PHQ404 - Devoir 2

Théo Lenoir (lent1201) - Yuhao Dong

1 Construction du Hamiltonien

Pour construire ce hamiltonien, nous utilisons simplement la définition :

$$H = J \sum_{i=1}^{N} \vec{S_i}.\vec{S_{i+1}}$$

Ici, pour chaque i nous pouvons définir $\vec{S_i}$ par :

$$\vec{S_i} = S_i^x + S_i^y + S_i^z$$

et on a ainsi :

$$\vec{S_i}.\vec{S_{i+1}} = S_i^x S_{i+1}^x + S_i^y S_{i+1}^y + S_i^z S_{i+1}^z$$

avec chaque terme défini en utilisant les produits de Kronecker. Considérons tout d'abord le cas $i = 1$ et seulement l'opérateur agissant sur les x. Nous avons :

$$\begin{aligned} S_1^x S_2^x &= (S^x \otimes I_2 \otimes ... \otimes I_2).(I_2 \otimes S^x \otimes I_2 \otimes ... \otimes I_2) \\ &= (S^x \otimes I_2 \otimes I_{2^{N-2}}).(I_2 \otimes S^x \otimes I_{2^{N-2}}) \\ &= [(S^x \otimes I_2).(I_2 \otimes S^x)] \otimes I_{2^{N-2}} \end{aligned}$$

où S_x représente la matrice de Pauli agissant sur les x. Nous avons ici factorisé les matrices identités à droite de l'expression. Nous pouvons en déduire l'expression globale :

$$\vec{S_1}.\vec{S_2} = [(S^x \otimes I_2).(I_2 \otimes S^x) + (S^y \otimes I_2).(I_2 \otimes S^y) + (S^z \otimes I_2).(I_2 \otimes S^z)] \otimes I_{2^{N-2}}$$

et on peut remarquer que le terme entre crochets est un terme constant, exprimés seulement en fonction de matrice identités et de matrices de Pauli. Nous le nommons σ. Dans le cas général pour i quelconque (mais différent de N), nous pouvons effectuer un calcul similaire en factorisant également à gauche, et nous avons :

$$\vec{S_i}.\vec{S_{i+1}} = I_{2^{i-1}} \otimes \sigma \otimes I_{2^{N-i-1}}$$

Pour le cas particulier de $i = N$, nous avons simplement :

$$\vec{S_N}.\vec{S_1} = (I_{2^{N-1}} \otimes S^x).(S^x \otimes I_{2^{N-1}}) + (I_{2^{N-1}} \otimes S^y).(S^y \otimes I_{2^{N-1}}) + (I_{2^{N-1}} \otimes S^z).(S^z \otimes I_{2^{N-1}})$$

Et en sommant tous ces termes, ainsi qu'en les multipliant par J, on obtient alors l'expression de l'hamiltonien souhaitée. Notons que pour l'implémentation en python, il faut que toutes ces matrices soit déclarées en tant que matrices creuses pour faciliter le calcul.

2 Vérification du Hamiltonien

Nous pouvons utiliser la description du calcul précédent pour comparer notre implémentation en python avec le résultat analytique attendu. Détaillons ainsi le calcul, en prenant par exemple $J = 4$. On a :

$$\begin{aligned} H &= 4\sum_{1}^{3} \vec{S_i}.\vec{S_{i+1}} \\ &= 4(\vec{S_1}.\vec{S_2} + \vec{S_2}.\vec{S_3} + \vec{S_3}.\vec{S_1}) \\ &= 4(\sigma \otimes I_2 + I_2 \otimes \sigma + \vec{S_3}.\vec{S_1}) \end{aligned}$$

Nous pouvons calculer la valeur de σ :

$$\sigma = (S^x \otimes I_2).(I_2 \otimes S^x) + (S^y \otimes I_2).(I_2 \otimes S^y) + (S^z \otimes I_2).(I_2 \otimes S^z)$$

Calculons chacune des composantes une par une :

$$\begin{aligned} \sigma_x &= (S^x \otimes I_2).(I_2 \otimes S^x) \\ &= \frac{1}{2}\left[\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\right] . \left[\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \frac{1}{2}\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\right] \\ &= \frac{1}{4}\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} . \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \\ &= \frac{1}{4}\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \end{aligned}$$

et de même, on a :

$$\sigma_y = \frac{1}{4}\begin{pmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{pmatrix}$$

et :

$$\sigma_z = \frac{1}{4}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

et on en déduit l'expression complète de σ :

$$\sigma = \frac{1}{4}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 2 & 0 \\ 0 & 2 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

On retrouve par ailleurs (logiquement) l'expression du hamiltonien pour $N = 2$. On a ensuite :

$$\sigma \otimes I_2 = \frac{1}{4}\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

ainsi que :

$$I_2 \otimes \sigma = \frac{1}{4}\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

on a aussi :

$$\vec{S_3}.\vec{S_1} = (I_4 \otimes S^x).(S^x \otimes I_4) + (I_4 \otimes S^y).(S^y \otimes I_4) + (I_4 \otimes S^z).(S^z \otimes I_4)$$

$$= \frac{1}{4}\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 2 & 0 \\ 0 & 2 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

et on arrive alors à :

$$H = \begin{pmatrix} 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & 0 & 2 & 0 & 0 & 0 \\ 0 & 2 & -1 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 2 & 2 & 0 \\ 0 & 2 & 2 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & -1 & 2 & 0 \\ 0 & 0 & 0 & 2 & 0 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \end{pmatrix}$$

ce qui est bien le résultat retourné par notre algorithme. Celui-ci est vérifié pour $N = 3$.

3 Calcul des basses énergies

Le calcul des énergies pour les différents N revient à calculer les valeurs propres du hamiltoniens pour ces valeurs. Les valeurs propres les plus faibles correspondront ensuite aux états de plus basses énergies. Il nous suffit pour cela d'utiliser la fonction *eigsh* de la bibliothèque *scipy.sparse.linalg* pour calculer ces valeurs. Cette fonction agit bien sur des matrices creuses ce qui permet de limiter le temps de calcul.

4 Algorithme epsilon

Pour construire l'algorithme epsilon, on reconstruit tout d'abord le triangle nécessaire à cet algorithme. On utilise pour cela les règles suivantes :

$$\varepsilon_0^j = 0 \text{ et } \varepsilon_1^j = S_j$$

où les S_j sont les valeurs de la suite dont on cherche la limite. On complète ensuite le triangle selon la règle suivante :

$$\text{Si } j < n - k : \varepsilon_k^j = \varepsilon_{k-2}^{j+1} + \frac{1}{\varepsilon_{k-1}^{j+1} - \varepsilon_{k-1}^j}$$

On vérifie également la condition que $\varepsilon_{k-1}^{j+1} - \varepsilon_{k-1}^j \neq 0$. Un exemple de triangle généré par ce code est donné en figure 1 :

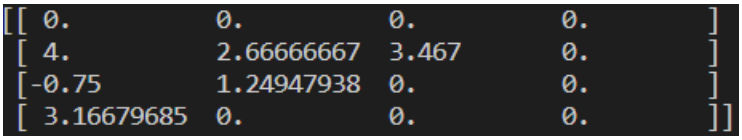

FIGURE 1 – Exemple de triangle dans l'algorithme epsilon

et la valeur de limite prédite par cette algorithme se trouve sur la pointe du triangle.

5 Vérification de l'algorithme epsilon

On utilise la suite des somme partielles du développement limité de l'arctangente en 0 pour vérifier l'implémentation de notre algorithme epsilon. On considère pour cela 4, 5, 6 ou 7 termes dans la série. On remarque que pour un nombre pair de termes dans le vecteur inital, l'algorithme ne nous donne pas une précision intéressante. Cela vient du fait que le nombre d'élements dans la base du triangle ne nous permet pas de former un triangle complet avec une pointe qui nous donnerait une prédiction correcte. Cela est le cas cependant avec un nombre impair de termes, et la précision de l'approximation de π augmente avec le nombre de termes, jusqu'à atteindre $\pi \approx 3.14161$ avec 7 termes.

6 Extrapolation vers $N \to +\infty$

On s'intéresse à deux séries que l'on peut former à partir des énergies calculées précédemment, à savoir E_0/N et $E_1 - E_0$. On utilise plusieurs approximation, en utilisant soit tous les points, soit seulement les points pairs, ou seulement les points impairs. On applique l'algorithme epsilon à ces suites pour en estimer la limite. On observe des comportements très différents en fonction de la suite initiale considérée. En effet, pour E_0/N, le résultat converge avec dépassement en considérant tous les termes, ne converge pas avec seulement les termes pairs et converge sans dépassement (mais plus lentement) avec seulement les termes impairs. Pour $E_1 - E_0$, le calcul est impossible avec tous les termes (la condition de division non nulle dans l'algorithme epsilon n'est pas respectée), et le calcul ne converge que si l'on considère les termes impairs. Il est cependant logique avec ce que l'on a vu avant que le calcul me converge pas avec les termes pairs seulement, étant donné que le triangle ne présente pas de pointe. Les graphiques de ces résultats sont montrés sur les figures suivantes :

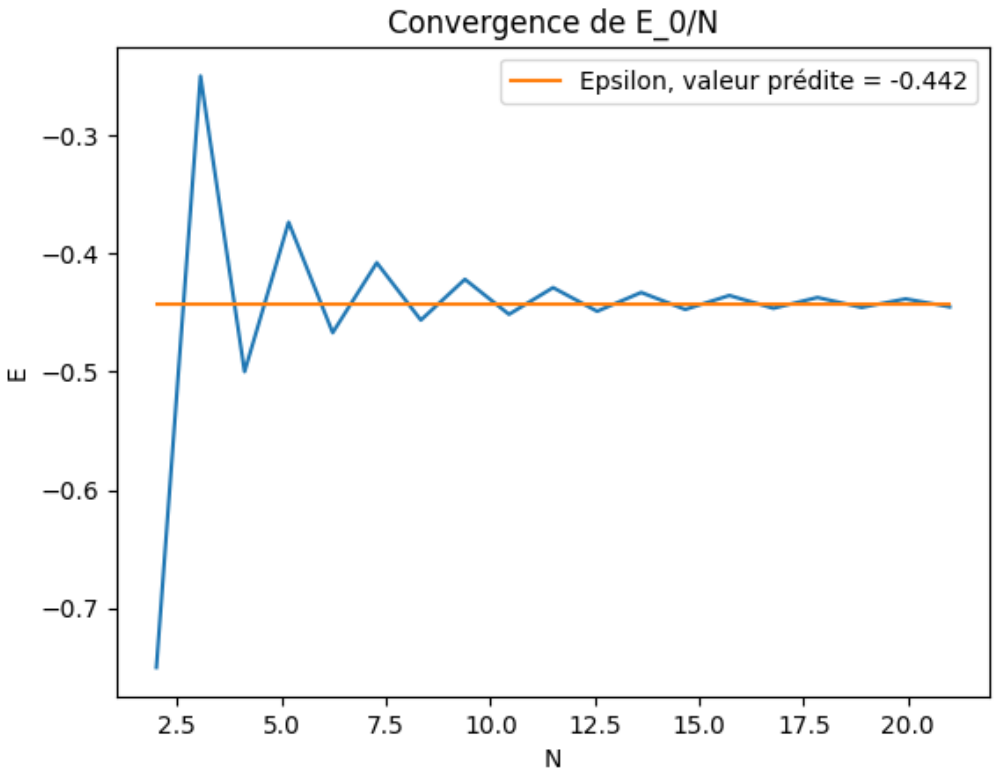
Convergence de E_0/N
Epsilon, valeur prédite = -0.442
−0.3
−0.4
−0.5
−0.6
−0.7
E
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
N

FIGURE 2 –

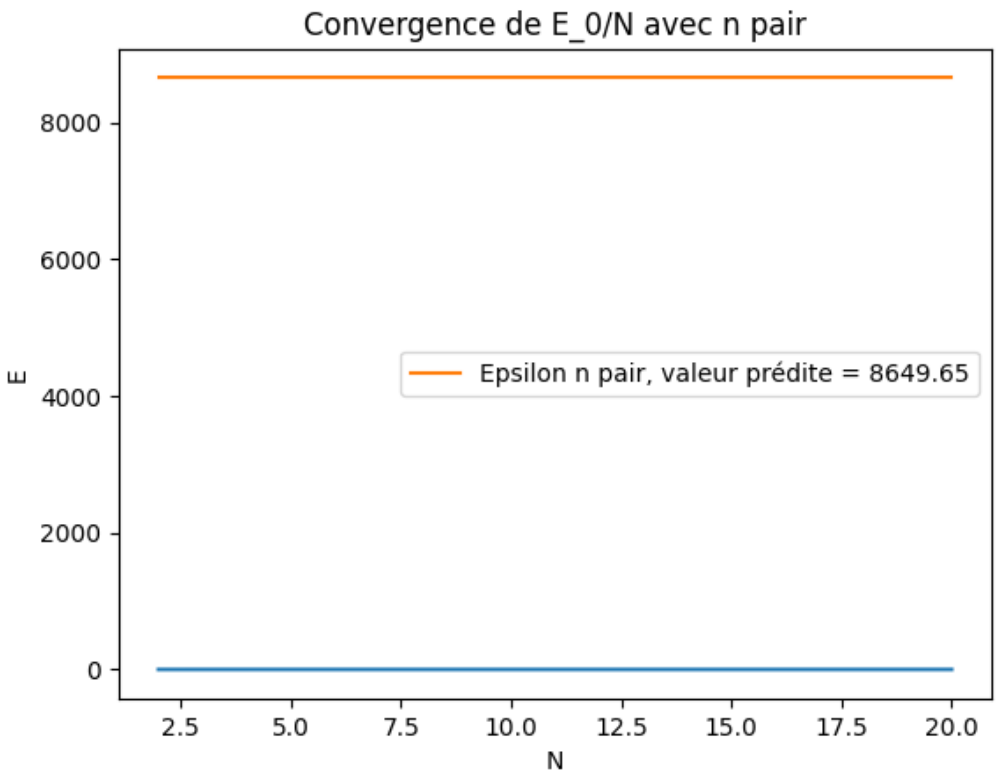
Convergence de E_0/N avec n pair
8000
6000
4000
2000
0
E
Epsilon n pair, valeur prédite = 8649.65
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
N

FIGURE 3 –

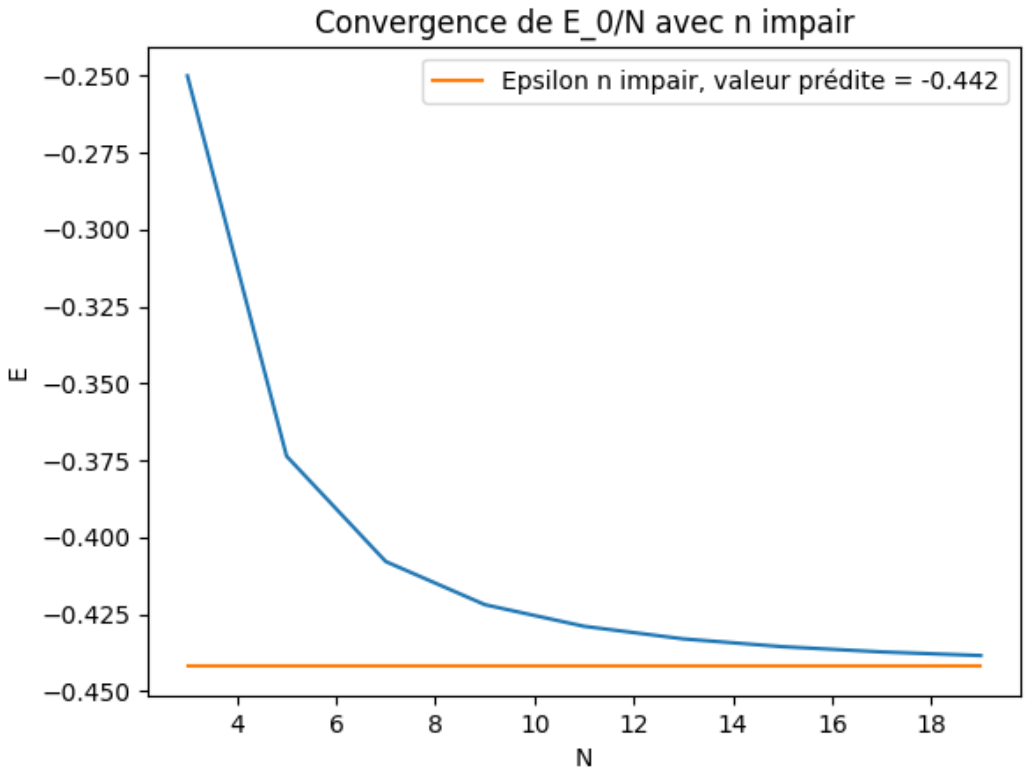
Convergence de E_0/N avec n impair
Epsilon n impair, valeur prédite = -0.442
−0.250
−0.275
−0.300
−0.325
−0.350
−0.375
−0.400
−0.425
−0.450
E
4
6
8
10
12
14
16
18
N

FIGURE 4 –

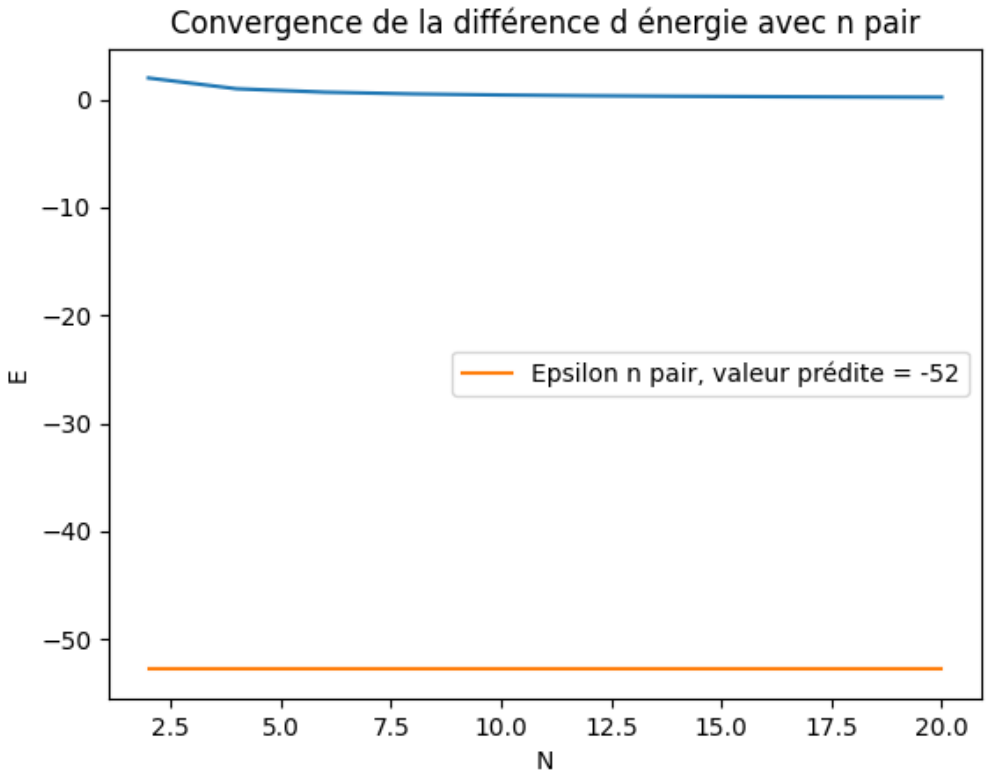
Convergence de la différence d énergie avec n pair
0
−10
−20
−30
−40
−50
E
Epsilon n pair, valeur prédite = -52
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
N

FIGURE 5 –

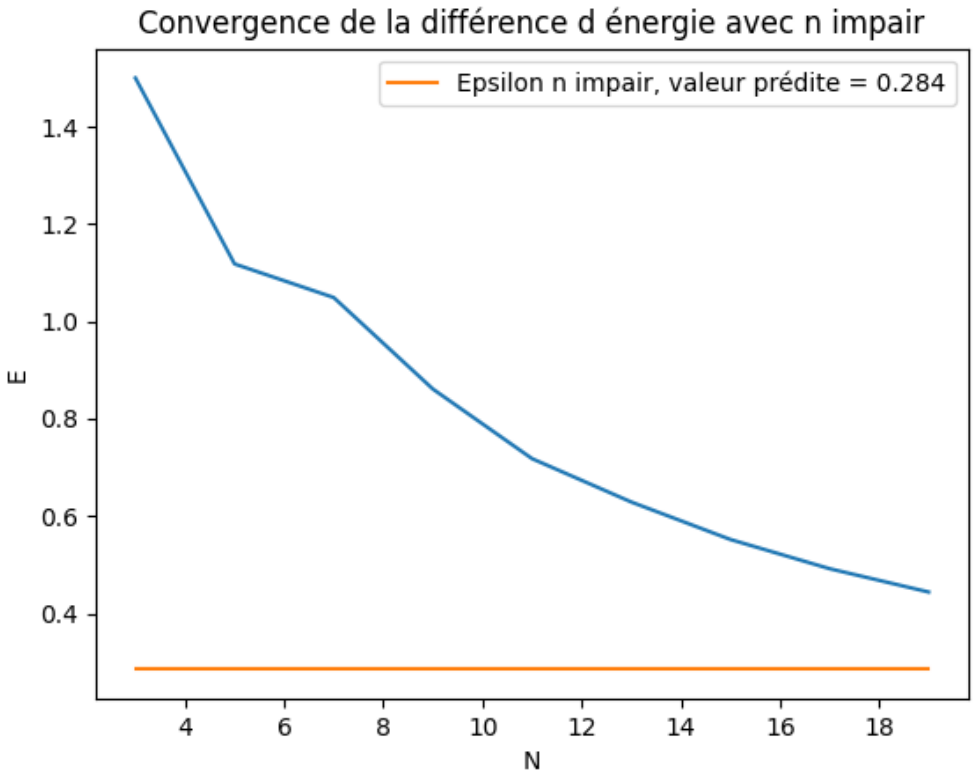
Convergence de la différence d énergie avec n impair
Epsilon n impair, valeur prédite = 0.284
E
N
1.4
1.2
1.0
0.8
0.6
0.4
4
6
8
10
12
14
16
18

FIGURE 6 –