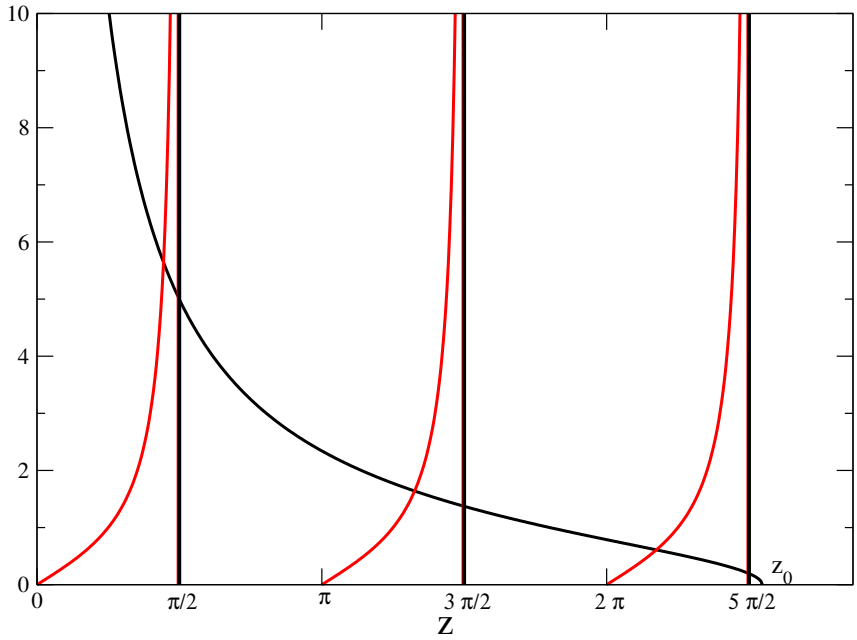

FIGURE 3.1 – *Solution graphique de l'équation 3.3 pour* $z_0 = 8$. *En rouge* $\tan(z)$ *et en noir* $\sqrt{(z_0/z)^2 - 1}$.

C'est une équation transcendante pour z, et par conséquent pour E, en fonction de z_0, qui est une mesure de la taille du puits (sa largeur et sa profondeur).

Pour le cas d'une fonction impaire, on peut refaire la même démarche et trouver l'équation transcendante suivante

$$\cot(z) = -\sqrt{(z_0/z)^2 - 1}. \tag{3.4}$$

Les deux équations transcendantes doivent être résolues numériquement pour obtenir les valeurs des énergies permises. Mais on peut se faire une idée sur les solutions possibles en traçant les courbes des deux cotés de l'égalité sur un même graphe. C'est ce qu'on voit dans la figure 3.1 pour le cas de la première équation. On peut voir qu'à mesure que z_0 augmente, le nombre d'intersections des deux courbes, et donc le nombre d'états liés paires augmente lui aussi. On voit aussi que quelque soit la valeur de z_0, il existe toujours au moins une intersection, et donc un état lié pair. Ceci n'est pas le cas par contre pour les états impaires.

Maintenant on va résoudre numériquement les équations transcendantes en choisissant des valeurs particulières pour les paramètres utilisés. On prend le cas d'une particule de masse égale à celle d'un électron m_e. La largeur du potentiel est de $a = 2\,a_0$, où a_0 est le rayon de Bohr. La profondeur du potentiel est de $V_0 = 300$ eV.

```
clear all
physconsts
V0=300;
z0_sqr=(2*me*a0^2/hbar^2)*V0*ee;
z0=sqrt(z0_sqr);
z=linspace(0,z0,1000);
%figure
subplot(2,1,1);
plot(z,tan(z),'r','LineWidth',3,z,sqrt((z0./z).^2-1),'LineWidth',3,'k')
axis([0 5 -5 5])
```

```
title('Solutions paires')
grid
subplot(2,1,2);
plot(z,cot(z),'r','LineWidth',3,z,-sqrt((z0./z).^2-1),'LineWidth',3,'k')
axis([0 5 -5 5])
title('Solutions impaires')
grid

%Premiere solution paire
zsol=bissec(1.0,1.5,100,1.0e-6,@(z) tan(z)-sqrt((z0./z).^2-1));
E0=hbar^2*zsol^2/(2*me*a0^2)/ee-V0;
fprintf('E0 = %g eV \n',E0);

%Premiere solution impaire
zsol=bissec(2.5,3.0,100,1.0e-6,@(z) cot(z)+sqrt((z0./z).^2-1));
E1=hbar^2*zsol^2/(2*me*a0^2)/ee-V0;
fprintf('E1 = %g eV \n',E1);

%Deuxiemme solution paire
zsol=bissec(3.5,4.0,100,1.0e-6,@(z) tan(z)-sqrt((z0./z).^2-1));
E2=hbar^2*zsol^2/(2*me*a0^2)/ee-V0;
fprintf('E2 = %g eV \n',E2);
```

Le script `physconsts` dans la deuxième ligne contient les constantes physiques nécessaires. Son contenu est le suivant

```
% Bohr radius in m
a0=5.2917721067e-11;

% Electron charge in Coulomb
ee=1.602176565e-19;

% Planck's reduced constant in J s
hbar=1.054571726e-34;

% Electron mass in Kg
me=9.10938291e-31;
```

On obtient les graphes montrés dans la figure 3.2. On voit sur la figure qu'il y a deux solutions paires et une solution impaires. A partir de la figure on peut aussi lire les intervalles dans lesquelles se trouvent les intersections recherchées. A partir de là, on utilise la méthode de la bissection pour trouver les valeurs exactes des abscisses z correspondants aux intersections. En exécutant le programme, on obtient les énergies permises :

```
E0 = -277.287 eV
E1 = -210.556 eV
E2 = -105.844 eV
```

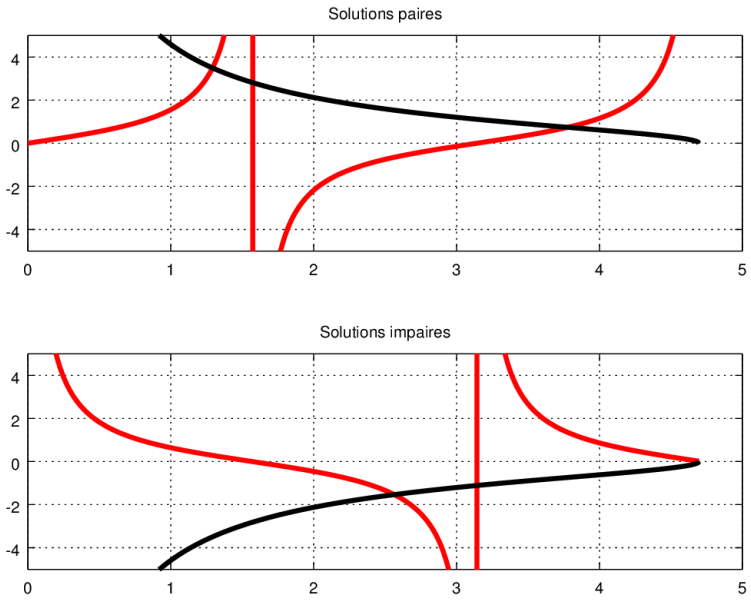

FIGURE 3.2 – *Solutions graphiques pour* $m = m_e$, $a = 2a_0$ *et* $V_0 = 300$ *eV.*

3.2 Méthode spectrale

Dans cet exercice on va explorer l'idée d'utiliser une base complète de fonctions d'onde pour résoudre l'équation de Schrödinger. Cette méthode est la base de la plupart des codes de calcul de la structure électronique en physique du solide. On cite deux exemples en particulier : le code Abinit [1] utilise des ondes planes comme base, et le code Siesta [2] utilise des orbitales atomiques. Pour revenir au présent problème, on va mettre le puits de potentiel fini à l'intérieur d'un autre puits de potentiel infini comme montré sur la figure 3.3. Le potentiel est alors donné par

$$V(x) = \begin{cases} \infty & x \leq 0 \,\text{et}\, x \geq L, \\ 0 & 0 < x \leq (L-a)/2 \;\text{ et }\; (L+a)/2 \leq x < L, \\ -V_0 & (L-a)/2 < x < (L+a)/2. \end{cases}$$

On va alors utiliser les fonctions d'onde solutions du puits de potentiel infini comme base pour étudier le puits de potentiel fini. En effet les fonctions propres du puits de potentiel infini sont donnés par

$$\varphi_n(x) = \sqrt{\frac{2}{L}} \sin\left(\frac{n\pi}{L}x\right),$$

avec l'énergie propre

$$E_n = \frac{\hbar^2 n^2 \pi^2}{2ma^2},$$

et l'indice $n = 1, 2, 3 \ldots \infty$. Les éléments de matrice de l'hamiltonien $H_{mn} = \langle \varphi_m | H | \varphi_n \rangle$ dans cette base sont donnés par

$$H_{mn} = T_{mn} + V_{mn},$$

1. Voir le site http://www.abinit.org/.
2. Voir le site http://departments.icmab.es/leem/siesta/.

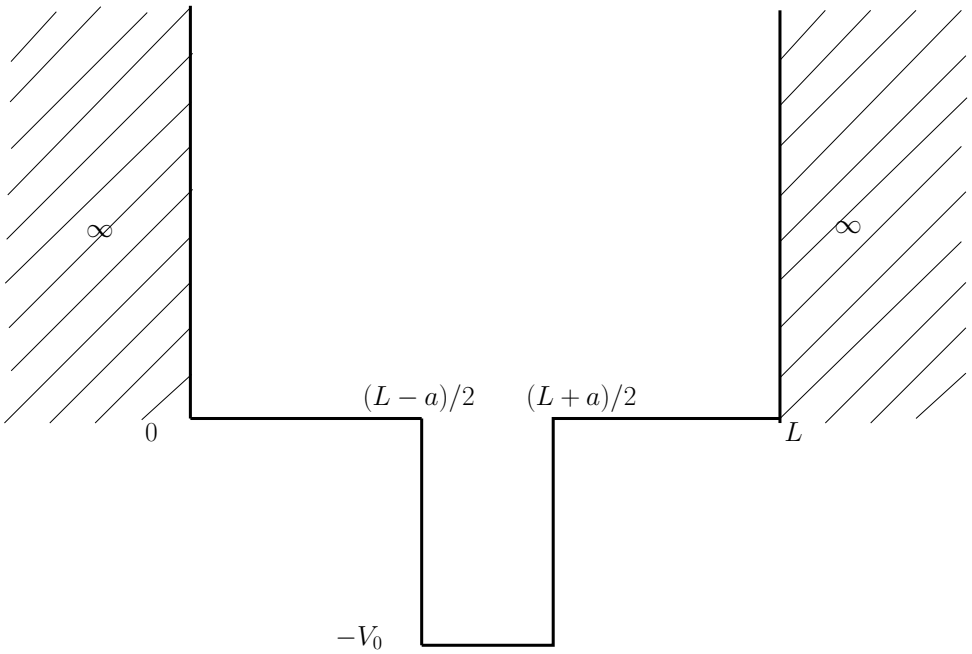

FIGURE 3.3 – *Puits de potentiel fini à l'intérieur d'un puits infini.*

où le premier terme vient de l'énergie cinétique et le deuxième vient de l'énergie potentielle. Le premier est diagonal et on a

$$T_{mn} = E_n \delta_{mn},$$

et le deuxième donne

$$V_{mn} = -\frac{2V_0}{L} \int_{(L-a)/2}^{(L+a)/2} \sin\left(\frac{m\pi}{L}x\right) \sin\left(\frac{n\pi}{L}x\right).$$

Si $m = n$(élément diagonal), on a

$$V_{mn} = V_{mm} = \frac{-V_0}{L}\left\{a - \frac{L}{2m\pi}\left[\sin\left(\frac{m\pi(L+a)}{L}\right) - \sin\left(\frac{m\pi(L-a)}{L}\right)\right]\right\},$$

et si $m \neq n$, on obtient

$$\begin{aligned} V_{mn} \;=\; & \frac{-V_0}{L}\left\{\frac{L}{(m-n)\pi}\left[\sin\left(\frac{(m-n)\pi(L+a)}{2L}\right) - \sin\left(\frac{(m-n)\pi(L-a)}{2L}\right)\right] - \right. \\ & \left. \frac{L}{(m+n)\pi}\left[\sin\left(\frac{(m+n)\pi(L+a)}{2L}\right) - \sin\left(\frac{(m+n)\pi(L-a)}{2L}\right)\right]\right\}. \end{aligned}$$

On peut écrire maintenant un programme pour utiliser la méthode qu'on vient d'énoncer. Pour ce faire on va utiliser le rayon de Bohr a_0 comme unité de mesure des distances. On va prendre la masse de la particule comme étant celle de l'électron m_e. Pour les énergies on va utiliser l'électron-volt. Et donc on doit introduire une constante qu'on va appeler `scale` dans le programme telle que

$$scale = \frac{\hbar^2}{2m_e {a_0}^2 e} \approx 13.60569\,\text{eV},$$

où e est la charge de l'électron. Avec ces unités on prend la largeur du puits infini comme $L = 11$, la largeur du puits fini comme $a = 2$ et sa profondeur comme $V_0 = 300$. La dimension de la matrice

hamiltonienne ou bien le nombre de fonctions de base qu'on va utiliser est de $N_{dim} = 200$. Une fois qu'on a construit la matrice hamiltonienne, on fera appel à la fonction `eig` pour la diagonaliser. L'instruction

```
[V D] = eig(Hmat);
```

permet de diagonaliser la matrice `Hmat`. Les valeurs propres sont alors sauvegardées dans les éléments diagonaux de la matrice `D`, et les vecteurs propres sont écrits dans la matrice `V`. Les valeurs propres sont réelles et elle sont ordonnées de manière ascendante. Les vecteurs propres sont tels que la i^{eme} colonne de `V` contient le vecteur propre associé à la i^{eme} valeur propre contenue dans `D`.

```
clear all
global ener
L=11;    % in units of a0, the Bohr radius
aa=2;    % in units of a0, the Bohr radius
V0=300;  % in eV
ndim=200;
K=zeros(ndim);
V=zeros(ndim);
scale=13.6056925383822; % hbar^2/(2*me*a0^2)/ee  in eV

for m=1:ndim
  K(m,m) = scale*pi^2*m^2/L^2;
end
for m=1:ndim
 for n=1:ndim
   if n==m
       V(m,n)=(-V0/L)*(aa-(L/(m+n)/pi)*(sin(((m+n)*pi/L)*(L+aa)/2) - ...
              sin(((m+n)*pi/L)*(L-aa)/2)));
   else
       V(m,n) = (-V0/L)*((L/(m-n)/pi)*(sin(((m-n)*pi/L)*(L+aa)/2) - ...
                 sin(((m-n)*pi/L)*(L-aa)/2)) + ...
                (-L/(m+n)/pi)*(sin(((m+n)*pi/L)*(L+aa)/2) - ...
                 sin(((m+n)*pi/L)*(L-aa)/2)));
   end
  end
end

Hmat = K + V;
[V D] = eig(Hmat);
E0=D(1,1);
fprintf('E0 = %g eV\n',E0);
Np=200;
x=linspace(0,L,Np);
psi_l=zeros(1,Np);
for n=1:Np
   psi_l(n) = mypsi(x(n),V(:,1),L);
end
if psi_l(Np/2)<0
```

```
  psi_l = -psi_l;
end
subplot(3,1,1);
plot(x,psi_l,'r','LineWidth',3)
axis([0 11 -1 1])
grid

E1=D(2,2);
fprintf('E1 = %g eV\n',E1);
psi_l=zeros(1,Np);
for n=1:Np
  psi_l(n) = mypsi(x(n),V(:,2),L);
end
if psi_l(Np/2)<0
  psi_l = -psi_l;
end
subplot(3,1,2);
plot(x,psi_l,'r','LineWidth',3)
axis([0 11 -1 1])
grid

E2=D(3,3);
fprintf('E2 = %g eV\n',E2);
psi_l=zeros(1,Np);
for n=1:Np
   psi_l(n) = mypsi(x(n),V(:,3),L);
end
if psi_l(Np/2)<0
   psi_l = -psi_l;
end
subplot(3,1,3);
plot(x,psi_l,'r','LineWidth',3)
axis([0 11 -1 1])
grid
```

On a besoin de la fonction `mypsi` qui calcule la fonction d'onde à partir des vecteurs propres. Voici la fonction

```
function y = mypsi(t,vec,L)
   y=0;
   for n=1:length(vec)
       y = y + sqrt(2/L)*sin(n*pi*t/L)*vec(n);
   end
```

Après exécution du programme on obtient pour les énergies permises :

```
E0 = -277.286 eV
E1 = -210.555 eV
E2 = -105.84 eV
```

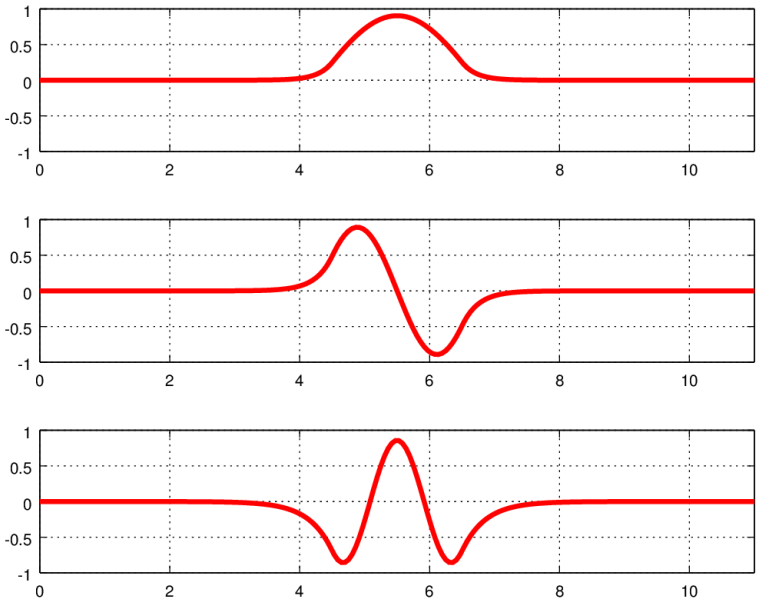

FIGURE 3.4 – *Les fonctions d'onde des états liés d'un puits de potentiel fini obtenues par la méthode spectrale. L'énergie augmente de haut en bas.*

Ces valeurs sont à comparer avec celles obtenues en utilisant la solution analytique. L'erreur absolue est de l'ordre de 0.004 eV. Les trois fonctions d'onde des états liés obtenues par la méthode spectrale sont montrées sur la figure 3.4

3.3 Différences finies

Dans ce qui suit nous allons étudier le problème présenté dans la section précédente en utilisant une méthode numérique simple. En effet, étant donné que l'équation de Schrödinger ne contient comme dérivée que la dérivée seconde, on va approximer celle-ci par des différences finies centrées. L'idée est de subdiviser l'intervalle d'intégration en petits sous-intervalles de largeur τ. En écrivant le développement de Taylor à l'ordre 2 pour la fonction ψ au voisinage d'un point x_i, on a

$$\psi(x_i \pm \tau) = \psi(x_i) \pm \tau\psi'(x_i) + \frac{\tau^2}{2}\psi''(x_i) + O(\tau^3).$$

On a alors

$$\left.\frac{d^2\psi}{dx^2}\right|_{x_i} = \psi''(x_i) \approx \frac{\psi(x_i+\tau) - 2\psi(x_i) + \psi(x_i-\tau)}{\tau^2}.$$

En remplaçant dans l'équation de Schrödinger, on obtient

$$-\frac{\hbar^2}{2m}\frac{\psi(x_i+\tau) - 2\psi(x_i) + \psi(x_i-\tau)}{\tau^2} + V(x_i)\psi(x_i) = E\psi(x_i).$$

On peut alors exprimer $\psi(x_i+\tau)$ comme

$$\psi(x_i+\tau) = \frac{2m}{\hbar^2}\tau^2(V(x_i) - E)\psi(x_i) + 2\psi(x_i) - \psi(x_i-\tau).$$

Ayant la valeur de ψ aux points x_i et $x_i - \tau$, on peut calculer sa valeur au point $x_i + \tau$. Cet algorithme va nous permettre d'intégrer numériquement l'équation de Schrödinger, pourvu que nous fixions les deux premières valeurs de ψ, aux points x_1 et $x_2 = x_1 + \tau$.

Pour revenir aux cas qui nous intéresse, on sait que le potentiel est infini pour $x < 0$ et $x > L$. En prenant $x_1 = 0$, on a alors $\psi(x_1) = 0$, étant donné que ψ est nulle pour $x < 0$ et qu'elle doit être continue en $x = 0$. Si on prend $x_N = L$, on a aussi $\psi(x_N) = 0$. Cette deuxième condition n'est pas vraiment utile pour notre algorithme, puisqu'on veut savoir la valeur de $\psi(x_2)$. Cependant c'est en imposant à la fonction ψ de s'annuler au point $x = L$ qu'on va retrouver les énergies permises. Quant à la valeur de $\psi(x_2)$, on remarque que l'équation de Schrödinger étant linéaire, si une fonction f est une solution de l'équation, alors αf, où α est un nombre complexe, est aussi solution. Ceci nous permet donc de donner n'importe quelle valeur non nulle à $\psi(x_2)$. La recherche des énergies permises consiste alors à donner une valeur à E et à intégrer numériquement l'équation, jusqu'à $x_N = L$. Si la valeur $\psi(x_N)$ n'est pas nulle, alors E ne correspond pas à une énergie permise. On refait alors la procédure avec une autre valeur de E. En général, on devrait diviser l'intervalle $[-V_0\ 0]$ en un grand nombre de sous-intervalles et on inspecte le changement de signe de $\psi(x_N)$ dans chacun des sous-intervalles. Si on un trouve un changement de signe dans un sous-intervalle donnée, alors on utilise une méthode plus efficace pour trouver la valeur de E qui donne $\psi(x_N) = 0$. Dans le cas présent, on va utiliser la méthode de la bissection pour effectuer cette tâche. Et comme on connaît déjà les valeurs des énergies permises à partir des deux sections précédentes, on va fixer les intervalles de recherche "à la main". Comme dans la section précédente, on voudrait exprimer les distances en unités du rayon de Bohr a_0 et les énergies en électron-volt. Pour ce faire, on va définir une variable `scale` qui aura pour valeur

$$scale = \frac{2m_e {a_0}^2 e}{\hbar^2} \approx 0.073499.$$

Le programme principal est comme suit :[3]

```
clear all
global ener
Np=10000;
global L;
L=11;
global x
x=linspace(0,L,Np);
global psi_l
psi_l=zeros(1,Np);
psi_l(1)=0;
psi_l(2)=psi_l(1)+(x(2)-x(1))^4;
global scale
scale=0.0734986475094125 % distances in Bohr radius and energies in eV

[E0 niter]= bissec(-277,-277.5,800,1e-4,@integrate);
fprintf('E0 = %g eV\n',E0);
subplot(3,1,1);
plot(x,psi_l,'r','LineWidth',3)
```

3. A noter que l'utilisation des variables globales comme c'est fait ici est fortement déconseillée. On se le permet ici uniquement par soucis de clarté, et parce que les scripts sont simples, ce qui limite les erreurs eventuelles due à l'utilisation de ce type de variables.

```
axis([0 11 -0.1 0.4]) grid

[E1 niter]= bissec(-210,-211.5,800,1e-5,@integrate);
fprintf('E1 = %g eV\n',E1);
subplot(3,1,2);
plot(x,psi_l,'r','LineWidth',3)
axis([0 11 -0.02 0.02])
grid

[E2 niter]= bissec(-105,-106,800,1e-5,@integrate);
fprintf('E2 = %g eV\n',E2);
subplot(3,1,3);
plot(x,psi_l,'r','LineWidth',3)
axis([0 11 -0.0001 0.0001])
grid
```

La fonction qui effectue l'intégration numérique, appelée `integrate`, est la suivante

```
function y= integrate(t)
 global x
 global psi_l
 global ener
 global scale
 ener=t;
 for i=3:length(x)
   psi_l(i) = scale*(pot(x(i))-ener)*psi_l(i-1)*(x(i)-x(i-1))^2 + ...
                2 * psi_l(i-1) - psi_l(i-2);
 end
 y = psi_l(end);
```

Quant au potentiel, il est donné par

```
function ret = pot(t)
 global L;
 V0=300;
 a=2;
 if length(t)>1
    ret = arrayfun(@pot,t);
 elseif (t > (L/2-a/2)) && (t<(L/2+a/2))
    ret = -V0;
 else
    ret = 0;
 end
```

L'exécution du programme donne les énergies permises :

```
E0 = -277.287 eV
E1 = -210.557 eV
E2 = -105.844 eV
```

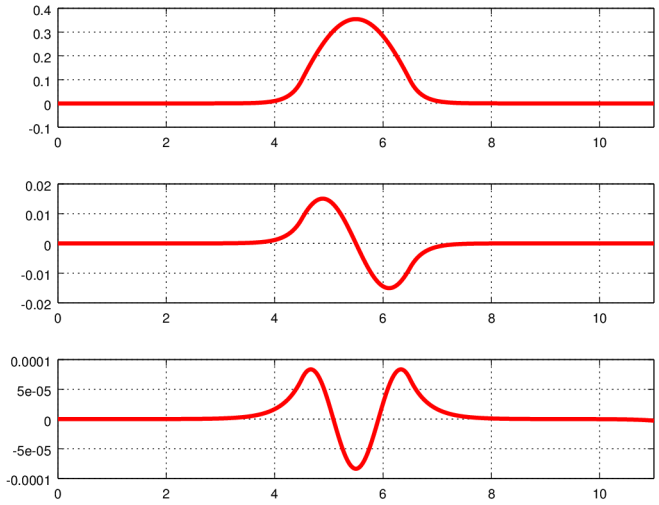

FIGURE 3.5 – *Les fonctions d'onde des états liés d'un puits de potentiel fini obtenues par la méthode des différences finies. L'énergie augmente de haut en bas.*

Ces valeurs sont à comparer avec celles obtenues en utilisant la solution analytique. L'erreur absolue est de l'ordre de 0.001 eV. Les trois fonctions d'onde des états liés obtenues par la méthode des différences finies sont montrées sur la figure 3.5.

3.4 Equation de Schrödinger dépendante du temps

Jusque là on a vu différentes méthodes de résolution de l'équation de Schrödinger stationnaire. Dans ce qui suit on s'intéressera à l'équation dépendante du temps, et plus particulièrement à la propagation d'un paquet d'onde à une dimension, et les phénomènes qui en découlent, tels que la réflexion, la transmission et l'effet tunnel. L'équation de Schrödinger dépendante du temps à une dimension s'écrit

$$\frac{-\hbar^2}{2m}\frac{\partial^2\psi}{\partial x^2} + V(x)\psi = i\hbar\frac{\partial\psi}{\partial t},$$

où ψ est évidemment fonction de x et de t. De plus ψ n'est plus réelle, comme dans le cas stationnaire, mais complexe, à cause de la présence de $i = \sqrt{-1}$ dans le second membre de l'équation. A présent on va suivre le traitement du problème tel que donné dans le livre de Giordano et Nakanishi[5]. Pour utiliser la méthode des différences finies, vue auparavant, on va décomposer la fonction d'onde en ses parties réelle et imaginaire :

$$\psi(x,t) = R(x,t) + i\,I(x,t).$$

En prenant $\hbar$ et m comme unités, l'équation se décompose en

$$\frac{\partial R}{\partial t} = \left(-\frac{1}{2}\frac{\partial^2}{\partial x^2} + V(x)\right) I$$
$$\frac{\partial I}{\partial t} = \left(\frac{1}{2}\frac{\partial^2}{\partial x^2} - V(x)\right) R.$$

On utilise maintenant les différences finie centrées pour approximer les dérivées partielles. On a

$$\frac{\partial^2 R(x,t)}{\partial x^2} \approx \frac{R(x+\Delta x,t) - 2R(x,t) + R(x-\Delta x,t)}{(\Delta x)^2},$$

pour la dérivée spatiale, et

$$\frac{\partial I(x,t)}{\partial t} \approx \frac{I(x,t+\Delta t/2) - I(x,t-\Delta t/2)}{\Delta t},$$

pour la partie temporelle. En insérant dans les équations précédentes, on obtient

$$\begin{aligned} I(x,t+\Delta t/2) \approx & I(x,t-\Delta t/2) + \frac{\Delta t}{2(\Delta x)^2}\left[R(x+\Delta x,t) - 2R(x,t) + R(x-\Delta x,t)\right] \\ & - \Delta t\, V(x) R(x,t), \end{aligned}$$

et

$$\begin{aligned} R(x,t+\Delta t) \approx & R(x,t) - \frac{\Delta t}{2(\Delta x)^2}\left[I(x+\Delta x,t+\Delta t/2) - 2I(x,t+\Delta t/2) + I(x-\Delta x,t+\Delta t/2)\right] \\ & + \Delta t\, V(x) I(x,t+\Delta t/2). \end{aligned}$$

On discrétise l'espace en prenant $x_m = m\Delta x$ et le temps en prenant $t_n = n\Delta t$. A partir des deux équations précédentes on voit qu'il est naturel d'évaluer $R(x,t)$ aux points $(m\Delta x, n\Delta t)$ et d'évaluer $I(x,t)$ aux points $(m\Delta x, (n\pm\frac{1}{2})\Delta t)$. En plus d'être précise, cette méthode offre l'avantage de conserver la norme de ψ, c'est à dire l'intégral de $\psi^*\psi$ sur tout l'espace, au cours du temps.

Dans ce qui suit on prendra comme condition initiale une gaussienne pour ψ à $t = 0$. Elle a la forme

$$\psi(x, t=0) = C \exp[-(x-x_0)^2/\sigma^2],$$

où x_0 est le centre du paquet, σ sa largeur, et C est un facteur de normalisation. Cette fonction d'onde ne reste pas comme telle au cours du temps. En effet, si on en fait une simulation, on constatera qu'elle s'élargit, tout en restant contrée sur x_0. Mais bien sûr en s'élargissant, le pic diminue afin que la norme soit conservée. La fonction de départ est au fait une somme de plusieurs composantes de Fourier de termes e^{ikx}. Pour obtenir la gaussienne, on y met le même nombre de termes avec un k positif, qu'avec un k négatif. Ces termes correspondent donc à des ondes qui se propagent vers la droite et vers la gauche. La somme est donc une gaussienne qui s'élargit, mais qui garde son centre fixe. Pour décrire une particule qui se déplace dans un sens, on doit donc rajouter quelque chose à la gaussienne de départ. On la multiplie par un terme qui fait que les différentes composantes de Fourier ne sont plus centrées sur $k = 0$, mais sur k_0. On obtient donc

$$\psi(x, t=0) = C \exp[-(x-x_0)^2/\sigma^2] \exp[ik_0 x].$$

Celle-ci décrit maintenant une particule qui se déplace vers la droite avec une vitesse moyenne de $v_0 = \hbar k_0/m$. Pour l'implémentation, quelques remarques/recommandations doivent être prises en compte. Les parties réelle et imaginaires de ψ oscillent comme fonctions de x. Pour bien décrire les oscillations dans les parties réelle et imaginaire de la fonction d'onde, on doit avoir $\Delta x \ll \lambda_0$, où $\lambda_0 = 2\pi/k_0$ est la valeur moyenne de la longueur d'onde. De plus, pour avoir dans l'espace réciproque une valeur moyenne bien définie de k_0 , le choix de σ doit être tel que $\sigma \gg 1/k_0$, puisque la largeur du paquet d'onde dans l'espace des impulsions est de l'ordre de $1/\sigma$. Par ailleurs, la vitesse du paquet d'onde ne doit pas dépasser $\Delta x/\Delta t$, et donc les choix de Δx et Δt doivent être tels que $\Delta x/\Delta t \geq \hbar k_0/m = v_0$.

Le programme principal est le suivant

```
clear all;
Np=1000;
x = linspace(0,1,Np);

% Parametres du paquet d'onde initial;
x0=0.4;
C=10;
sigma_squared=1e-3;
k0=500; % Discretisation
dx=1e-3;
dt=1e-7; %5e-8;

% Paquet d'onde initial;
psi_l=C*exp(-(x-x0).^2/sigma_squared).*exp(1i*k0*x);

% Parties reelle et imaginaire initiales
r_initial=real(psi_l);
i_initial=imag(psi_l);

% Potentiel global V0;
V0=1e5;
v=vpot(x);

r_actuel=r_initial;
i_actuel=i_initial;

for iter = 1:15000;
  r_nouveau = realupdate(Np, r_actuel, i_actuel, dt, dx, v);
  r_actuel=r_nouveau;
  i_nouveau = imagupdate(Np, r_actuel, i_actuel, dt, dx, v);
  prob_dens=r_actuel.^2+i_nouveau.*i_actuel;
  i_actuel=i_nouveau;

% Visualiser le résultat
  if rem(iter, 10)== 0;
    p1=plot(x, prob_dens,'-b');
    hold on
    p2=plot(x,v/1e3,'--r');
    set(p1,'Linewidth',2);
    title('Diffusion par un potentiel');
    if V0>0
      axis([0 1 0 200]);
    else
      axis([0 1 V0/1e3-1 200]);
    end
    xlabel('x');
    ylabel('Densite de probabilite');
```

```
    drawnow;
    hold off
  end
end
```

Les fonctions `realupdate` et `imagupdate` qui font respectivement la mise à jour de la partie réelle et de la partie imaginaire sont comme suit

```
function r_nouveau = realupdate(N, r_act, i_act, dt, dx, v)
 r_nouveau= zeros(1,N);
 s=dt/(2*dx^2);
 for m=2:N-1
   r_nouveau(m) = r_act(m) - s*(i_act(m+1)-2*i_act(m)+i_act(m-1))...
                           +dt*v(m).*i_act(m);
 end;
 % Conditions aux limites
 r_nouveau(1)=r_nouveau(2);
 r_nouveau(N)=r_nouveau(N-1);
```

et

```
function i_nouveau = imagupdate(N, r_act, i_act, dt, dx, v)
 i_nouveau= zeros(1,N);
 s=dt/(2*dx^2);
 for m=2:N-1
   i_nouveau(m) = i_act(m) + s*(r_act(m+1)-2*r_act(m)+r_act(m-1))...
                           -dt*v(m).*r_act(m);
 end;  % Conditions aux limites
 i_nouveau(1)=i_nouveau(2);
 i_nouveau(N)=i_nouveau(N-1);
```

Le potentiel est donné par

```
function ret=vpot(t)
 global V0;
 if length(t)>1
    ret = arrayfun(@vpot,t);
 elseif t > 0.55 %% && t<0.6
   ret = V0;
 else
    ret = 0;
 end
```

Dans la figure 3.6 sont représentés quatre séquences successives d'un film montrant la diffusion d'un paquet d'onde par une barrière de potentiel. On voit dans la dernière séquence, deux paquets d'onde, un réfléchi et un transmet. Si on avait fait cette expérience avec un grand nombre de particules réelles, les deux paquets donneraient alors la proportion de particules réfléchies et celle des particules transmises. La diffusion par un puits de potentiel ainsi que l'effet sont laissés comme exercices pour l'étudiant.

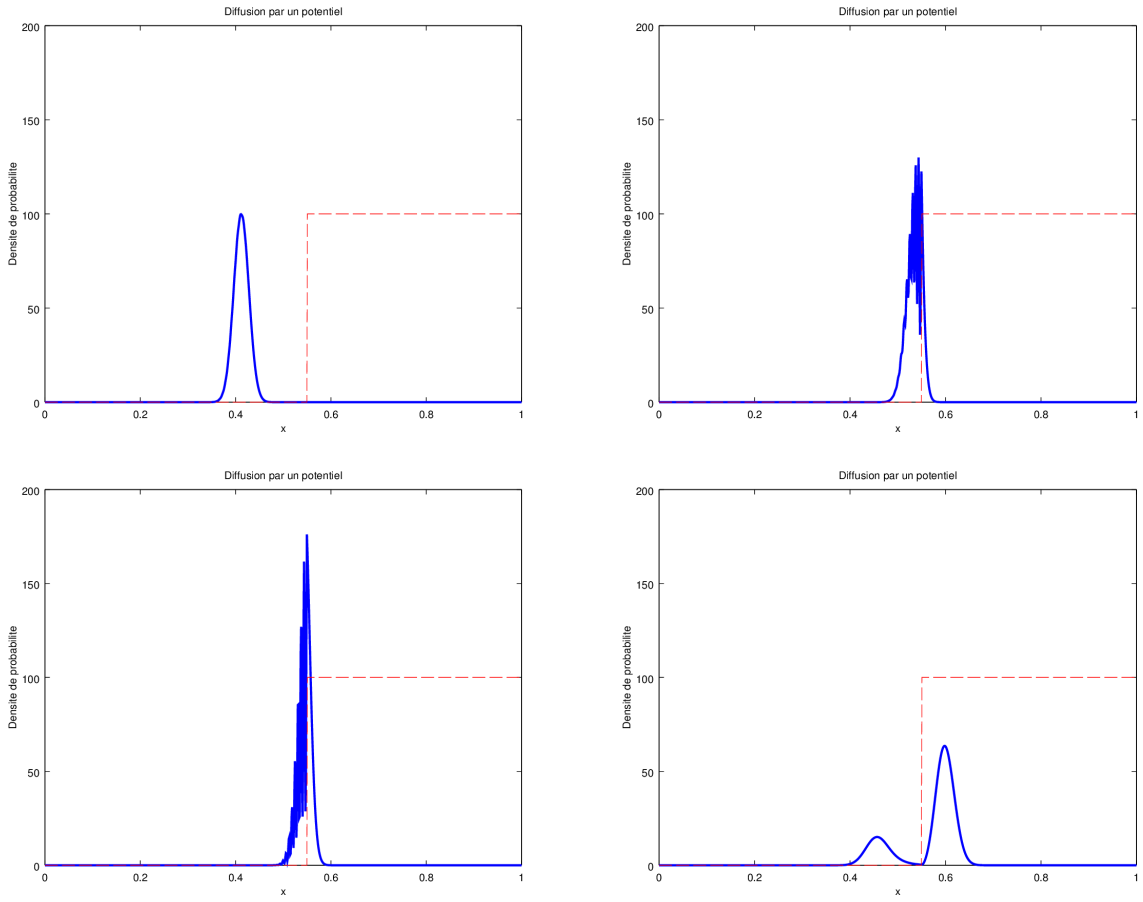

FIGURE 3.6 – *Réflexion par une barrière de potentiel. La progression se fait de gauche à droite et de haut en bas. Les paramètres utilisés sont ceux fixé dans le script principal.*

Chapitre 4

Introduction à Monte Carlo

4.1 Introduction

Dans ce chapitre on va découvrir une méthode de simulation nouvelle : elle est stochastique et donc non-déterministe. La méthode Monte Carlo consiste à utiliser des nombres aléatoires pour résoudre des problèmes liés à l'optimisation, l'intégration numérique, ou à l'échantillonnage d'une distribution de probabilité donnée. En physique elle est très utilisée pour traiter des systèmes ayant un grand nombre de degré de libertés couplés. Dans ce chapitre on va s'intéresser à un de ces problème, qui est celui d'un système de spins magnétiques en interaction : le modèle d'Ising. Pour l'instant et afin d'introduire la méthode de manière la plus simple, on va l'utiliser pour évaluer numériquement la valeur de π.

4.2 Estimation Monte Carlo de la valeur de π

Pour évaluer la valeur de π par une méthode stochastique, on imagine qu'on a tracé sur terre un carré de côté unité (1 mètre par exemple), et qu'on a inscrit un cercle à l'intérieur de sorte que son diamètre soit l'unité. On jette maintenant des pierres de manière aléatoire pour qu'elle tombent dans le carré. A la fin on compte le nombre de pierre tombées à l'intérieur du cercle, soit N_{in}, et le nombre totale de pierre tombées à l'intérieur du carré, soit N_{tot}. Puisque le jet de pierre n'est pas du tout biaisé, on s'attend à ce que les seuls paramètres qui comptent soient l'aire du cercle et celle du carré. Si on prend le rapport des deux nombres, il doit être approximativement égal au rapport des deux aires :

$$\frac{N_{in}}{N_{tot}} \approx \frac{\pi/4}{1},$$

et donc que

$$\pi \approx 4 \times \frac{N_{in}}{N_{tot}}.$$

Maintenant pour réaliser cette expérience sur un ordinateur, on va utiliser un générateur de nombre aléatoires uniformes dans l'intervalle [0 1[. La fonction qui implémente ce générateur s'appelle `rand`. Puisque tous les nombres sont positifs, on va donc prendre un carré et un cercle centrés sur l'origine. Mais au lieu de considérer toute les surfaces des deux, on s'intéresse uniquement à celles dans le premier quadrant. Le résultat final n'est pas modifié puisque les aires d'intérêt sont le quart des aires totales. On génère donc N_{tot} couples (x, y) de nombres aléatoires qui tombent tous dans le carré bien sûr. On considère qu'un point est à l'intérieur du cercle si $x^2 + y^2 < 1$. On appelle N_{in} le nombre

de ces points. A la fin on utilise la même formule pour estimer π. Ceci est un script qui implémente cette méthode

```
clear all
Ntot=1000000;
Nin=0;
for ip=1:Ntot
  x=rand();
  y=rand();
  if (x^2+y^2) <1
     Nin = Nin+1;
  end
end
Pi_est = 4*Nin/Ntot;
fprintf('Pi_est = %g  erreur absolue = %g\n',Pi_est,abs(Pi_est-pi));
```

L'exécution du script donne

```
> pi_mc
Pi_est = 3.14207  erreur absolue = 0.000479346
> pi_mc
Pi_est = 3.13961  erreur absolue = 0.00198465
```

On voit que le résultat retourné n'est pas le même dans les deux exécutions du script. Ceci est normal, puisque la méthode n'est pas déterministe. Les couples (x, y) générés dans la première exécution ne sont nécessairement pas les mêmes que ceux générés dans la deuxième.

4.3 Intégration Monte Carlo à une dimension

Pour estimer une intégrale définie d'une fonction $f(x)$ sur l'intervalle $[a\ b]$ avec la méthode Monte Carlo, imaginons qu'on génère un seul point x_1 aléatoirement dans cet intervalle. L'estimation qu'on puisse faire de l'intégrale est alors l'aire du rectangle de base $b-a$ et de hauteur $f(x_1)$. Ce qui donne

$$I = \int_a^b f(x)\,dx \approx (b-a)f(x_1).$$

Si on génère deux points, x_1 et x_2, aléatoirement, alors il serait juste d'approximer l'intégrale par la somme des aires de deux rectangles. Pour ne par favoriser un point sur l'autre, on va associer à chacun d'eux la même base pour les rectangles respectifs : $(b-a)/2$. Quant aux hauteurs, elles auront $f(x_1)$ et $f(x_2)$. L'estimation de l'intégrale est alors

$$I = \int_a^b f(x)\,dx \approx \frac{b-a}{2}\left[f(x_1) + f(x_2)\right].$$

Il est facile de voir que plus on génère de points, mieux est l'approximation. On peut généraliser alors la formule précédente pour le cas de N points uniformément distribués dans $[a\ b]$[3] :

$$I = \int_a^b f(x)\,dx \approx \frac{b-a}{N}\sum_{i=1}^{N} f(x_i).$$

Voici une fonction qui implémente cet algorithme

```
function res=mcintegrate(a,b,np,f)
 res=0;
 for ip=1:np
   x=rand();
   res = res + f((b-a)*x+a);
 end
 res = (b-a) * res / np;
```

Les arguments de la fonctions sont les suivants : `a` et `b` sont les bornes d'intégration, `np` est le nombre de points générés, et `f` est la fonction à intégrer. Pour générer des points aléatoirement on fait appel à la fonction `rand` qui génère une variable aléatoire r uniforme dans l'intervalle [0 1[. Pour avoir une variable aléatoire x_i uniforme dans $[a\ b]$, on fait la transformation $x_i = (b-a) \times r + a$. On calcule par exemple

$$I = \int_0^1 \frac{4}{1+x^2},$$

et on compare à la valeur exacte $I_{ex.} = \pi$:

```
> Ntot=100000;
> err1 = 4 * mcintegrate(0,1,Ntot,@(x) 1/(1+x^2)) - pi
err1 =  0.0015221
> err2 = 4 * mcintegrate(0,1,Ntot,@(x) 1/(1+x^2)) - pi
err2 =  0.0022081
```

On voit bien sûr que deux appels à la même fonction ne donnent pas un résultat identique. C'est le fait que les point x_i générés lors des deux appels ne sont pas nécessairement les mêmes.

4.4 Intégrales doubles

Comme pour le cas d'une intégrale simple où on associe un rectangle à chaque point tiré aléatoirement, dans le cas d'une intégrale double, on associe un parallélépipède à chaque point. Supposons qu'on veuille calculer

$$I = \int_a^b dx \int_c^d dy\, f(x,y).$$

On génère N points (x_i, y_i) aléatoirement dans $[a\ b] \times [c\ d]$. A chaque point, on associe une base de surface $(b-a)\times(d-c)$ et de hauteur $f(x_i, y_i)$. Alors l'intégrale recherchée est la somme des volumes de ces parallélépipèdes[3] :

$$I = \int_a^b dx \int_c^d dy\, f(x,y) \approx \frac{(b-a)(d-c)}{N} \sum_{i=1}^{N} f(x_i, y_i).$$

Une fonction qui implémente cet algorithme est

```
function res=mcintegrate2D(a,b,c,d,np,f)
 res=0;
 for ip=1:np
   x=rand();
   y=rand();
   res = res + f((b-a)*x+a,(d-c)*y+c);
```

```
end
res = (b-a) * (d-c) * res / np;
```

Les arguments de la fonctions sont : `a`, `b`, `c` et `d` sont les bornes d'intégration, `np` est le nombre de points générés, et `f` est la fonction à intégrer. Comme pour le cas de l'intégrale à une dimension, il faut opérer une transformation pour ramener les points générés par `rand` dans [0 1[×[0 1[vers $[a\ b[\times[c\ d[$. Comme exemple, on calcule

$$I = \int_0^1 \int_0^1 \frac{dx\,dy}{(1+x^2)(1+y^2)},$$

et on compare à la valeur exacte $I_{ex.} = (\pi/4)^2$:

```
> Ntot=100000;
> err = mcintegrate2D(0,1,0,1,Ntot,@(x,y) (1/(1+x^2))*(1/(1+y^2))) ...
                                - (pi/4)^2
err =     2.1095e-04
> err = mcintegrate2D(0,1,0,1,Ntot,@(x,y) (1/(1+x^2))*(1/(1+y^2))) ...
                                - (pi/4)^2
err =     7.9996e-04
```

4.5 Modèle d'Ising ferromagnétique à deux dimensions

4.5.1 Introduction

Introduit en 1925 par Lenz pour son étudiant Ising, le modèle qui porte le nom d'Ising décrit un système de moments magnétiques localisés sur des sites d'un réseau périodique. Ces moments interagissent entre eux par une interaction d'échange de courte portée. De plus, chaque moment est de spin 1/2 et à cause d'une forte anisotropie, ne peut s'orienter que le long de l'axe de forte anisotropie. Le hamiltonien du système est donné par

$$\mathcal{H} = -J \sum_{<ij>} \sigma_i \sigma_j - h \sum_i \sigma_i,$$

où J est la constante d'échange, et la notation $< ij >$ veut dire que la somme se fait uniquement sur les spins proches voisins. Le deuxième terme représente l'interaction des spins avec un champ magnétique extérieur h, qu'on appelle communément terme de Zeeman. Dans la suite σ_i peut avoir deux valeurs possibles : $\sigma_i = \pm 1$. Il est à noter que J peut être positif ou négatif. Dans le cas $J > 0$, les spin ont tendance à s'orienter parallèlement les uns aux autres, ce qui donne un état fondamental ferromagnétique. Dans le cas où $J < 0$, il résulte un état fondamental anti-ferromagnétique, puisque chaque spin a tendance à s'orienter anti-parallèlement à ses voisins. Mais l'état fondamental anti-ferromagnétique n'est possible que si le réseau est bipartite.

Dans la suite on s'intéressera au cas ferromagnétique, c'est à dire $J > 0$, et en l'absence d'un champ magnétique extérieur, c'est à dire $h = 0$.

4.5.2 Approximation du champ moyen

Le modèle d'Ising peut être traité en première approximation par la méthode du champ moyen introduite par Weiss en 1905 et qu'il a appelée théorie du champ moléculaire. Dans cette approximation on remplace les spins σ_j voisins du spin σ_i par leurs valeurs moyennes thermiques $\langle \sigma_j \rangle$. De plus,

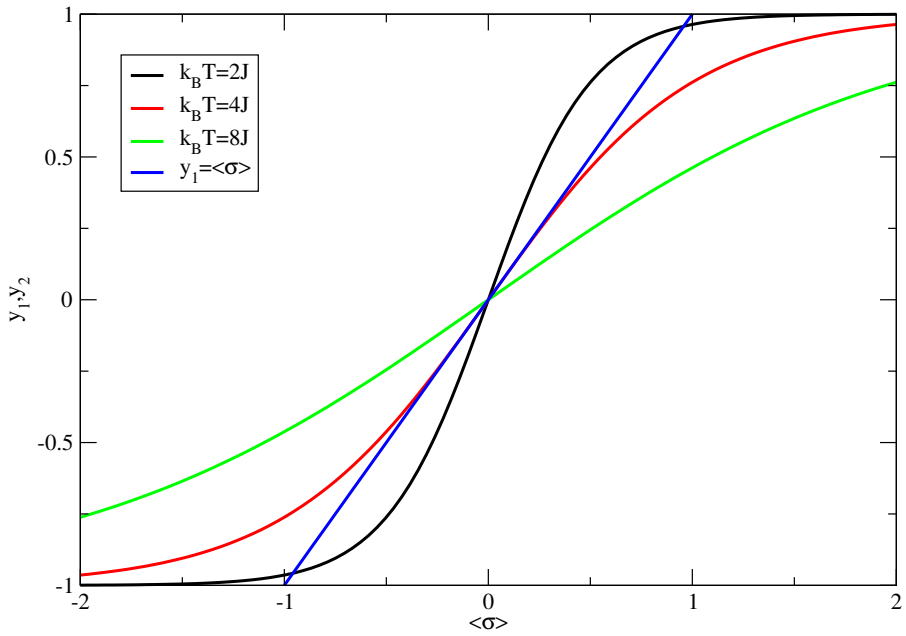

FIGURE 4.1 – *Tracés des courbes $y_1$et y_2 (voir le texte) en fonction de $\langle\sigma\rangle$ pour différentes valeurs du paramètre T et pour $z = 4$.*

on considère que le système est homogène, ce qui donne une valeur moyenne thermique identique pour tous les sites : $\langle\sigma_j\rangle = \langle\sigma\rangle$. Le hamiltonien devient alors

$$\mathcal{H}_{cm} = -zJ\,\langle\sigma\rangle \sum_i \sigma_i,$$

où z est le nombre de premiers voisins : $z = 4$ pour un réseau carré. On voit maintenant que le hamiltonien en champ moyen ressemble au terme de Zeeman. C'est à dire qu'on a affaire à un système de spins indépendants les uns des autres, mais interagissant avec un champ extérieur. La différence avec le terme de Zeeman est que le champ extérieur maintenant dépend de la valeur moyenne $\langle\sigma\rangle$. Puisque les spins sont indépendants, on prend un deux et on calcule sa valeur moyenne. On obtient

$$\langle\sigma\rangle = \frac{1 \times e^{zJ\langle\sigma\rangle/k_BT} - 1 \times e^{-zJ\langle\sigma\rangle/k_BT}}{e^{zJ\langle\sigma\rangle/k_BT} + e^{-zJ\langle\sigma\rangle/k_BT}},$$

ce qui donne

$$\langle\sigma\rangle = \tanh\left(\frac{zJ\,\langle\sigma\rangle}{k_BT}\right), \tag{4.1}$$

où T est la température, et k_B est la constante de Boltzmann. On trouve donc une équation transcendante qui donne la valeur de $\langle\sigma\rangle$. Cette équation peut être résolue graphiquement pour avoir une idée de $\langle\sigma\rangle\,(T)$ comme fonction de la température. Pour ce faire on cherche l'intersection entre deux courbes représentant les deux côtés de l'égalité : $y_1 = \langle\sigma\rangle$ et $y_2 = \tanh\left(\frac{zJ\langle\sigma\rangle}{k_BT}\right)$. La température est alors un paramètre qui va gouverner la présence et le nombre de ces intersections. La figure 4.1 représente des tracés de ces fonctions pour différentes valeurs de T et pour le cas d'un réseau carré, donnant $z = 4$. On voit que pour $k_BT > 4J$, il y a une seule intersection, qui a lieu au point $\langle\sigma\rangle = 0$. Pour ces températures, l'aimantation est nulle et on dit qu'on est dans la phase paramagnétique. Pour des températures telles que $k_BT < 4J$, il existe trois intersections : une au point $\langle\sigma\rangle = 0$ et deux autres au points $\langle\sigma\rangle = \pm\sigma^*$, où $\sigma^* \neq 0$. Un calcul de l'énergie libre permet de montrer que ce sont les

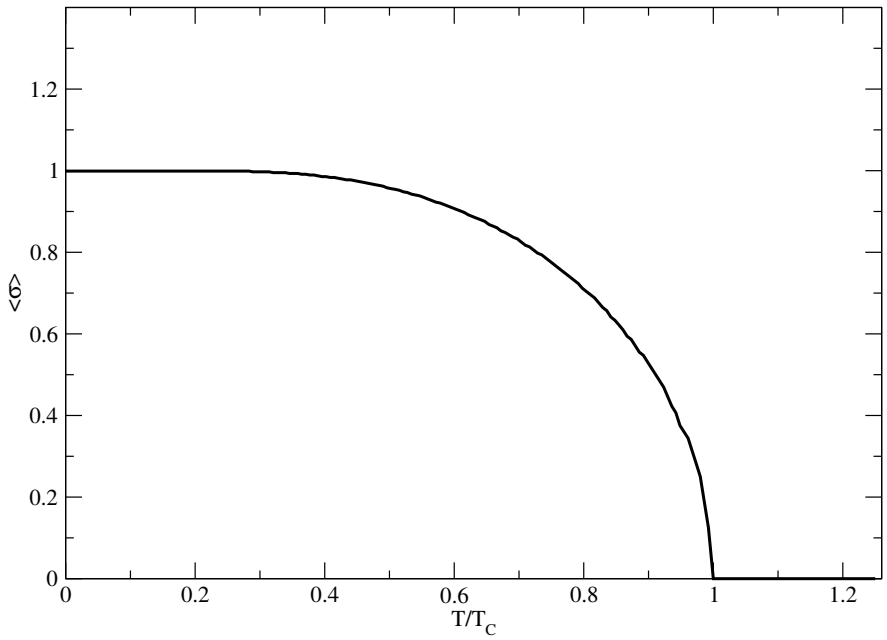

FIGURE 4.2 – *Variations de $\langle\sigma\rangle$ en fonction de la température, obtenues par la résolution numérique de l'équation 4.1.*

deux solutions non-nulles qui sont stables. Donc le système acquiert une aimantation non-nulle, et on dit qu'on est dans la phase ferromagnétique. On a ainsi trouvé que le système subit une transition de phase à la température $T_C = 4J/k_B$. Il passe d'une phase paramagnétique aux hautes températures à une phase ferromagnétique aux basses températures. Ceci est vrai pour le cas d'un réseau carré où $z = 4$. Mais la donne ne change pas pour les autres réseaux : on a toujours $T_C = zJ/k_B$ et ce quelque soit le réseau.

Pour avoir une idée précise sur la valeur de $\langle\sigma\rangle$, on doit résoudre l'équation transcendante 4.1 numériquement. Pour ce faire, on utilise par exemple la méthode de la bissection. La figure 4.2 montre $\langle\sigma\rangle$ comme fonction de la température réduite T/T_C obtenue par la résolution numérique de l'équation 4.1. On voit en effet que $\langle\sigma\rangle = 0$ pour $T > T_C$ ce qui correspond à la phase paramagnétique. A T_C la courbe enregistre une croissance et $\langle\sigma\rangle$ devient non nul. De plus la tangente de la courbe à cette endroit diverge. Puis $\langle\sigma\rangle$ augmente de manière monotone à mesure que T diminue. C'est la phase ferromagnétique. Enfin $\langle\sigma\rangle$ atteint sa valeur à saturation ($\langle\sigma\rangle = 1$) aux basses températures.

On vient de voir que la théorie du champ moyen prévoit la présence d'une transition de phase quelque soit la dimensionnalité du système étudié. Or, il est facile de montrer que le modèle d'Ising n'a pas de transition de phase à une dimension. Ceci est donc un défaut particulièrement remarquable de l'approximation du champ moyen. Par contre pour le cas du modèle d'Ising à deux dimensions, une solution analytique exacte existe– elle est due à Onsager[10]– et montre bien qu'on a une transition de phase à température finie. Pour ce qui est du cas tridimensionnel, il n'existe pas de solution analytique, mais des simulations Monte Carlo ont montré là aussi qu'une transition de phase existe bel et bien à température finie. Par ailleurs, il est connu que même lorsque la transition prédite par la théorie du champ moyen existe, la température à laquelle la transition a lieu est en générale surestimée par cette théorie. Mais, il est aussi connu que la théorie du champ moyen devient de plus en plus exacte à mesure que la dimensionnalité du système augmente. En effet, elle est exacte lorsque la dimensionnalité est infinie. C'est à cause des manquement de l'approximation du champ moyen aux basses dimensionnalités, qu'il faut faire appel à d'autres méthodes, plus fiables, pour mener à

bien l'étude du modèle d'Ising qui nous concerne ici. Une de ces méthode est bien entendu Monte Carlo qu'on présente en détail dans ce qui suit.

4.5.3 Simulation Monte Carlo

Comme mentionné dans l'introduction, Monte Carlo réfère à un large éventail de méthodes numériques basées sur l'échantillonnage par le biais de nombres aléatoires[6]. Ces méthodes peuvent être utilisées par exemple pour estimer des intégrales, notamment à de larges dimensions où les autres méthodes de quadrature ne sont plus utilisables. En physique, la première utilisation de la méthode a été pour la description d'un système de sphères dures [7]. Elle a ensuite été adaptée à bien d'autres problèmes : on peut citer l'étude des propriétés d'équilibre de systèmes de spins, et le problème de diffusion (Monte Carlo cinétique).

Pour étudier un systèmes de spins en interaction, il faut calculer des moyennes statistiques d'observables, telles que l'énergie et l'aimantation. Dans la méthode Monte Carlo cela équivaut à générer des configurations aléatoirement dans l'espoir d'échantillonner au mieux l'espace des configurations (ou espace des phases). La méthode la plus simple consiste à affecter de manière aléatoire des orientations à chacun des spins dans le système. Mais cette façon de faire n'est pas satisfaisante du fait qu'on passe beaucoup de temps à échantillonner des régions de l'espace des configurations dont la contribution au moyennes statistiques recherchées est négligeable. Cette méthode porte le nom d'échantillonnage simple. Une amélioration de cette méthode permet d'échantillonner les configurations qui contribuent le plus aux moyennes statistiques. Cette façon de faire s'appelle échantillonnage par importance. Dans ce qui suit on parlera de ces deux méthodes, avec un exemple simple d'application au modèle d'Ising.

4.5.4 Echantillonnage simple

On se spécifie au cas du modèle d'Ising pour illustrer l'échantillonnage simple. Dans ce modèle un spin quelconque ne peut être que dans deux états possibles : up ou +1 et down ou -1. L'échantillonnage simple consiste à générer N configurations aléatoires du système. Dans une configuration donnée on affecte de manière aléatoire un état (up ou down) à chacun des spins formant le système. La valeur moyenne d'une observable A est alors donnée par[3] :

$$\langle A \rangle = \frac{\sum_{s=1}^{N} A(s) \exp[-\beta E(s)]}{\sum_{s=1}^{N} \exp[-\beta E(s)]},$$

où $\beta = 1/k_B T$, et où la sommation s'effectue sur toutes les configurations générées. $A(s)$ est la valeur de l'observable A dans la configuration s, et $E(s)$ est l'énergie de cette configuration. La valeur moyenne $\langle A \rangle$ dépend du nombre de configurations : c'est-à-dire quand N augmente, la précision de la valeur moyenne augmente.

Cette méthode souffre de deux inconvénients majeurs. Le premier est le fait que les configurations ainsi générées correspondent au maximum de l'entropie et par conséquent sont plutôt caractéristiques des hautes températures. Si on veut étudier le système à des températures de l'ordre de ou inférieures à l'interaction mutuelle entre les spins, les configurations générées par l'échantillonnage simple auront un poids statistique négligeable[3] et ne permettent pas un échantillonnage représentatif de l'espace des configurations possibles. Le deuxième inconvénient est que cette méthode est totalement "symétrique", et par conséquent ne permet pas d'observer une éventuelle transition de phase en diminuant la température. Il est vrai que ce problème peut être remédié en introduisant un faible champ appliqué/extérieur qui permettrait de briser la symétrie. C'est pour remédier au plus sérieux de ces deux problèmes que la méthode de l'échantillonnage par importance est mise en avant.

4.5.5 Echantillonnage par importance

En utilisant toujours le modèle d'Ising pour illustration, l'échantillonnage par importance s'agit de générer comme précédemment des configurations de spins de manière aléatoire, mais avec une distribution qui suit de près la distribution de Boltzmann[3]. Ceci permet en effet un meilleur échantillonnage de l'espace des configurations et par conséquent de meilleurs moyennes statistiques, comparativement à l'échantillonnage simple. La valeur moyenne d'une grandeur physique A est maintenant donnée par[3] :

$$\langle A \rangle = \frac{1}{N} \sum_{s=1}^{N} A(s),$$

où N est le nombre de configurations choisies à la température T, et $A(s)$ est la valeur de A dans la configuration s. L'échantillonnage par importance peut être implémenté par un algorithme simple, appelé Metropolis et qu'on détaillera à présent.

4.5.6 Algorithme de Metropolis

Il est l'algorithme Monte Carlo le plus populaire. Il a été créé par Nicolas Metropolis et ces collègues dans les années cinquante pour des simulations de gaz de sphères dures[7]. Il est utilisé aussi pour illustrer plusieurs des concepts généraux impliqués dans un calcul Monte Carlo réel, y compris l'équilibration, la mesure de valeurs moyenne et le calcul des erreurs[8].

Un cycle Monte Carlo consiste à visiter en moyenne tous les spins du système et à tenter de les flipper un à un. On choisit un spin au hasard et puis on calcule la différence d'énergie ΔE qui résulterait si on flippait le spin, c'est à dire si on changeait σ_i en $-\sigma_i$. Si ΔE est négatif, on accepte le flip en accord avec la minimisation de l'énergie interne. Si ΔE est positif, on accepte le flip avec une probabilité de $\exp[-\Delta E/k_B T]$. Ceci est fait pour rendre compte des effets de l'agitation thermique qui permet au système de surpasser des barrières énergétiques afin d'explorer l'espace des configurations. En pratique, on génère un nombre aléatoire r et on le compare à $\exp[-\Delta E/k_B T]$. Le flip est accepté si $r < \exp[-\Delta E/k_B T]$. Si le flip est accepté, on met à jour l'énergie totale et l'aimantation pour tenir compte du changement. Et puis un autre spin est tiré au hasard et on tente de le flipper[3]. On procède de cette manière un nombre de fois égal au nombre de spins dans le système. Le tirage des spins se fait au hasard pour limiter les corrélations entre les différentes configurations ainsi générées. Et en moyenne on visite chaque spin une fois dans un cycle Monte Carlo.

En se spécifiant au modèle d'Ising sur un réseau carré à deux dimensions, les étapes du déroulement de l'algorithme de Metropolis sont les suivantes :

1. On fixe les paramètres du système à simuler : la taille latérale L du système, la constante d'échange J, la température T, et la liste des premiers voisins de chaque spin. Dans cette étape on génère également l'état initial de chaque spin en lui affectant aléatoirement +1 ou -1.
2. On thermalise le système en effectuant un nombre important de cycles Monte Carlo, afin de ramener la température effective du système à la température à laquelle on veut le simuler.
3. On procède à effectuer un nombre important de cycles Monte Carlo afin d'effectuer des mesures. Afin de limiter les corrélations on laisse le système évoluer un certain nombre de cycles entre deux mesure successives.
4. On calcule les moyennes des observables mesurées (énergie totale par exemple) et on calcule d'autres observables qui dépendent de ces moyennes (chaleur spécifique par exemple).

4.5.7 Chaleur spécifique et susceptibilité

La chaleur spécifique est définie comme la dérivée de l'énergie interne par spin par rapport à la température :

$$C_v = \frac{\partial E}{\partial T}.$$

Quand on effectue des simulations numériques cette formule exige qu'on évalue la dérivée numériquement, ce qui donne des résultats pas satisfaisants. Dans les simulations on préfère utiliser une formulation équivalente[3] :

$$C_v = \frac{\langle E^2\rangle - \langle E\rangle^2}{k_B T^2}. \tag{4.2}$$

En effet cette formule ne fait pas usage de dérivation numérique.

La susceptibilité magnétique statique est définie comme la dérivée de l'aimantation par rapport au champ magnétique extérieur :

$$\chi = \left.\frac{\partial M}{\partial h}\right|_{h=0}.$$

Cette formule exige aussi qu'on effectue une dérivation numérique, ce qui introduit des erreurs dans le résultat. Dans les simulations numériques on lui préfère une formulation équivalente[3] :

$$\chi = \frac{\langle M^2\rangle - \langle M\rangle^2}{k_B T}. \tag{4.3}$$

Ce sont ces deux formules pour C_v et pour χ qu'on va utiliser dans l'implémentation plus bas. Ce sont deux exemples de la relation générale entre fonctions de réponse linéaire et fluctuations à l'équilibre. Dans le modèle d'Ising l'aimantation est donnée par

$$M = \sum_{i=1}^{N_s} \sigma_i,$$

où N_s est le nombre de spins dans le système. L'aimantation par spin est alors

$$\langle\sigma\rangle = \frac{\langle M\rangle}{N_s}.$$

4.5.8 Implémentation

1. Initialisation : la fonction `initialize` prend deux arguments : `Lx` et `Ly` qui sont les longueurs latérales du système à simuler. Elle retourne trois variables : `spos` contient les positions des spins, `vois` contient la liste des voisins de chaque spin, et `sstate` contient l'état de chaque spins (up ou down). Il faut noter que la liste des voisins est obtenue en tenant compte des conditions aux limites périodiques.

```
function [spos vois sstate] = initialize(Lx,Ly)
 Ns=Lx*Ly;
 sstate=zeros(1,Ns);
 spos=zeros(2,Ns);
 vois=zeros(4,Ns);

 for i=1:Lx
```

```
  for j=1:Ly
    num = i + (j-1)*Lx;
    spos(1,num)=i-1;
    spos(2,num)=j-1;

    if i==1
      im1=Lx;
    else
      im1=i-1;
    end

    if i==Lx
      ip1=1;
    else
      ip1=i+1;
    end

    if j==1
      jm1=Ly;
    else
      jm1=j-1;
    end

    if j==Ly
      jp1=1;
    else
      jp1=j+1;
    end

    vois(1,num) = im1 + (j-1)*Lx;
    vois(2,num) = ip1 + (j-1)*Lx;
    vois(3,num) = i + (jm1-1)*Lx;
    vois(4,num) = i + (jp1-1)*Lx;

    if rand() < 0.5
      sstate(num) = -1;
    else
      sstate(num) = +1;
    end
  end
end
```

2. Calcul de l'énergie et de l'aimantation : La fonction `compute_et_mt` prend quatre arguments : `sstate` contient l'état de chaque spin, `voisins` contient la liste des quatre voisins de chaque spin, `Ns` le nombre de spins, et `Jexc` est la constante d'échange. Elle retourne deux valeurs : `Et` est l'énergie totale et `Mt` est l'aimantation totale.

```
function [Et Mt] = compute_et_mt(sstate,voisins,Ns,Jexc)
```

```
Et=0;
Mt=0;
for is=1:Ns
  Mt = Mt + sstate(is);
  vois_1=voisins(1,is);
  vois_2=voisins(2,is);
  vois_3=voisins(3,is);
  vois_4=voisins(4,is);
  Et = Et - Jexc*sstate(is) *...
      (sstate(vois_1)+sstate(vois_2)+sstate(vois_3)+sstate(vois_4));
end
Et = Et/2;
```

3. Effectuer un cycle Monte Carlo : Les arguments de la fonction `onestepmc` sont : `sstat` contient l'état de chaque spin, `voisins` contient la liste des voisins de chaque spin, `Ns` est le nombre total de spins, `Jexc` est la constante d'échange, `Temp` est la température en unités de `Jexc`, `Et` est l'énergie totale, et `Mt` est l'aimantation totale. La fonction retourne trois variables déjà définies : `sstate`, `Et`, et `Mt`.

```
function [sstate Et Mt] = onestepmc(sstate,voisins,Ns,Jexc,Temp,...
                                    Et,Mt)

for i=1:Ns
  is = randi(Ns);
  vois_1=voisins(1,is);
  vois_2=voisins(2,is);
  vois_3=voisins(3,is);
  vois_4=voisins(4,is);
  DE = 2*Jexc*sstate(is)*   ...
      (sstate(vois_1)+sstate(vois_2)+sstate(vois_3)+sstate(vois_4));
 if DE<0
   Et = Et + DE;
   Mt = Mt - 2 * sstate(is);
   sstate(is) = -sstate(is);
 elseif exp(-DE/Temp) > rand()
   Et = Et + DE;
   Mt = Mt - 2 * sstate(is);
   sstate(is) = -sstate(is);
 end
end
```

4. Ecriture d'un vecteur de nombres flottants dans un fichier : La fonction `write_vector` effectue cette tâche et accepte trois arguments : `x` est le nom de la variable contenant le vecteur, `vecname` est le nom qu'on veut donner à ce vecteur dans le fichier où on l'écrit, et enfin `fileID` est l'identifiant du fichier vers lequel s'effectue l'écriture.

```
function write_vector(x,vecname,fileID)
 fprintf(fileID,'%s %s',vecname,' = [ ');
 for m=1:length(x)-1
```

```
  fprintf(fileID,'%8.4f %s',x(m),',');
 end
 fprintf(fileID,'%8.4f %s\n',x(end),'];');
```

5. Programme principal : dans ce script, on appelle d'abord la fonction `initialize` pour fixer les paramètres du système. Puis on appelle la fonction `compute_et_mt` pour calculer l'énergie et l'aimantation totales. En suite on entre dans une boucle pour effectuer un certain nombre `NMCCEQ` de cycles Monte Carlo, par le biais de la fonction `onestepmc`, afin d'atteindre l'équilibre à la plus haute température qu'on veut simuler. Une fois l'équilibre atteint, on entame une boucle qui va fixer la température en commençant par la plus haute, jusqu'à la plus basse. A l'intérieur de cette boucle de température, il y a une boucle qui effectue un certain nombre `NMC` de cycles Monte Carlo, toujours par le biais de la fonction `onestepmc`, afin d'effectuer des mesures. A la fin de cette boucle les moyennes requises sont effectuées, et les valeurs de la chaleur spécifique et de la susceptibilité calculée. Les valeurs des quantités calculées sont mises dans des vecteurs et ces vecteurs sont écrits, à l'aide des fonctions `write_vector` et `fprintf`, à la fin dans un fichier appelé `results.m`. Celui-ci est écrit sous forme d'un script qu'il suffit d'exécuter par la suite pour obtenir les graphes des quantités calculées.

```
clear all

Lx=16;
Ly=Lx;
Ns=Lx*Ly;
Jexc = 1;
Ntemp=50;
Temp=linspace(0.1,5,Ntemp);

NMCCEQ=1000;
NMC=40000;

[spinpos neighbors spinstate] = initialize(Lx,Ly);

[Et Mt] = compute_et_mt(spinstate,neighbors,Ns,Jexc);

for imc=1:NMCCEQ
  [spinstate Et Mt] = onestepmc(spinstate,neighbors,Ns,Jexc,...
                                Temp(Ntemp),Et,Mt);
end

fprintf('=== Thermalization done. ==== \n');
fprintf(' Et = %g     Mt = %g \n',Et/Ns,Mt/Ns);

Em=zeros(1,Ntemp);
E2m=zeros(1,Ntemp);
Mm_abs=zeros(1,Ntemp);
M2m=zeros(1,Ntemp);
Cv=zeros(1,Ntemp);
ksi=zeros(1,Ntemp);
```

```
for it=Ntemp:-1:1
  NbMes=0;
  for imc=1:NMC
     [spinstate Et Mt] = onestepmc(spinstate,neighbors,Ns,Jexc,...
                                        Temp(it),Et,Mt);
     if rem(imc,10)==0
       Em(it) = Em(it) + Et/Ns;
       E2m(it) = E2m(it) + Et*Et/Ns/Ns;
       Mm_abs(it) = Mm_abs(it) + abs(Mt)/Ns;
       M2m(it) = M2m(it) + Mt*Mt/Ns/Ns;
       NbMes = NbMes + 1;
     end
  end % imc=1:NMC

  Em(it)=Em(it)/NbMes;
  E2m(it)=E2m(it)/NbMes;
  Cv(it)=(E2m(it)-Em(it)*Em(it))/(Temp(it)*Temp(it));
  Mm_abs(it)=Mm_abs(it)/NbMes;
  M2m(it)=M2m(it)/NbMes;
  ksi(it)=(M2m(it)-Mm_abs(it)*Mm_abs(it))/Temp(it);
end % it=Ntemp:-1:1

fileID = fopen('results.m','w');
fprintf(fileID,'%% Results for Lx = %d \n',Lx);

write_vector(Temp,'Temp',fileID);
write_vector(Em,'Em',fileID);
write_vector(Cv,'Cv',fileID);
write_vector(Mm_abs,'Mm_abs',fileID);
write_vector(ksi,'ksi',fileID);

fprintf(fileID,'\n%s\n','subplot(2,2,1)');
fprintf(fileID,'%s\n','plot(Temp,Em,''LineWidth'',4)');
fprintf(fileID,'title(''E moyenne'')\n');
fprintf(fileID,'%s\n','grid');

fprintf(fileID,'\n%s\n','subplot(2,2,3)');
fprintf(fileID,'%s\n','plot(Temp,Cv,''LineWidth'',4)');
fprintf(fileID,'title('' chaleur specifique '')\n');
fprintf(fileID,'%s\n','grid');

fprintf(fileID,'\n%s\n','subplot(2,2,2)');
fprintf(fileID,'%s\n','plot(Temp,Mm_abs,''LineWidth'',4)');
fprintf(fileID,'title('' M moyenne'')\n');
fprintf(fileID,'%s\n','grid');
```

```
fprintf(fileID,'\n%s\n','subplot(2,2,4)');
fprintf(fileID,'%s\n','plot(Temp,ksi,''LineWidth'',4)');
fprintf(fileID,'title('' susceptibilite '')\n');
fprintf(fileID,'%s\n','grid');
fclose(fileID);
```

6. Exemple de fichier de sortie `results.m` :[1]

```
% Results for Lx = 16
Temp = [0.1000 , 0.2000 , 0.3000 , 0.4000 , 0.5000 , 0.6000 ,
        0.7000 , 0.8000 , 0.9000 , 1.0000 , 1.1000 , 1.2000 ,
        1.3000 , 1.4000 , 1.5000 , 1.6000 , 1.7000 , 1.8000 ,
        1.9000 , 2.0000 , 2.1000 , 2.2000 , 2.3000 , 2.4000 ,
        2.5000 , 2.6000 , 2.7000 , 2.8000 , 2.9000 , 3.0000 ,
        3.1000 , 3.2000 , 3.3000 , 3.4000 , 3.5000 , 3.6000 ,
        3.7000 , 3.8000 , 3.9000 , 4.0000 , 4.1000 , 4.2000 ,
        4.3000 , 4.4000 , 4.5000 , 4.6000 , 4.7000 , 4.8000 ,
        4.9000 , 5.0000 ];
Em = [-2.0000 , -2.0000 , -2.0000 , -2.0000 , -2.0000 , -2.0000 ,
      -1.9999 , -1.9996 , -1.9988 , -1.9975 , -1.9940 , -1.9886 ,
      -1.9808 , -1.9679 , -1.9515 , -1.9283 , -1.8972 , -1.8580 ,
      -1.8095 , -1.7487 , -1.6605 , -1.5523 , -1.4075 , -1.2534 ,
      -1.1342 , -1.0430 , -0.9685 , -0.9100 , -0.8633 , -0.8189 ,
      -0.7828 , -0.7464 , -0.7146 , -0.6847 , -0.6638 , -0.6389 ,
      -0.6149 , -0.5973 , -0.5732 , -0.5570 , -0.5381 , -0.5252 ,
      -0.5084 , -0.4964 , -0.4841 , -0.4741 , -0.4587 , -0.4463 ,
      -0.4396 , -0.4270 ];
Cv = [ 0.0000 , 0.0000 , 0.0000 , 0.0000 , 0.0000 , 0.0000 ,
       0.0000 , 0.0000 , 0.0000 , 0.0001 , 0.0002 , 0.0003 ,
       0.0004 , 0.0006 , 0.0007 , 0.0011 , 0.0013 , 0.0018 ,
       0.0021 , 0.0027 , 0.0039 , 0.0050 , 0.0059 , 0.0054 ,
       0.0043 , 0.0032 , 0.0025 , 0.0021 , 0.0018 , 0.0016 ,
       0.0014 , 0.0012 , 0.0012 , 0.0010 , 0.0010 , 0.0009 ,
       0.0008 , 0.0008 , 0.0007 , 0.0007 , 0.0006 , 0.0006 ,
       0.0005 , 0.0005 , 0.0005 , 0.0004 , 0.0004 , 0.0004 ,
       0.0004 , 0.0004 ];
Mm_abs =[1.0000 , 1.0000 , 1.0000 , 1.0000 , 1.0000 , 1.0000 ,
         1.0000 , 0.9999 , 0.9997 , 0.9994 , 0.9985 , 0.9970 ,
         0.9949 , 0.9914 , 0.9867 , 0.9796 , 0.9698 , 0.9563 ,
         0.9383 , 0.9129 , 0.8659 , 0.7977 , 0.6678 , 0.5141 ,
         0.3931 , 0.3116 , 0.2541 , 0.2130 , 0.1932 , 0.1720 ,
         0.1628 , 0.1461 , 0.1384 , 0.1298 , 0.1268 , 0.1191 ,
         0.1145 , 0.1096 , 0.1080 , 0.1032 , 0.0996 , 0.0985 ,
         0.0967 , 0.0950 , 0.0935 , 0.0930 , 0.0875 , 0.0853 ,
         0.0872 , 0.0831 ];
ksi = [ 0.0000 , 0.0000 , 0.0000 , 0.0000 , 0.0000 , 0.0000 ,
        0.0000 , 0.0000 , 0.0000 , 0.0000 , 0.0000 , 0.0000 ,
```

1. Le fichier a été édité pour les besoins de la mise en page.

```
          0.0000 ,  0.0001 ,  0.0001 ,  0.0002 ,  0.0003 ,  0.0005 ,
          0.0008 ,  0.0013 ,  0.0040 ,  0.0087 ,  0.0193 ,  0.0238 ,
          0.0209 ,  0.0152 ,  0.0113 ,  0.0086 ,  0.0066 ,  0.0053 ,
          0.0045 ,  0.0035 ,  0.0031 ,  0.0028 ,  0.0024 ,  0.0022 ,
          0.0019 ,  0.0018 ,  0.0016 ,  0.0015 ,  0.0013 ,  0.0013 ,
          0.0012 ,  0.0011 ,  0.0011 ,  0.0010 ,  0.0010 ,  0.0009 ,
          0.0009 ,  0.0008 ];

subplot(2,2,1) plot(Temp,Em,'LineWidth',4)
title('E moyenne')
grid

subplot(2,2,3)
plot(Temp,Cv,'LineWidth',4')
title(' chaleur specifique ')
xlabel('T')
grid

subplot(2,2,2)
plot(Temp,Mm_abs,'LineWidth',4')
title(' M moyenne')
grid

subplot(2,2,4)
plot(Temp,ksi,'LineWidth',4')
title(' susceptibilite ')
xlabel('T')
grid
```

4.5.9 Résultats

Dans le script principal introduit plus haut la taille latérale du système est fixée à 16, et les nombres de cycles Monte Carlo pour la thermalisation et les mesures sont fixés respectivement à 1000 et 40000. L'exécution du script, comme expliqué, résulte en la création d'un autre script, appelé `results.m`, contenant les valeurs des quantités calculées, ainsi que des instructions pour tracer les graphes de ces quantités en fonction de la température. L'exécution du script `results.m` dans ce cas donne les graphes montrés dans la figure 4.3 .

On voit que l'énergie moyenne par spin démarre par la valeur de -2 pour les basses températures. Cette valeur est facile à retrouver si on sait que pour ces températures tous les spins sont alignés dans la même direction. Un spin quelconque est entouré de quatre voisins. Ce qui donne une contribution de $-4J$ à l'énergie totale. Sauf qu'il faut diviser cette valeur par deux pour éviter de compter les liaisons deux fois. On obtient donc bien une valeur de $-2J$, ou bien -2 si on prend J comme unité de mesure de l'énergie, ce qu'on a bien sur fait ici. Par la suite l'énergie augmente de manière monotone en fonction de la température. Mais une inflexion de la courbe est observable entre $T = 2$ et $T = 3$. Pour des températures plus élevée l'énergie augmente toujours mais moins rapidement. Il est facile de montrer que pour les très hautes températures l'énergie est nulle. Ceci vient du fait qu'un spin quelconque est entouré de quatre voisins, qui ont la même probabilité d'être up que d'être down. Ce

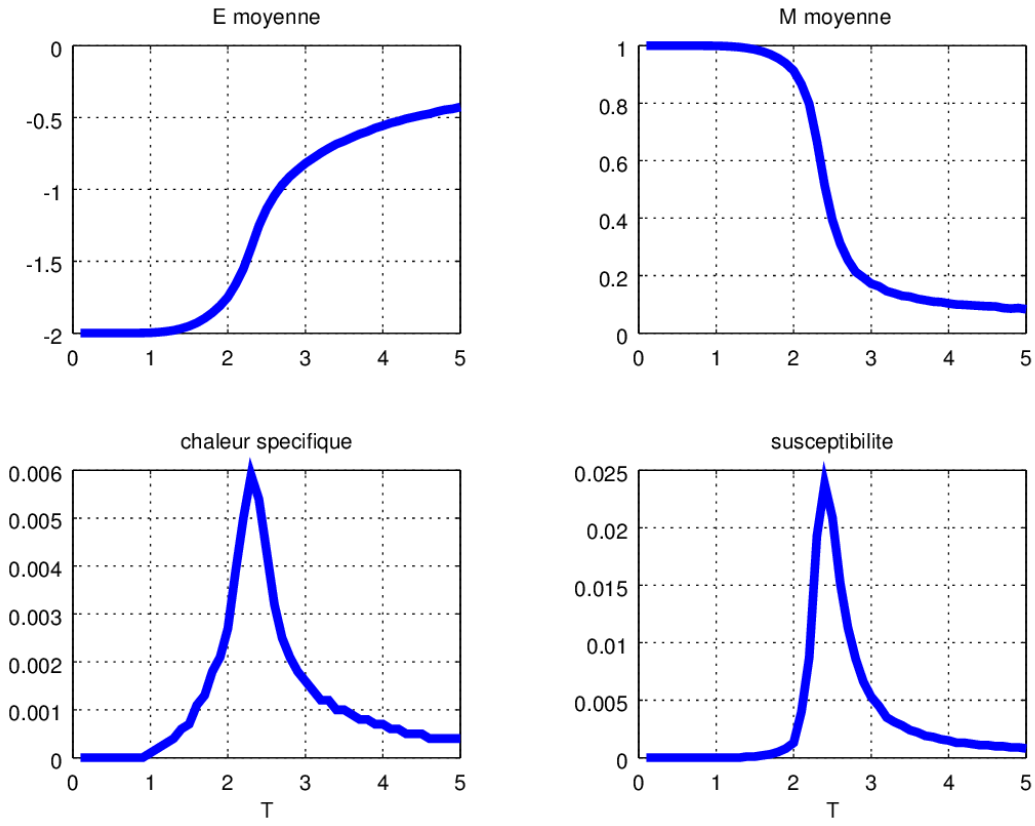

FIGURE 4.3 – *Graphes de l'énergie moyenne par spin, de l'aimantation par spin, de la chaleur spécifique, et de la susceptibilité magnétique en fonction de la température pour le modèle d'Ising ferromagnétique sur un réseau carré de taille latérale* $L = 16$.

qui donne zéro si on calcule leurs contribution à l'énergie totale. La raison qu'on ne trouve pas zéro ici est due au fait qu'on a fixé la plus haute température à $T = 5J$, ce qui n'est apparemment pas assez élevé.

A très basse température l'aimantation par spin est saturée à $\langle\sigma\rangle = 1$, ce qui est dû à l'alignement parfait de tous les spins dans une seul direction. A mesure que la température augmente l'aimantation diminue graduellement. Comme pour l'énergie, la diminution est plus rapide entre $T = 2$ et $T = 3$, où la courbe de l'aimantation enregistre elle aussi une inflexion. Pour des températures plus élevées l'aimantation continue de diminuer, mais avec un taux moins élevé, pour s'annuler aux très hautes températures. En principe on a affaire ici à une transition de la phase ferromagnétique à basse température, à la phase paramagnétique à haute température, comme on l'a vu déjà dans le traitement du modèle d'Ising en champ moyen. De plus, cette transition doit être abrupte. C'est à dire qu'à partir d'une température critique T_C on doit avoir $\langle\sigma\rangle = 0$. Deux raisons font qu'on ne voit pas cette transition abrupte dans ces résultats. La principale raison est que le système étudié n'est pas infini. Or les transitions de phase n'existent en principe que pour des systèmes infinis. La deuxième raison vient du fait que ce qu'on calcule est au fait la valeur moyenne de la valeur absolue de l'aimantation, c'est à dire $\langle|\sigma|\rangle$ au lieu de $\langle\sigma\rangle$. Pour comprendre pourquoi on fait ce choix, il faut se rappeler qu'en termes de poids statistique, une configuration avec une aimantation positive a le même poids que celle qu'on obtiendrait en flippant tous les spins de cette configuration. Ainsi si on calcule rigoureusement la valeur moyenne $\langle\sigma\rangle$, on trouverait toujours zéro. Dans des simulations Monte Carlo, il se trouve que la valeur moyenne $\langle\sigma\rangle$ va être nulle sur une large gamme de températures, c'est à dire même au-dessous de T_C. Aux très basses températures la symétrie est brisée, et le système choisit une direction donnée pour l'aimantation, mais si le système est petit, on peut observer des renversement de l'aimantation au cours de la simulation[9]. C'est pour toutes ces raison qu'on choisit de calculer la valeur moyenne de la valeur absolue de l'aimantation.

La chaleur spécifique est nulle au basses températures comme on le voit sur la figure 4.3. La raison est le fait que le modèle d'Ising a une symétrie discrète et son spectre d'excitation possède un gap. En effet le minimum d'énergie qu'il faut fournir pour exciter le système correspond au flip d'un seul spin. Cette énergie est de $4J$, donc non-nulle. Aux très hautes températures l'énergie du système sature à la valeur de zéro. C'est à dire que l'énergie est constante en fonction de la température. Ce qui donne une chaleur spécifique nulle. Ce qui est remarquable dans la courbe de la chaleur spécifique c'est la présence d'un pic entre $T = 2$ et $T = 3$. Ce pic est bien sur lié à l'inflexion qu'on a déjà évoquée dans la discussion de la courbe de l'énergie. Il nous renseigne sur la présence d'une transition de phase qui a lieu à une température proche de celle où on observe le pic. En principe la chaleur spécifique est divergente à la transition, mais on ne voit qu'un pic dans la figure. La raison est dû au fait que le système étudié est fini. C'est la même raison qui a fait qu'on n'a pas observé une transition abrupte de l'aimantation. La valeur de la température de transition donnée par le pic est $T_{cv} = 2.3$. A noter bien sur que le pas de température utilisé dans ce calcul est de 0.1, et on est donc incapable d'obtenir la température avec une précision inférieure à ce pas. Il se trouve que cette valeur n'est pas trop loin de la valeur exacte de $T_C = 2.269$ obtenue à partir de la solution analytique d'Onsager[10].

Aux très basses températures les spins sont tous aligné suivant la même direction, et l'aimantation est par conséquent saturée. Si on applique un champ magnétique à ce système, rien ne change quant à cette aimantation. Ce qui donne une susceptibilité magnétique nulle à ces températures, comme on le voit sur la figure 4.3. Aux très hautes températures l'agitation thermique empêche toute formation d'une aimantation, même en présence d'un champ magnétique. Ce qui donne là aussi une susceptibilité nulle. La susceptibilité devient importante pour les températures intermédiaire. Et comme pour la chaleur spécifique, ce qui est marquant dans la courbe de la susceptibilité est la présence d'un pic prononcé entre $T = 2$ et $T = 3$. Ce pic indique que même avec un champ très faible,

le système acquiert une aimantation finie. Dans la réalité, la susceptibilité diverge, mais le système étudié n'étant pas infini, on ne voit pas cette divergence. La présence de la divergence indique que le système acquiert spontanément une aimantation non nulle. La température à laquelle on observe le pic doit correspondre à la température de transition. Dans ces calculs on trouve $T_\chi = 2.4$. Ce qui est proche elle aussi de la valeur exacte de $T_C = 2.269$.

Annexe A

Problèmes

A.1 Chapitre 1

1. Résoudre le système $A\,x = b$, où

$$A = \begin{pmatrix} 2 & 3 & -1 \\ 1 & -2 & 5 \\ 6 & -3 & 15 \end{pmatrix},$$

et

$$b = \begin{pmatrix} 1 \\ 3 \\ 5 \end{pmatrix}.$$

2. Tracer le graphe de la fonction $y = \tan(x)$ dans l'intervalle $[0\ 2\pi]$, et en montrant uniquement les parties où y est dans l'intervalle $[-10\ 10]$.
3. Ecrire une fonction qui donne l'aire d'un cercle, et qui prend un seul argument : le rayon du cercle.
4. Ecrire une fonction qui calcule l'écart quadratique moyen en s'inspirant de la fonction `moyenne` dans le texte.

A.2 Chapitre 2

1. La méthode de la sécante est apparentée à la méthode de Newton-Raphson, où la dérivée de la fonction au point x_i, i.e. $f'(x_i)$, est remplacée par[1]

$$f'(x_i) \longrightarrow \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}.$$

   Ecrire une fonction Matlab/Octave qui implémente la méthode la sécante. A noter que la méthode de la sécante nécessite deux points de départ, x_0 et x_1, au lieu d'un seul pour la méthode Newton-Raphson.
2. La méthode de Simpson pour calculer des intégrales définies repose sur une interpolation quadratique de la fonction à intégrer. En ce sens elle ressemble à la méthode des trapèzes, à ceci près que cette dernière consiste en une interpolation linéaire. Pour calculer

$$I = \int_a^b f(x)\,dx,$$

en utilisant la méthode de Simpson, on subdivise l'intervalle $[a\ b]$ en N sous-intervalles de largeur $h = (b-a)/N$ chacun, et où N est un nombre pair. La valeur approchée de l'intégral est alors

$$I = \int_a^b f(x)\,dx \approx \frac{h}{3}\left[f(x_0) + 2\sum_{i=1}^{N/2-1} f(x_{2i}) + 4\sum_{i=1}^{N/2} f(x_{2i-1}) + f(x_N)\right],$$

où $x_i = a + i \times h$[1, 2, 3]. Ecrire une fonction Matlab/Octave qui implémente la méthode de Simpson.

3. Dans la méthode de Runge-Kutta d'ordre 4 la fonction f doit être évaluée quatre fois à chaque pas d'intégration. On évalue d'abord les quatre variables intermédiaires :

$$\begin{aligned} k_1 &= h\,f(t_i, y_i); \\ k_2 &= h\,f(t_i + \frac{h}{2}, y_i + \frac{k_1}{2}); \\ k_3 &= h\,f(t_i + \frac{h}{2}, y_i + \frac{k_2}{2}); \\ k_4 &= h\,f(t_i + h, y_i + k_3); \end{aligned}$$

pour ensuite évaluer y_{i+1} comme[1, 2]

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4).$$

Implémenter la méthode de Runge-Kutta d'ordre 4 pour une équation différentielle ordinaire de premier ordre. Faire un test avec l'équation

$$\frac{dy}{dt} = -y$$

dans l'intervalle [0 6] et avec la condition initiale $y_0 = 1$. Comparer à la solution exacte.

4. Implémenter la méthode de Runge-Kutta d'ordre 4 pour une équation différentielle ordinaire de second ordre. Faire un test avec l'équation

$$\frac{d^2y}{dt^2} = -y$$

dans l'intervalle [0 10] et avec les conditions initiales $y_0 = 1$ et $y_0' = 0$. Comparer à la solution exacte.

A.3 Chapitre 3

1. Retrouver les fonctions d'ondes des états liés à partir de la solution analytique. Tracer ces courbes et comparer les à celles obtenues avec les deux autres méthodes.
2. Expliquer comment trouver quelques unes des premières énergies les plus basses et les fonctions propres correspondantes pour un oscillateur harmonique à une dimension. A la différence du cas du puits fini, tous les états du potentiel harmoniques sont des états liés.

3. Utiliser la méthode spectrale pour trouver les énergies et fonctions propres pour un système de deux puits de potentiels finis identiques, séparés par une distance de l'ordre de la largeur d'un des puits.
4. Etudier la diffusion d'un paquet d'onde par un puits de potentiel fini, ainsi que l'effet tunnel par des choix appropriés de V_0 et de sa la largeur. Cette dernière est fixée dans le fichier potentiel.

A.4 Chapitre 4

1. Ecrire une fonction Matlab/Octave qui calcule une intégrale triple à l'aide de la méthode de Monte Carlo.
2. Résoudre numériquement l'équation transcendante 4.1 pour trouver $< \sigma >$ en fonction de la température T, et retracer ainsi la courbe de la figure 4.2.
3. Pour trouver avec précision la valeur de la température critique dans les simulations Monte Carlo, on utilise une quantité appelée cumulant de Binder[11]. Il est défini par

$$U_L = 1 - \frac{\langle M^4 \rangle_L}{3 \langle M^2 \rangle_L^2},$$

où M est l'aimantation, et L est la taille du système. On calcule le cumulant en fonction de la température pour différentes tailles L du système. En traçant les différents cumulants sur un même graphe, on trouve qu'ils s'intersectent en un point qui correspond à la température critique. Rajouter au programme Monte Carlo quelques instructions qui permettent de calculer le cumulant de Binder et de l'imprimer dans le fichier de sortie.

4. Etudier le modèle d'Ising sur un réseau cubique à trois dimensions.

Bibliographie

[1] S. E. Koonin, et D. C. Meredith, *Computational Physics, Fortran Version*, (Westview Press 1990).

[2] D. Yang, *C++ and Object Oriented Numeric Computing for Scientists and Engineers*, (Springer-Verlag, New York 2001).

[3] H. Gould, J. Tobochnik, et W. Christia, *An Introduction to Computer Simulation Methods* (third edition), (Addison-Wesley 2007).

[4] D. J. Griffiths, *Introduction to Quantum Mechanics* (second edition), (Pearson Education 2005).

[5] N. J. Giordano, et H. Nakanishi, *Computational Physics* (second edition), (Pearson Education 2006).

[6] J. M. Thijssen, *Computational Physics* (second edition), (Cambridge University Press 2007).

[7] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, E. Teller, J. Chem. Phys. **21**, 1087 (1953).

[8] M. E. J. Newman, et G. T. Barkema, *Monte Carlo methods in statistical physics*, (Oxford University Press 2001).

[9] J. Kotze, arXiv :0803.0217 [cond-mat.stat-mech] (2008).

[10] L. Onsager, Phys. Rev. **65**, 117 (1944).

[11] K. Binder, Z. Phys. B–Condensed Matter **43**, 119 (1981).