

Nom étudiant: Votre nom

Numéro étudiant: Numéro étudiant

**LU3PY126 FOAD**  
**TP 4 : Équation de Schrödinger**

---

## Table des matières

# Introduction

## Objectifs du TP :

- Résoudre l'équation de Schrödinger 1D par différences finies
- Diagonaliser des matrices tridiagonales
- Étudier trois potentiels : puits infini, harmonique, double puits
- Comparer solutions numériques et théoriques
- Analyser l'effet tunnel dans le double puits

## Organisation du code :

- Fichiers Python : .../python/q01\_q02.py à q14.py
- Graphiques : .../figures/\*.pdf
- Bibliothèques : numpy, scipy, matplotlib

## Méthode numérique :

On discrétise l'équation de Schrödinger stationnaire :

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

La dérivée seconde devient :

$$\frac{d^2\psi}{dx^2} \approx \frac{\psi_{i+1} - 2\psi_i + \psi_{i-1}}{\delta x^2}$$

On obtient un problème de valeurs propres :

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

avec  $\mathbf{H}$  matrice tridiagonale.

## Exécution :

`python run_all.py`

# Question 1 Définition des potentiels

## 1.1 Énoncé

Définir et tracer trois potentiels sur  $[-L/2, L/2]$  avec  $L = 5$ ,  $n = 100$  :

- Puits infini :  $V(x) = 0$
- Oscillateur harmonique :  $V(x) = \frac{1}{2}x^2$
- Double puits : polynôme degré 4 avec racines en  $r_1, r_2, r_3, r_4$

## 1.2 Code

```
1 # -*- coding: utf-8 -*-
2 """
3 TP4 - Questions 1 et 2 : Définition des potentiels et discréétisation
4
5 Question 1 : Écrire trois fonctions pour les potentiels (puits infini, harmonique, double puits)
6 Question 2 : Définir l'intervalle [-L/2, L/2], le discréétiser et tracer les trois potentiels
7 """
8
9 import numpy as np
10 import matplotlib.pyplot as plt
11 import os
12
13
```

```

14 def potentiel_puits(x):
15     """
16     Potentiel du puits carré infini : V(x) = 0 partout
17
18     Paramètres :
19         x : position (scalaire ou tableau numpy)
20
21     Retour :
22         V(x) = 0
23     """
24     return np.zeros_like(x)
25
26
27 def potentiel_harmonique(x):
28     """
29     Potentiel harmonique : V(x) = (1/2) m^-2 x^2
30
31     On prend    = 1/m et ^2/(2m) = 1, donc :
32     V(x) = (1/2) k x^2 avec k = 1
33
34     Paramètres :
35         x : position
36
37     Retour :
38         V(x) = (1/2) x^2
39     """

```

```

40     return 0.5 * x**2
41
42
43 def potentiel_double_puits(x, a=1.0, r1=-2.0, r2=-0.5, r3=0.5, r4=2.0):
44     """
45     Double puits :  $V(x) = a(x - r_1)(x - r_2)(x - r_3)(x - r_4)$ 
46
47     Polynôme de degré 4 dont les racines sont  $r_1, r_2, r_3, r_4$ .
48
49     Paramètres :
50         x : position
51         a : facteur multiplicatif
52         r1, r2, r3, r4 : racines du polynôme
53
54     Retour :
55          $V(x)$ 
56     """
57     return a * (x - r1) * (x - r2) * (x - r3) * (x - r4)
58
59
60 def main():
61     """
62     Programme principal - Questions 1 et 2
63     """
64     # Paramètres
65     L = 5.0

```

```

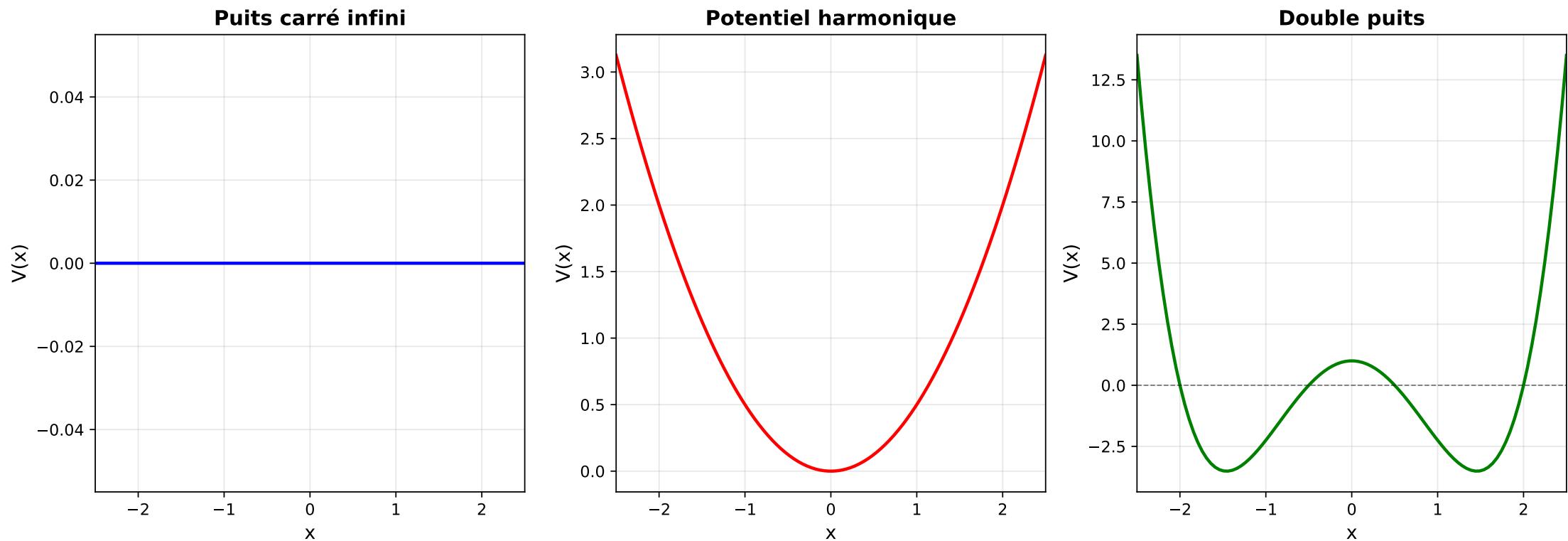
66     n = 100
67
68     # Discrétisation de l'intervalle [-L/2, L/2]
69     delta_x = L / (n - 1)
70     x = np.linspace(-L/2, L/2, n)
71
72     print("== TP4 - Questions 1 et 2 ==")
73     print(f"Paramètres :")
74     print(f"  L = {L}")
75     print(f"  n = {n}")
76     print(f"  x = {delta_x:.4f}")
77     print(f"  Intervalle : [{x[0]:.2f}, {x[-1]:.2f}]")
78
79     # Calcul des trois potentiels
80     V_puits = potentiel_puits(x)
81     V_harmonique = potentiel_harmonique(x)
82     V_double_puits = potentiel_double_puits(x, a=1.0, r1=-2.0, r2=-0.5, r3=0.5, r4=2.0)
83
84     # Tracer les trois potentiels
85     plt.figure(figsize=(14, 5))
86
87     # Puits infini
88     plt.subplot(1, 3, 1)
89     plt.plot(x, V_puits, 'b-', linewidth=2)
90     plt.xlabel('x', fontsize=12)
91     plt.ylabel('V(x)', fontsize=12)

```

```
92 plt.title('Puits carré infini', fontsize=13, fontweight='bold')
93 plt.grid(True, alpha=0.3)
94 plt.xlim(-L/2, L/2)
95
96 # Potentiel harmonique
97 plt.subplot(1, 3, 2)
98 plt.plot(x, V_harmonique, 'r-', linewidth=2)
99 plt.xlabel('x', fontsize=12)
100 plt.ylabel('V(x)', fontsize=12)
101 plt.title('Potentiel harmonique', fontsize=13, fontweight='bold')
102 plt.grid(True, alpha=0.3)
103 plt.xlim(-L/2, L/2)
104
105 # Double puits
106 plt.subplot(1, 3, 3)
107 plt.plot(x, V_double_puits, 'g-', linewidth=2)
108 plt.axhline(y=0, color='k', linestyle='--', linewidth=0.8, alpha=0.5)
109 plt.xlabel('x', fontsize=12)
110 plt.ylabel('V(x)', fontsize=12)
111 plt.title('Double puits', fontsize=13, fontweight='bold')
112 plt.grid(True, alpha=0.3)
113 plt.xlim(-L/2, L/2)
114
115 plt.tight_layout()
116
117 # Sauvegarde
```

```
118     output_dir = os.path.join(os.path.dirname(__file__), '..', 'figures')
119     os.makedirs(output_dir, exist_ok=True)
120     output_path = os.path.join(output_dir, 'q01_q02.pdf')
121     plt.savefig(output_path, dpi=150)
122     print(f"\nFigure sauvegardée : {output_path}")
123
124     plt.show()
125
126     print("\n==== Interprétation ===")
127     print("• Puits infini :  $V(x) = 0$  partout (particule libre dans la boîte)")
128     print("• Potentiel harmonique :  $V(x) = (1/2)x^2$  (oscillateur quantique)")
129     print("• Double puits : polynôme de degré 4 avec deux minimums symétriques")
130
131
132 if __name__ == "__main__":
133     main()
```

## 1.3 Résultats



### Commentaire :

Les trois potentiels sont correctement définis. Le double puits présente deux minima symétriques séparés par une barrière centrale.

# Question 2 Conditions aux limites

## 2.1 Énoncé

Interpréter physiquement les conditions  $\psi(-L/2) = \psi(L/2) = 0$ .

## 2.2 Code

```
1 # -*- coding: utf-8 -*-
2 """
3 TP4 - Question 3 : Interprétation physique des conditions aux limites
4
5 Question : Quel sens physique cela a-t-il de poser (-1) = (n) = 0 ?
6 Quelle est la conséquence pour le potentiel effectif ?
7 """
8
9
10 def main():
11     """
12     Programme principal - Question 3
13     """
14     print("== TP4 - Question 3 : Conditions aux limites ==\n")
15
16     print("Question : Quel sens physique a la condition (-1) = (n) = 0 ?")
17     print("Quelle conséquence pour le potentiel effectif ?\n")
```

```

18
19     print("Réponse :")
20     print(" " * 70)
21     print("• Poser (-1) = (n) = 0 signifie que la fonction d'onde s'annule")
22     print(" aux extrémités de l'intervalle [-L/2, L/2].")
23     print()
24     print("• Sens physique :")
25     print(" - La particule ne peut pas exister hors de cet intervalle")
26     print(" - On simule un potentiel infini aux bords : V(±L/2) = +")
27     print(" - C'est comme si la particule était confinée dans une \"boîte\"")
28     print()
29     print("• Conséquence pour le potentiel effectif :")
30     print(" - Quel que soit le potentiel V(x) défini dans [-L/2, L/2],")
31     print(" on ajoute implicitement des \"murs infinis\" aux extrémités")
32     print(" - Le potentiel effectif devient :")
33     print("     V_eff(x) = V(x)      si x ∈ [-L/2, L/2]")
34     print("     V_eff(x) = +      sinon")
35     print()
36     print("• Condition de validité :")
37     print(" - Il faut que L soit suffisamment grand pour que la fonction")
38     print(" d'onde soit négligeable aux bords : (±L/2) ≈ 0")
39     print(" - Si ce n'est pas le cas, les résultats sont faussés car la")
40     print("     particule \"sent\" les murs artificiels")
41     print(" " * 70)
42
43

```

```
44 if __name__ == "__main__":
45     main()
```

## 2.3 Discussion

Les conditions  $\psi(\pm L/2) = 0$  signifient que la particule ne peut pas exister aux bords du domaine. Cela équivaut à placer des murs infinis ( $V \rightarrow \infty$ ) en  $x = \pm L/2$ .

**Conséquences :**

- La particule est confinée dans  $[-L/2, L/2]$
- États liés uniquement
- Spectre discret d'énergies
- Impact sur la précision : si  $L$  est trop petit, les fonctions d'onde sont tronquées artificiellement

# Question 3 Construction et diagonalisation de l'hamiltonien

## 3.1 Énoncé

**Q4** : Construire la matrice hamiltonienne tridiagonale.

**Q5** : Diagonaliser avec `scipy.linalg.eigh_tridiagonal`.

## 3.2 Code

```
1 # -*- coding: utf-8 -*-
2 """
3 TP4 - Questions 4 et 5 : Diagonalisation de la matrice Hamiltonienne
4
5 Question 4 : Créer les tableaux d et e pour la matrice tridiagonale H
6 Question 5 : Utiliser eigh_tridiagonal pour trouver valeurs et vecteurs propres
7 """
8
9 import numpy as np
10 from scipy.linalg import eigh_tridiagonal
11 import os
12 import sys
13
14 sys.path.append(os.path.dirname(__file__))
15 from q01_q02 import potentiel_puits
16
```

```

17
18 def construire_hamiltonien(x, V, delta_x):
19     """
20         Construit les diagonales d et e de la matrice Hamiltonienne tridiagonale.
21
22     La matrice H a la forme :
23         H_ii = 2/x2 + V_i (diagonale principale)
24         H_i(i±1) = -1/x2 (diagonales adjacentes)
25
26     Paramètres :
27         x : tableau des positions
28         V : tableau des valeurs du potentiel
29         delta_x : pas de discrétisation
30
31     Retour :
32         d : diagonale principale (taille n)
33         e : diagonale supérieure (taille n-1)
34     """
35     n = len(x)
36
37     # Diagonale principale : 2/x2 + V_i
38     d = 2.0 / delta_x**2 + V
39
40     # Diagonales adjacentes : -1/x2
41     e = -np.ones(n - 1) / delta_x**2
42

```

```
43     return d, e
44
45
46 def diagonaliser_hamiltonien(d, e):
47     """
48     Diagonalise la matrice Hamiltonienne tridiagonale.
49
50     Paramètres :
51         d : diagonale principale
52         e : diagonale supérieure
53
54     Retour :
55         w : valeurs propres (énergies), ordre croissant
56         v : vecteurs propres (fonctions d'onde), en colonnes
57     """
58     w, v = eigh_tridiagonal(d, e)
59     return w, v
60
61
62 def main():
63     """
64     Programme principal - Questions 4 et 5
65     """
66     # Paramètres
67     L = 5.0
68     n = 100
```

```

69
70     # Discrétisation
71     delta_x = L / (n - 1)
72     x = np.linspace(-L/2, L/2, n)
73
74     print("== TP4 - Questions 4 et 5 ==")
75     print(f"Paramètres :")
76     print(f"  L = {L}")
77     print(f"  n = {n}")
78     print(f"  x = {delta_x:.4f}")
79
80     # Potentiel du puits infini (V = 0 partout)
81     V = potentiel_puits(x)
82
83     print(f"\n== Question 4 : Construction de la matrice H ==")
84
85     # Construire les diagonales de H
86     d, e = construire_hamiltonien(x, V, delta_x)
87
88     print(f"Taille de la diagonale principale d : {len(d)}")
89     print(f"Taille de la diagonale supérieure e : {len(e)}")
90     print(f"Premiers éléments de d : {d[:5]}")
91     print(f"Premiers éléments de e : {e[:5]}")
92
93     print(f"\n== Question 5 : Diagonalisation ==")
94

```

```
95 # Diagonaliser
96 w, v = diagonaliser_hamiltonien(d, e)
97
98 print(f"Nombre de valeurs propres trouvées : {len(w)}")
99 print(f"Forme de la matrice des vecteurs propres : {v.shape}")
100 print(f"\nPremières énergies propres (10 premières) :")
101 for i in range(min(10, len(w))):
102     print(f"  E[{i}] = {w[i]:.6f}")
103
104 print(f"\nÉnergies minimale et maximale :")
105 print(f"  E_min = {w[0]:.6f}")
106 print(f"  E_max = {w[-1]:.6f}")
107
108
109 if __name__ == "__main__":
110     main()
```

### 3.3 Méthode

L'hamiltonien est une matrice tridiagonale  $(n - 2) \times (n - 2)$  :

- Diagonale :  $d_i = \frac{2}{\delta x^2} + V_i$
- Sous/sur-diagonale :  $e_i = -\frac{1}{\delta x^2}$

La fonction `scipy.linalg.eigh_tridiagonal` exploite la structure tridiagonale pour une diagonalisation efficace en  $O(n^2)$  au lieu de  $O(n^3)$ .

# Question 4 Puits infini : comparaison des énergies

## 4.1 Énoncé

Comparer les énergies numériques avec la solution théorique :

$$E_p = \frac{\pi^2(p+1)^2}{L^2}$$

## 4.2 Code

```
1 # -*- coding: utf-8 -*-
2 """
3 TP4 - Question 6 : Comparaison des énergies propres avec la théorie (puits infini)
4
5 Tracer les énergies numériques et théoriques pour le puits carré infini.
6 """
7
8 import numpy as np
9 import matplotlib.pyplot as plt
10 import os
11 import sys
12
13 sys.path.append(os.path.dirname(__file__))
14 from q01_q02 import potentiel_puits
```

```

15 from q04_q05 import construire_hamiltonien, diagonaliser_hamiltonien
16
17
18 def energie_theorique_puits(p, L):
19     """
20         Énergie théorique pour le puits carré infini.
21
22          $E_p = \frac{\pi^2(p+1)^2}{2mL^2}$ 
23
24         Avec  $\frac{p^2}{2m} = 1$ , on a :
25          $E_p = \frac{\pi^2(p+1)^2}{L^2}$ 
26
27         Paramètres :
28             p : numéro du niveau (p = 0, 1, 2, ...)
29             L : largeur du puits
30
31         Retour :
32             E_p
33         """
34     return np.pi * (p + 1) / L**2
35
36
37 def main():
38     """
39         Programme principal - Question 6
40     """

```

```

41 # Paramètres
42 L = 5.0
43 n = 100
44
45 # Discrétisation
46 delta_x = L / (n - 1)
47 x = np.linspace(-L/2, L/2, n)
48
49 print("== TP4 - Question 6 : Puits carré infini ==")
50 print(f"Paramètres : L = {L}, n = {n}\n")
51
52 # Potentiel
53 V = potentiel_puits(x)
54
55 # Construire et diagonaliser H
56 d, e = construire_hamiltonien(x, V, delta_x)
57 w, v = diagonaliser_hamiltonien(d, e)
58
59 # Énergies théoriques
60 p_values = np.arange(len(w))
61 E_theo = energie_theorique_puits(p_values, L)
62
63 # Comparaison pour les premiers niveaux
64 print("Comparaison énergies numériques vs théoriques :")
65 print(f"{'p':<5} {'E_num':<15} {'E_theo':<15} {'Écart relatif':<15}")
66 print("-" * 55)

```

```

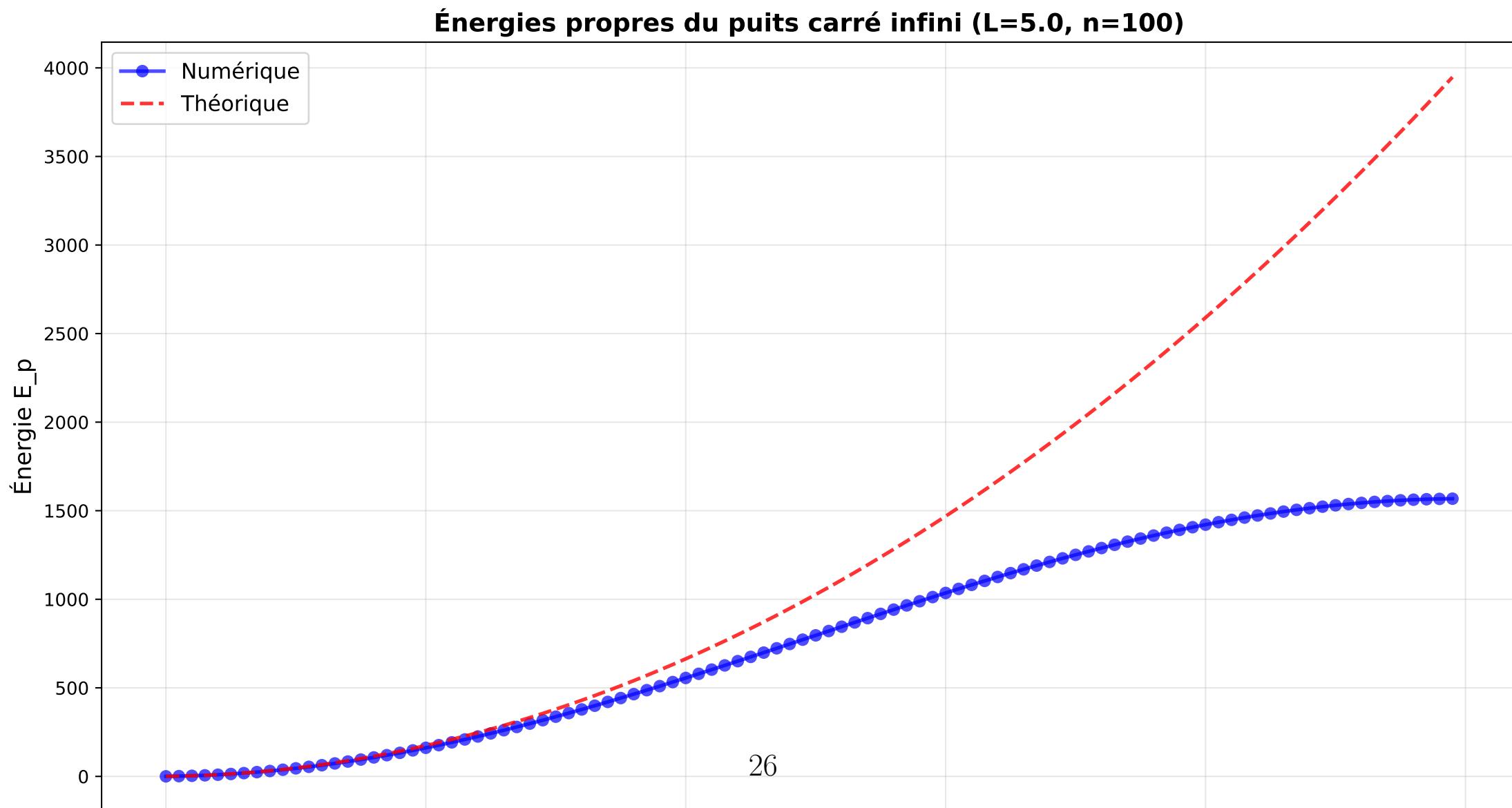
67     for p in range(min(10, len(w))):
68         ecart_rel = abs(w[p] - E_theo[p]) / E_theo[p] * 100
69         print(f"{p:<5} {w[p]:<15.6f} {E_theo[p]:<15.6f} {ecart_rel:<15.4f}%")
70
71 # Graphique
72 plt.figure(figsize=(12, 7))
73
74 plt.plot(p_values, w, 'bo-', linewidth=2, markersize=6, label='Numérique', alpha=0.7)
75 plt.plot(p_values, E_theo, 'r--', linewidth=2, label='Théorique', alpha=0.8)
76
77 plt.xlabel('Niveau p', fontsize=13)
78 plt.ylabel('Énergie E_p', fontsize=13)
79 plt.title(f'Énergies propres du puits carré infini (L={L}, n={n})',
80           fontsize=14, fontweight='bold')
81 plt.legend(fontsize=12)
82 plt.grid(True, alpha=0.3)
83 plt.tight_layout()
84
85 # Sauvegarde
86 output_dir = os.path.join(os.path.dirname(__file__), '..', 'figures')
87 os.makedirs(output_dir, exist_ok=True)
88 output_path = os.path.join(output_dir, f'q06_puits_energies_n{n}_L{int(L)}.pdf')
89 plt.savefig(output_path, dpi=150)
90 print(f"\nFigure sauvegardée : {output_path}")
91
92 plt.show()

```

```
93  
94     print("\n==== Discussion ===")  
95     print("• L'accord est excellent pour les bas niveaux d'énergie.")  
96     print("• Pour les niveaux élevés, l'écart augmente car la fonction d'onde")  
97     print(" oscille rapidement et n'est plus bien échantillonnée avec x fixe.")  
98     print("• Pour améliorer : augmenter n (diminuer x).")  
99  
100  
101 if __name__ == "__main__":  
102     main()
```



## 4.3 Résultats



**Commentaire :**

L'accord est excellent pour les bas niveaux. Pour  $p$  élevé, l'écart augmente car la longueur d'onde devient comparable à  $\delta x$ , violant la condition de discrétisation fine.

# Question 5 Normalisation et visualisation des fonctions d'onde

## 5.1 Énoncé

Normaliser les fonctions d'onde et tracer  $|\psi_p(x)|^2$  décalées de  $E_p$  pour  $p = 0, 1, 2$ .

## 5.2 Code

```
1 # -*- coding: utf-8 -*-
2 """
3 TP4 - Question 7 : Normalisation et visualisation des fonctions d'onde (puits infini)
4
5 Normaliser les fonctions d'onde et tracer ||² décalées verticalement par E_p.
6 """
7
8 import numpy as np
9 import matplotlib.pyplot as plt
10 import os
11 import sys
12
13 sys.path.append(os.path.dirname(__file__))
14 from q01_q02 import potentiel_puits
15 from q04_q05 import construire_hamiltonien, diagonaliser_hamiltonien
16
17
```

```

18 def normalize(v, delta_x):
19     """
20     Normalise les vecteurs propres (fonctions d'onde).
21
22     Les colonnes de v sont déjà normalisées par eigh_tridiagonal,
23     mais on applique le facteur 1/(x) pour respecter la normalisation continue.
24
25     Paramètres :
26         v : matrice des vecteurs propres (n × n)
27         delta_x : pas de discrétisation
28
29     Retour :
30         v_norm : vecteurs propres normalisés
31     """
32     v_norm = v / np.linalg.norm(v, axis=0) # Normaliser les colonnes
33     v_norm /= np.sqrt(delta_x)           # Facteur 1/(x)
34     return v_norm
35
36
37 def main():
38     """
39     Programme principal - Question 7
40     """
41     # Paramètres
42     L = 5.0
43     n = 100

```

```

44
45     # Discrétisation
46     delta_x = L / (n - 1)
47     x = np.linspace(-L/2, L/2, n)
48
49     print("== TP4 - Question 7 : Fonctions d'onde du puits infini ==")
50     print(f"Paramètres : L = {L}, n = {n}\n")
51
52     # Potentiel
53     V = potentiel_puits(x)
54
55     # Diagonaliser
56     d, e = construire_hamiltonien(x, V, delta_x)
57     w, v = diagonaliser_hamiltonien(d, e)
58
59     # Normaliser
60     v_norm = normalize(v, delta_x)
61
62     print("Normalisation vérifiée :")
63     for p in range(3):
64         integrale = np.sum(v_norm[:, p]**2) * delta_x
65         print(f" |_{p}|² dx = {integrale:.6f}")
66
67     # Tracer les fonctions d'onde au carré, décalées par E_p
68     plt.figure(figsize=(12, 8))
69

```

```

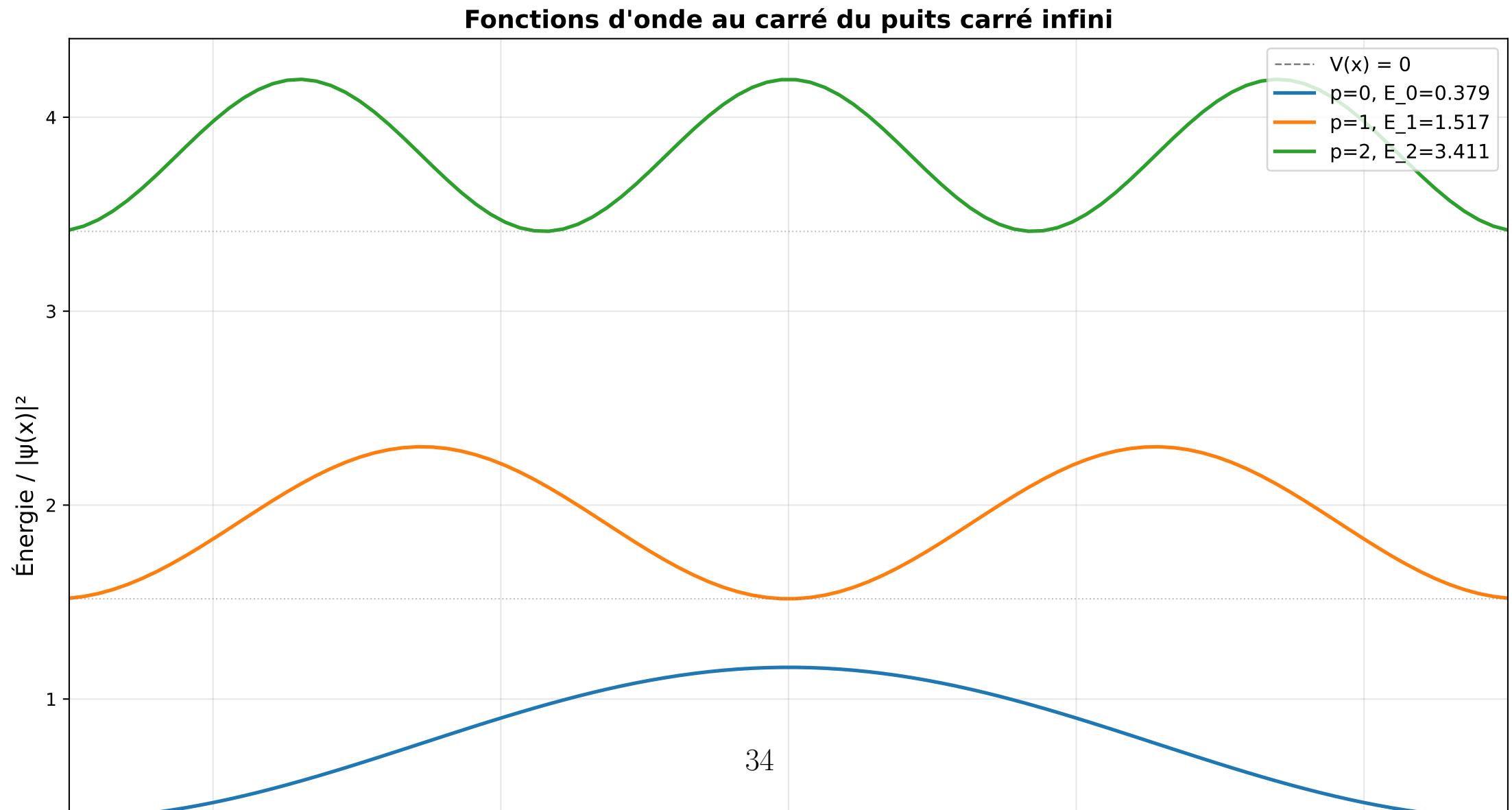
70 # Tracer le potentiel ( $V = 0$ )
71 plt.axhline(y=0, color='k', linestyle='--', linewidth=1, alpha=0.5, label=' $V(x) = 0$ ')
72
73 # Tracer les 3 premières fonctions d'onde
74 p_max = 3
75 for p in range(p_max):
76     psi_squared = v_norm[:, p]**2
77     # Décaler verticalement par  $E_p$ 
78     plt.plot(x, w[p] + psi_squared * 2, linewidth=2, label=f' $p={p}$ ,  $E_p={w[p]:.3f}$ ')
79     # Ligne horizontale pour  $E_p$ 
80     plt.axhline(y=w[p], color='gray', linestyle=':', linewidth=0.8, alpha=0.5)
81
82 plt.xlabel('x', fontsize=13)
83 plt.ylabel('Énergie /  $|(\psi)|^2$ ', fontsize=13)
84 plt.title('Fonctions d\'onde au carré du puits carré infini',
85            fontsize=14, fontweight='bold')
86 plt.legend(fontsize=11, loc='upper right')
87 plt.grid(True, alpha=0.3)
88 plt.xlim(-L/2, L/2)
89 plt.tight_layout()
90
91 # Sauvegarde
92 output_dir = os.path.join(os.path.dirname(__file__), '..', 'figures')
93 output_path = os.path.join(output_dir, f'q07_puits_fonctions_n{n}_L{int(L)}.pdf')
94 plt.savefig(output_path, dpi=150)
95 print(f"\nFigure sauvegardée : {output_path}")

```

```
96  
97     plt.show()  
98  
99     print("\n==== Discussion ===")  
100    print("• p=0 : fonction d'onde fondamentale, une seule bosse (pas de nœud)")  
101    print("• p=1 : premier état excité, deux bosses (un nœud au centre)")  
102    print("• p=2 : deuxième état excité, trois bosses (deux nœuds)")  
103    print("• Chaque état a (p+1) bosses et p nœuds, comme attendu.")  
104    print("• La fonction d'onde s'annule bien aux extrémités de l'intervalle.")  
105  
106  
107 if __name__ == "__main__":  
108     main()
```



## 5.3 Résultats



### **Commentaire :**

Les densités de probabilité montrent :

- $p = 0$  : pas de nœud, maximum au centre
- $p = 1$  : un nœud au centre
- $p = 2$  : deux nœuds

Le nombre de nœuds est égal à  $p$  (nombre quantique).

# Question 6 Comparaison détaillée numérique vs théorique

## 6.1 Énoncé

Comparer  $\psi_{num}$  et  $\psi_{theo}$  pour  $p = 1$  (bon accord) et  $p = 55$  (mauvais accord).

## 6.2 Code

```
1 # -*- coding: utf-8 -*-
2 """
3 TP4 - Question 8 : Comparaison fonctions d'onde numériques vs théoriques
4
5 Comparer _num avec _theo pour p=1 et p=55.
6 """
7
8 import numpy as np
9 import matplotlib.pyplot as plt
10 import os
11 import sys
12
13 sys.path.append(os.path.dirname(__file__))
14 from q01_q02 import potentiel_puits
15 from q04_q05 import construire_hamiltonien, diagonaliser_hamiltonien
16 from q07 import normalize
17
```

```

18
19 def fonction onde theorique_puits(x, p, L):
20     """
21         Fonction d'onde théorique pour le puits carré infini.
22
23     _p(x) = (2/L) sin((p+1)(x + L/2) / L)
24
25     Paramètres :
26         x : positions
27         p : numéro du niveau
28         L : largeur du puits
29
30     Retour :
31         _p(x)
32     """
33     return np.sqrt(2 / L) * np.sin(np.pi * (p + 1) * (x + L/2) / L)
34
35
36 def main():
37     """
38         Programme principal - Question 8
39     """
40     # Paramètres
41     L = 5.0
42     n = 100
43

```

```

44 # Discrétisation
45 delta_x = L / (n - 1)
46 x = np.linspace(-L/2, L/2, n)
47
48 print("== TP4 - Question 8 : Comparaison _num vs _theo ==")
49 print(f"Paramètres : L = {L}, n = {n}\n")
50
51 # Potentiel et diagonalisation
52 V = potentiel_puits(x)
53 d, e = construire_hamiltonien(x, V, delta_x)
54 w, v = diagonaliser_hamiltonien(d, e)
55 v_norm = normalize(v, delta_x)
56
57 # Niveaux à comparer
58 p_values = [1, 55]
59
60 for p in p_values:
61     print(f"\n== Niveau p = {p} ==")
62
63     # Fonctions d'onde
64     psi_num = v_norm[:, p]
65     psi_theo = fonction onde theorique_puits(x, p, L)
66
67     # Vérifier que le signe est cohérent (phase arbitraire)
68     if np.dot(psi_num, psi_theo) < 0:
69         psi_num = -psi_num

```

```

70
71     # Erreur relative
72     erreur = np.abs(psi_num - psi_theo)
73     erreur_max = np.max(erreur)
74     erreur_moy = np.mean(erreur)
75
76     print(f"Erreur max : {erreur_max:.6f}")
77     print(f"Erreur moyenne : {erreur_moy:.6f}")
78
79     # Graphique
80     fig, axes = plt.subplots(2, 1, figsize=(12, 10))
81
82     # Graphe 1 : (x)
83     axes[0].plot(x, psi_num, 'b-', linewidth=2, label='Numérique', alpha=0.7)
84     axes[0].plot(x, psi_theo, 'r--', linewidth=2, label='Théorique', alpha=0.8)
85     axes[0].set_xlabel('x', fontsize=12)
86     axes[0].set_ylabel('(x)', fontsize=12)
87     axes[0].set_title(f'Fonction d\'onde pour p={p} (E={w[p]:.3f})',
88                      fontsize=13, fontweight='bold')
89     axes[0].legend(fontsize=11)
90     axes[0].grid(True, alpha=0.3)
91     axes[0].set_xlim(-L/2, L/2)
92
93     # Graphe 2 : erreur |_num - _theo|
94     axes[1].plot(x, erreur, 'g-', linewidth=2)
95     axes[1].set_xlabel('x', fontsize=12)

```

```

96     axes[1].set_ylabel('|_num - _theo|', fontsize=12)
97     axes[1].set_title(f'Erreur absolue (max={erreur_max:.6f})',
98                         fontsize=13, fontweight='bold')
99     axes[1].grid(True, alpha=0.3)
100    axes[1].set_xlim(-L/2, L/2)
101
102    plt.tight_layout()
103
104    # Sauvegarde
105    output_dir = os.path.join(os.path.dirname(__file__), '..', 'figures')
106    output_path = os.path.join(output_dir, f'q08_puits_comparaison_p{p}_n{n}_L{int(L)}.pdf')
107    plt.savefig(output_path, dpi=150)
108    print(f"Figure sauvegardée : {output_path}")
109
110    plt.show()
111
112    print("\n==== Diagnostic ===")
113    print("• Pour p=1 : accord excellent, erreur négligeable")
114    print("→ la fonction d'onde varie lentement, bien échantillonnée par x")
115    print("• Pour p=55 : erreur importante, oscillations mal capturées")
116    print("→ la fonction d'onde oscille rapidement, x trop grand")
117    print("→ on atteint la limite de résolution : L/(p+1) x")
118    print("• Solution : augmenter n pour diminuer x")
119
120
121 if __name__ == "__main__":

```

122

main()

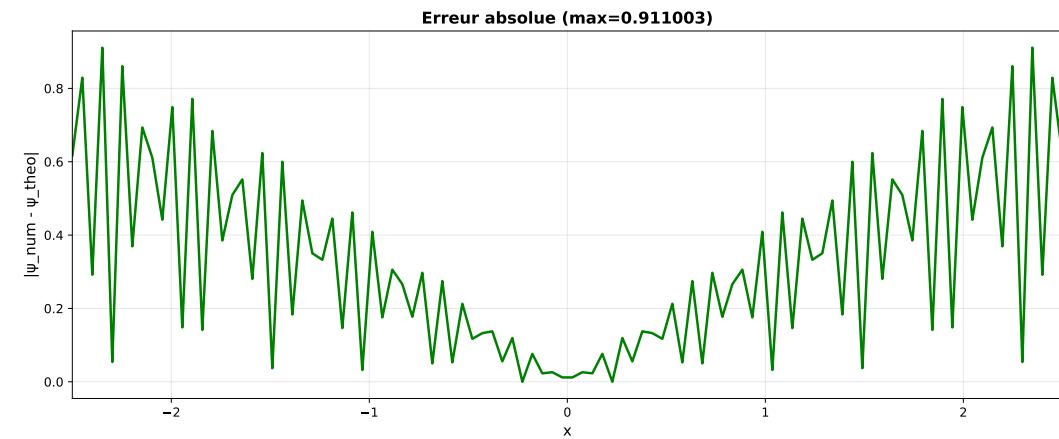
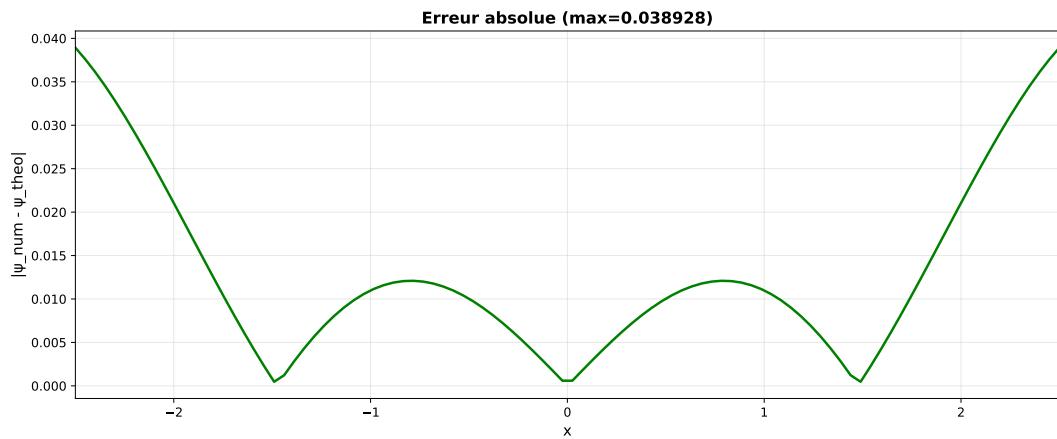
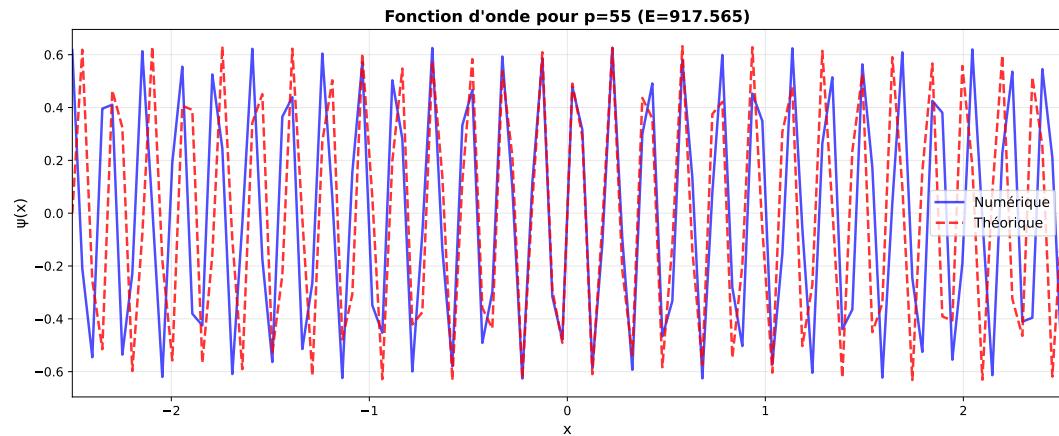
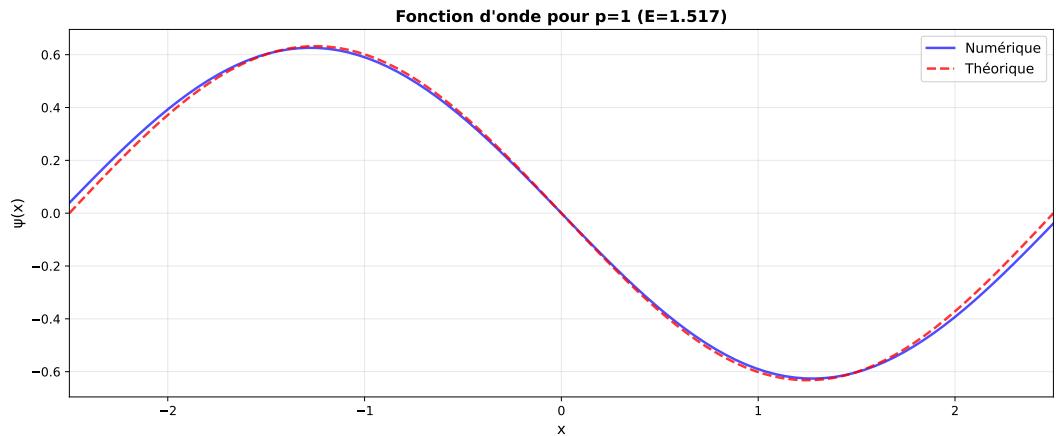
## 6.3 Résultats

### Cas $p = 1$ :

Accord excellent. La fonction d'onde a une longueur d'onde grande devant  $\delta x$ .

### Cas $p = 55$ :

Accord médiocre. Les oscillations rapides ne sont pas résolues : il faut au moins quelques points par oscillation. Augmenter  $n$  résoudrait le problème.



# Question 7 Oscillateur harmonique : L=5

## 7.1 Énoncé

Étudier  $V(x) = \frac{1}{2}x^2$  avec  $L = 5$ ,  $n = 100$ . Comparer avec :

$$E_p = \left(p + \frac{1}{2}\right)\sqrt{2}$$

## 7.2 Code

```
1 # -*- coding: utf-8 -*-
2 """
3 TP4 - Question 9 : Potentiel harmonique - Comparaison énergies avec théorie
4
5 Étudier le potentiel harmonique avec L=5, n=100 et comparer avec la théorie.
6 """
7
8 import numpy as np
9 import matplotlib.pyplot as plt
10 import os
11 import sys
12
13 sys.path.append(os.path.dirname(__file__))
14 from q01_q02 import potentiel_harmonique
```

```

15 from q04_q05 import construire_hamiltonien, diagonaliser_hamiltonien
16
17
18 def energie_theorique_harmonique(p):
19     """
20         Énergie théorique pour l'oscillateur harmonique.
21
22         E_p = (p + 1/2)
23
24         Avec k=1, m=1, =(k/m)=1, ^2/(2m)=1, on a  = 2.
25
26         Donc : E_p = (p + 1/2) 2
27
28         Paramètres :
29             p : numéro du niveau (p = 0, 1, 2, ...)
30
31         Retour :
32             E_p
33         """
34     return (p + 0.5) * np.sqrt(2)
35
36
37 def main():
38     """
39         Programme principal - Question 9
40     """

```

```

41 # Paramètres
42 L = 5.0
43 n = 100
44
45 # Discrétisation
46 delta_x = L / (n - 1)
47 x = np.linspace(-L/2, L/2, n)
48
49 print("== TP4 - Question 9 : Oscillateur harmonique ==")
50 print(f"Paramètres : L = {L}, n = {n}")
51 print(f"Potentiel : V(x) = (1/2)x²")
52 print(f" = 2 {np.sqrt(2):.6f}\n")
53
54 # Potentiel harmonique
55 V = potentiel_harmonique(x)
56
57 # Diagonaliser
58 d, e = construire_hamiltonien(x, V, delta_x)
59 w, v = diagonaliser_hamiltonien(d, e)
60
61 # Énergies théoriques
62 p_values = np.arange(min(50, len(w)))
63 E_theo = energie_theorique_harmonique(p_values)
64
65 # Comparaison
66 print("Comparaison énergies numériques vs théoriques :")

```

```

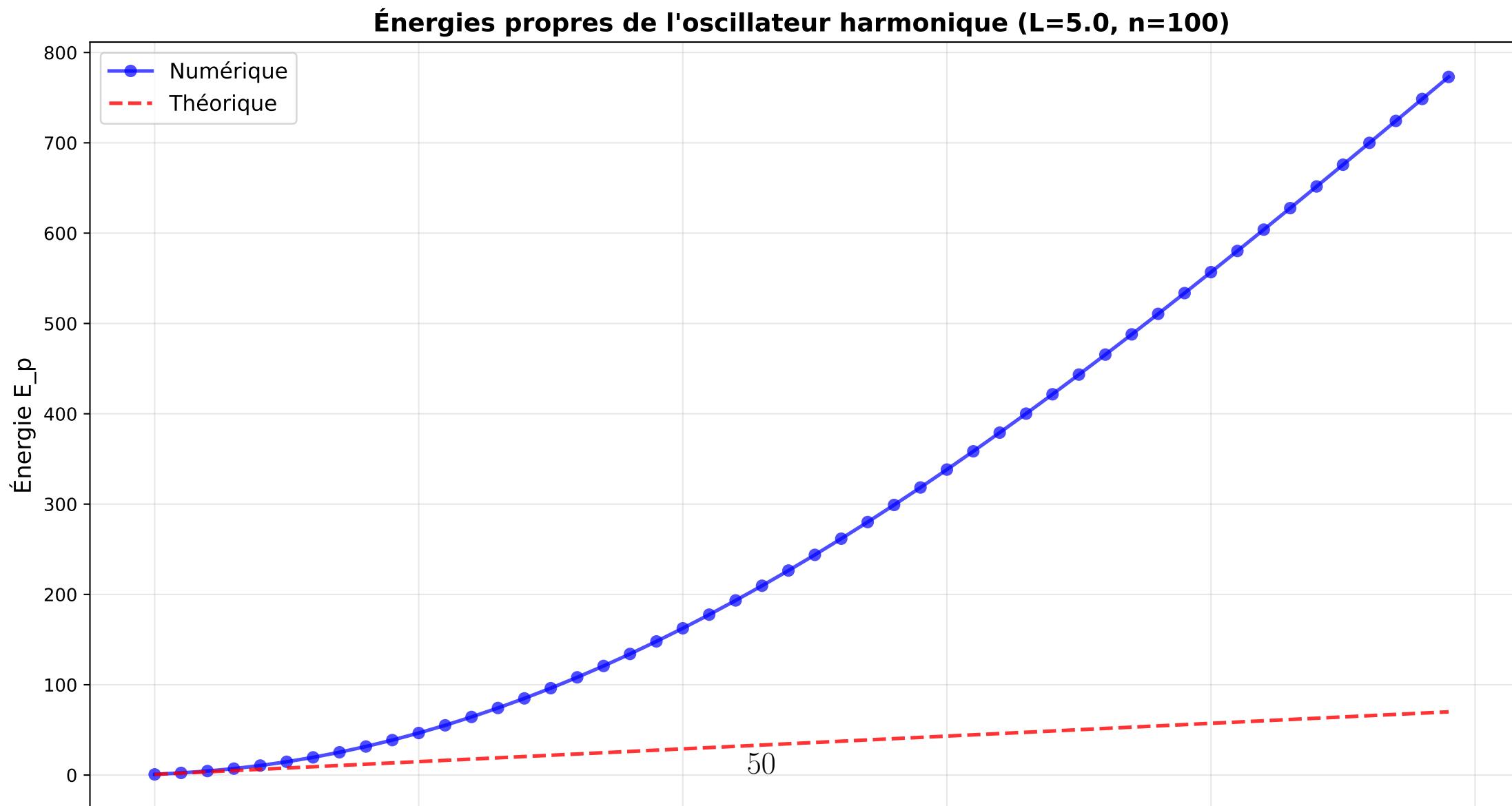
67 print(f"{'p':<5} {'E_num':<15} {'E_theo':<15} {'Écart relatif':<15}")
68 print("-" * 55)
69 for p in range(min(10, len(p_values))):
70     ecart_rel = abs(w[p] - E_theo[p]) / E_theo[p] * 100
71     print(f"{p:<5} {w[p]:<15.6f} {E_theo[p]:<15.6f} {ecart_rel:<15.6f}%")
72
73 # Graphique
74 plt.figure(figsize=(12, 7))
75
76 plt.plot(p_values, w[:len(p_values)], 'bo-', linewidth=2, markersize=6,
77           label='Numérique', alpha=0.7)
78 plt.plot(p_values, E_theo, 'r--', linewidth=2, label='Théorique', alpha=0.8)
79
80 plt.xlabel('Niveau p', fontsize=13)
81 plt.ylabel('Énergie E_p', fontsize=13)
82 plt.title(f'Énergies propres de l\'oscillateur harmonique (L={L}, n={n})',
83           fontsize=14, fontweight='bold')
84 plt.legend(fontsize=12)
85 plt.grid(True, alpha=0.3)
86 plt.tight_layout()
87
88 # Sauvegarde
89 output_dir = os.path.join(os.path.dirname(__file__), '..', 'figures')
90 output_path = os.path.join(output_dir, f'q09_harmonique_energies_n{n}_L{int(L)}.pdf')
91 plt.savefig(output_path, dpi=150)
92 print(f"\nFigure sauvegardée : {output_path}")

```

```
93
94     plt.show()
95
96     print("\n==== Commentaire ===")
97     print("• L'accord est bon pour les bas niveaux d'énergie.")
98     print("• Pour les niveaux élevés, l'écart augmente significativement.")
99     print("• Raison : la fonction d'onde s'étend loin du centre pour E élevée")
100    print(" et est tronquée par les conditions aux limites ( $\pm L/2$ ) = 0.")
101    print("• L=5 est trop petit : la particule \"sent\" les murs artificiels.")
102
103
104 if __name__ == "__main__":
105     main()
```



## 7.3 Résultats



**Commentaire :**

L'accord est bon pour les bas niveaux mais se dégrade rapidement. Raison : pour  $E$  élevée, la fonction d'onde s'étend loin et est tronquée par les conditions  $\psi(\pm L/2) = 0$ . La particule "sent" les murs artificiels.

# Question 8 Oscillateur harmonique : L=20

## 8.1 Énoncé

Refaire Q9 avec  $L = 20$  pour étudier l'influence de  $L$ .

## 8.2 Code

```
1 # -*- coding: utf-8 -*-
2 """
3 TP4 - Question 10 : Potentiel harmonique avec L=20
4
5 Refaire Q9 avec L=20 pour étudier l'influence de la discrétisation spatiale.
6 """
7
8 import numpy as np
9 import matplotlib.pyplot as plt
10 import os
11 import sys
12
13 sys.path.append(os.path.dirname(__file__))
14 from q01_q02 import potentiel_harmonique
15 from q04_q05 import construire_hamiltonien, diagonaliser_hamiltonien
16 from q09 import energie_theorique_harmonique
17
```

```

18
19 def main():
20     """
21     Programme principal - Question 10
22     """
23     # Paramètres
24     L = 20.0
25     n = 100
26
27     # Discrétisation
28     delta_x = L / (n - 1)
29     x = np.linspace(-L/2, L/2, n)
30
31     print("== TP4 - Question 10 : Oscillateur harmonique avec L étendu ==")
32     print(f"Paramètres : L = {L}, n = {n}")
33     print(f"x = {delta_x:.6f}")
34     print(f"Potentiel : V(x) = (1/2)x²\n")
35
36     # Potentiel harmonique
37     V = potentiel_harmonique(x)
38
39     # Diagonaliser
40     d, e = construire_hamiltonien(x, V, delta_x)
41     w, v = diagonaliser_hamiltonien(d, e)
42
43     # Énergies théoriques

```

```

44 p_values = np.arange(min(50, len(w)))
45 E_theo = energie_theorique_harmonique(p_values)
46
47 # Comparaison
48 print("Comparaison énergies numériques vs théoriques :")
49 print(f"{'p':<5} {'E_num':<15} {'E_theo':<15} {'Écart relatif':<15}")
50 print("-" * 55)
51 for p in range(min(10, len(p_values))):
52     ecart_rel = abs(w[p] - E_theo[p]) / E_theo[p] * 100
53     print(f"{p:<5} {w[p]:<15.6f} {E_theo[p]:<15.6f} {ecart_rel:<15.6f}%")
54
55 # Graphique
56 plt.figure(figsize=(12, 7))
57
58 plt.plot(p_values, w[:len(p_values)], 'bo-', linewidth=2, markersize=6,
59           label='Numérique (L=20)', alpha=0.7)
60 plt.plot(p_values, E_theo, 'r--', linewidth=2, label='Théorique', alpha=0.8)
61
62 plt.xlabel('Niveau p', fontsize=13)
63 plt.ylabel('Énergie E_p', fontsize=13)
64 plt.title(f'Énergies propres de l\'oscillateur harmonique (L={int(L)}, n={n})',
65           fontsize=14, fontweight='bold')
66 plt.legend(fontsize=12)
67 plt.grid(True, alpha=0.3)
68 plt.tight_layout()
69

```

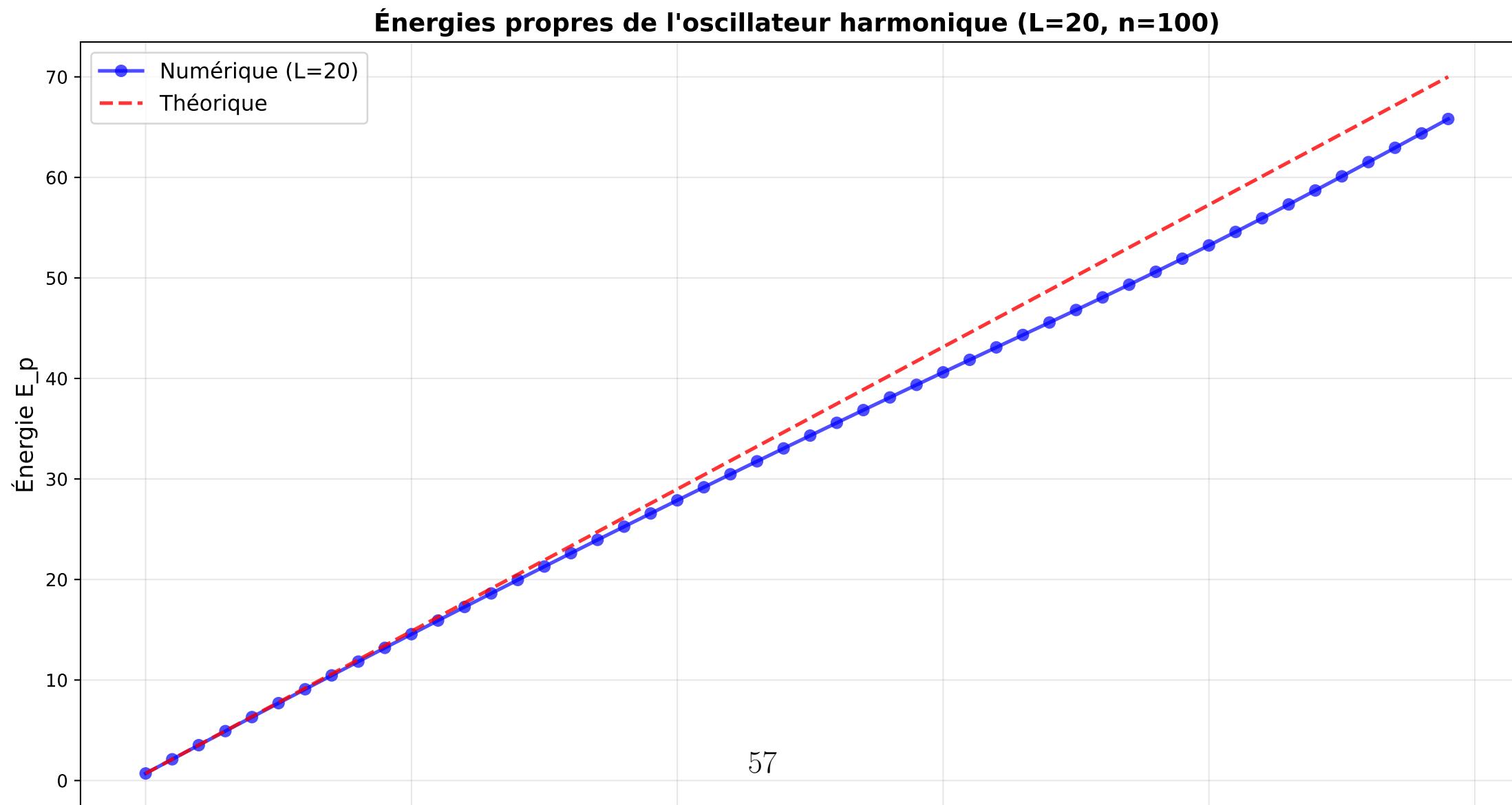
```

70 # Sauvegarde
71 output_dir = os.path.join(os.path.dirname(__file__), '..', 'figures')
72 output_path = os.path.join(output_dir, f'q10_harmonique_energies_n{n}_L{int(L)}.pdf')
73 plt.savefig(output_path, dpi=150)
74 print(f"\nFigure sauvegardée : {output_path}")
75
76 plt.show()
77
78 print("\n==== Commentaire ===")
79 print("• Avec L=20, l'accord est bien meilleur qu'avec L=5 (Q9).")
80 print("• Les conditions aux limites artificielles ((±L/2)=0) perturbent moins")
81 print(" car la fonction d'onde décroît exponentiellement loin du centre.")
82 print(f"• Inconvénient : x = {delta_x:.4f} est plus grand (n fixé)")
83 print(" donc la discrétisation spatiale est plus grossière.")
84 print("• Compromis : il faut augmenter n pour améliorer x tout en gardant L grand.")
85
86
87 if __name__ == "__main__":
88     main()

```



## 8.3 Résultats



**Commentaire :**

Avec  $L = 20$ , l'accord est bien meilleur. Les conditions aux limites perturbent moins car la fonction d'onde décroît exponentiellement loin du centre. Inconvénient :  $\delta x = L/(n - 1)$  augmente donc la discrétisation est plus grossière. Compromis : augmenter  $n$  en même temps que  $L$ .

# Question 9 Fonctions d'onde de l'oscillateur harmonique

## 9.1 Énoncé

Tracer  $V(x)$  et  $|\psi_p(x)|^2$  décalées de  $E_p$  pour  $p = 0, 1, 2$  avec  $L = 20$ ,  $n = 100$ .

## 9.2 Code

```
1 # -*- coding: utf-8 -*-
2 """
3 TP4 - Question 11 : Visualisation fonctions d'onde oscillateur harmonique
4
5 Tracer le potentiel V(x) et les densités de probabilité |_p(x)|² décalées de E_p
6 pour p = 0, 1, 2 avec L=20, n=100.
7 """
8
9 import numpy as np
10 import matplotlib.pyplot as plt
11 import os
12 import sys
13
14 sys.path.append(os.path.dirname(__file__))
15 from q01_q02 import potentiel_harmonique
16 from q04_q05 import construire_hamiltonien, diagonaliser_hamiltonien
17 from q07 import normalize
```

```
18
19
20 def main():
21     """
22     Programme principal - Question 11
23     """
24     # Paramètres
25     L = 20.0
26     n = 100
27
28     # Discrétisation
29     delta_x = L / (n - 1)
30     x = np.linspace(-L/2, L/2, n)
31
32     print("== TP4 - Question 11 : Fonctions d'onde oscillateur harmonique ==")
33     print(f"Paramètres : L = {L}, n = {n}")
34     print(f"x = {delta_x:.6f}\n")
35
36     # Potentiel harmonique
37     V = potentiel_harmonique(x)
38
39     # Diagonaliser
40     d, e = construire_hamiltonien(x, V, delta_x)
41     w, v = diagonaliser_hamiltonien(d, e)
42
43     # Normaliser
```

```

44 psi_norm = normalize(v, delta_x)
45
46 # Affichage énergies
47 print("Énergies des premiers niveaux :")
48 for p in range(min(5, len(w))):
49     print(f" E_{p} = {w[p]:.6f}")
50
51 # Graphique
52 plt.figure(figsize=(12, 8))
53
54 # Potentiel
55 plt.plot(x, V, 'k-', linewidth=2, label='V(x) = (1/2)x²', alpha=0.7)
56
57 # Fonctions d'onde p = 0, 1, 2
58 colors = ['blue', 'green', 'red']
59 for p in range(3):
60     psi_p = psi_norm[:, p]
61     rho_p = np.abs(psi_p)**2 # Densité de probabilité
62
63     # Décalage par E_p
64     plt.plot(x, w[p] + rho_p * 5, color=colors[p], linewidth=2,
65               label=f'|_p(x)|² + E_{p} (E={w[p]:.3f})', alpha=0.8)
66
67     # Ligne horizontale à E_p
68     plt.axhline(y=w[p], color=colors[p], linestyle='--', linewidth=1, alpha=0.4)
69

```

```

70 plt.xlabel('Position x', fontsize=13)
71 plt.ylabel('Énergie / Densité de probabilité', fontsize=13)
72 plt.title(f'Oscillateur harmonique : V(x) et |_p(x)|² (L={int(L)}, n={n})',
73           fontsize=14, fontweight='bold')
74 plt.legend(fontsize=11, loc='best')
75 plt.grid(True, alpha=0.3)
76 plt.xlim(-6, 6) # Focus sur région intéressante
77 plt.ylim(0, max(w[2] + 5*np.max(np.abs(psi_norm[:, 2])**2), np.max(V[abs(x) < 6])))
78 plt.tight_layout()

79
80 # Sauvegarde
81 output_dir = os.path.join(os.path.dirname(__file__), '..', 'figures')
82 output_path = os.path.join(output_dir, f'q11_harmonique_fonctions_n{n}_L{int(L)}.pdf')
83 plt.savefig(output_path, dpi=150)
84 print(f"\nFigure sauvegardée : {output_path}")

85
86 plt.show()

87
88 print("\n==== Commentaire ===")
89 print("• p=0 : État fondamental, densité maximale au centre ( gaussienne)")
90 print("• p=1 : Un nœud au centre, densité nulle en x=0")
91 print("• p=2 : Deux nœuds, densité maximale en trois lobes")
92 print("• Les fonctions d'onde s'étendent plus loin quand E augmente")
93 print("• Comportement classique : aux points de retour V(x) = E_p,")
94 print(" la densité quantique est maximale (vitesse classique nulle)")

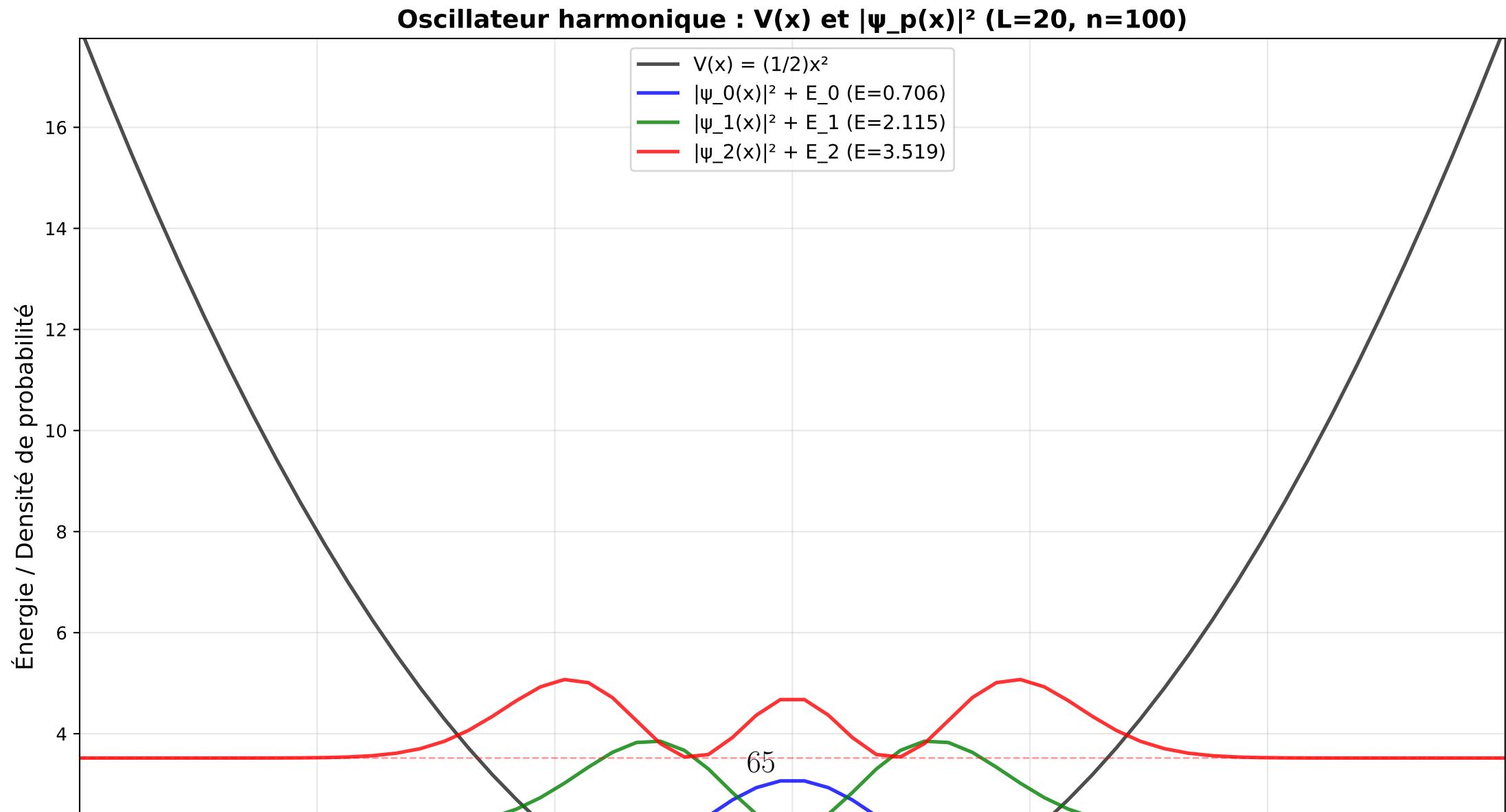
95

```

```
96  
97 if __name__ == "__main__":  
98     main()
```



### 9.3 Résultats



### **Commentaire :**

- $p = 0$  : gaussienne centrée (pas de nœud)
- $p = 1$  : un nœud au centre, densité nulle en  $x = 0$
- $p = 2$  : deux nœuds, trois lobes

Les maxima de densité correspondent aux points de retour classiques où  $V(x) = E_p$  (vitesse classique nulle).

# Question 10 Double puits symétrique ( $a=1$ )

## 10.1 Énoncé

Étudier le potentiel double puits avec  $a = 1$ ,  $r_1 = -2$ ,  $r_2 = -0.5$ ,  $r_3 = 0.5$ ,  $r_4 = 2$ ,  $L = 20$ ,  $n = 1000$ .

## 10.2 Code

```
1 # -*- coding: utf-8 -*-
2 """
3 TP4 - Question 12 : Double puits symétrique
4
5 Étudier le potentiel double puits symétrique avec a=1, r1=-2, r2=-0.5,
6 r3=0.5, r4=2, L=20, n=1000.
7 """
8
9 import numpy as np
10 import matplotlib.pyplot as plt
11 import os
12 import sys
13
14 sys.path.append(os.path.dirname(__file__))
15 from q01_q02 import potentiel_double_puits
16 from q04_q05 import construire_hamiltonien, diagonaliser_hamiltonien
17 from q07 import normalize
```

```
18
19
20 def main():
21     """
22     Programme principal - Question 12
23     """
24     # Paramètres
25     L = 20.0
26     n = 1000
27     a = 1.0
28     r1 = -2.0
29     r2 = -0.5
30     r3 = 0.5
31     r4 = 2.0
32
33     # Discrétisation
34     delta_x = L / (n - 1)
35     x = np.linspace(-L/2, L/2, n)
36
37     print("== TP4 - Question 12 : Double puits symétrique ==")
38     print(f"Paramètres : L = {L}, n = {n}")
39     print(f"x = {delta_x:.6f}")
40     print(f"Double puits : a = {a}, r1 = {r1}, r2 = {r2}, r3 = {r3}, r4 = {r4}\n")
41
42     # Potentiel double puits
43     V = potentiel_double_puits(x, a, r1, r2, r3, r4)
```

```

44
45     # Diagonaliser
46     d, e = construire_hamiltonien(x, V, delta_x)
47     w, v = diagonaliser_hamiltonien(d, e)
48
49     # Normaliser
50     psi_norm = normalize(v, delta_x)
51
52     # Affichage énergies
53     print("Énergies des premiers niveaux :")
54     for p in range(min(10, len(w))):
55         print(f"  E_{p} = {w[p]:.8f}")
56
57     print("\n==== Analyse des doublets ===")
58     print("Doublet 1 :")
59     print(f"  E_0 = {w[0]:.10f}")
60     print(f"  E_1 = {w[1]:.10f}")
61     print(f"  E = {w[1] - w[0]:.10e} (levée de dégénérescence)")
62
63     print("Doublet 2 :")
64     print(f"  E_2 = {w[2]:.10f}")
65     print(f"  E_3 = {w[3]:.10f}")
66     print(f"  E = {w[3] - w[2]:.10e}")
67
68     # Graphique 1 : Potentiel et énergies
69     fig, axes = plt.subplots(2, 1, figsize=(12, 10))

```

```

70
71 # Potentiel avec niveaux d'énergie
72 ax1 = axes[0]
73 ax1.plot(x, V, 'k-', linewidth=2, label='V(x)', alpha=0.7)
74 for p in range(min(10, len(w))):
75     if w[p] < 0: # États liés
76         ax1.axhline(y=w[p], color='blue', linestyle='--', linewidth=0.8, alpha=0.5)
77         ax1.text(-9, w[p], f'E_{p}', fontsize=9, color='blue')
78     ax1.axhline(y=0, color='red', linestyle='-', linewidth=1, alpha=0.5, label='E=0')
79 ax1.set_xlabel('Position x', fontsize=12)
80 ax1.set_ylabel('Énergie', fontsize=12)
81 ax1.set_title(f'Double puits symétrique (a={a})', fontsize=13, fontweight='bold')
82 ax1.legend(fontsize=11)
83 ax1.grid(True, alpha=0.3)
84 ax1.set_xlim(-5, 5)
85 ax1.set_ylim(-1.2, 0.5)

86
87 # Fonctions d'onde du premier doublet
88 ax2 = axes[1]
89 colors = ['blue', 'red']
90 for p in range(2):
91     psi_p = psi_norm[:, p]
92     ax2.plot(x, psi_p, color=colors[p], linewidth=2,
93               label=f'_{{p}}(x) (E={w[p]:.6f})', alpha=0.8)
94     ax2.axhline(y=0, color='k', linestyle='-', linewidth=0.5, alpha=0.3)
95     ax2.axvline(x=0, color='k', linestyle='--', linewidth=0.5, alpha=0.3)

```

```

96     ax2.set_xlabel('Position x', fontsize=12)
97     ax2.set_ylabel('(x)', fontsize=12)
98     ax2.set_title('États fondamentaux : symétrique (_0) et antisymétrique (_1)',  

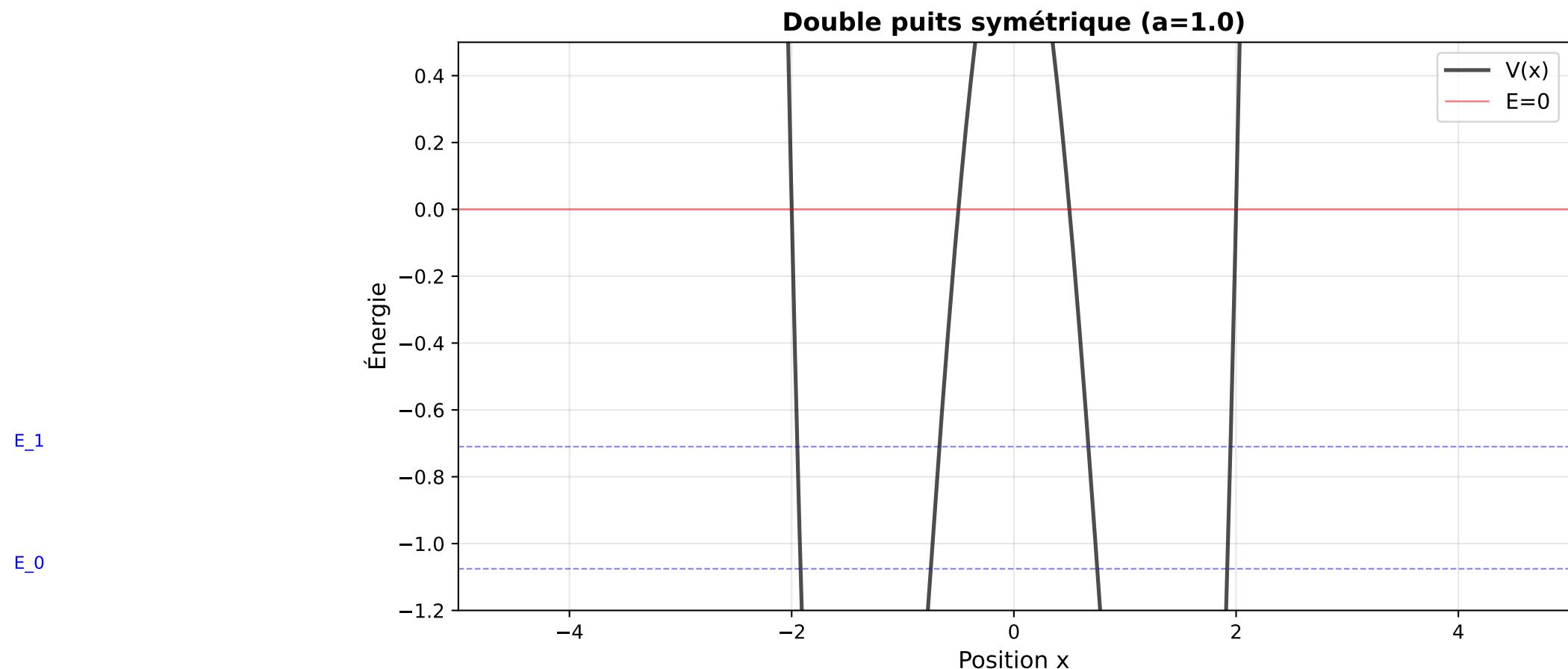
99                 fontsize=13, fontweight='bold')
100    ax2.legend(fontsize=11)
101    ax2.grid(True, alpha=0.3)
102    ax2.set_xlim(-5, 5)
103
104    plt.tight_layout()
105
106    # Sauvegarde
107    output_dir = os.path.join(os.path.dirname(__file__), '..', 'figures')
108    output_path = os.path.join(output_dir, f'q12_double_puits_sym_a{int(a)}.pdf')
109    plt.savefig(output_path, dpi=150)
110    print(f"\nFigure sauvegardée : {output_path}")
111
112    plt.show()
113
114    print("\n==== Commentaire ===")
115    print("• Le potentiel est symétrique par rapport à x=0.")
116    print("• Les états propres sont soit symétriques ((-x) = (x))")
117    print(" soit antisymétriques ((-x) = -(x)).")
118    print("• _0 : état fondamental symétrique, localisé dans les deux puits")
119    print("• _1 : premier état excité antisymétrique, nœud en x=0")
120    print("• La barrière centrale est faible (a=1) : fort couplage entre puits")
121    print("• Doublets quasi-dégénérés : E petit (effet tunnel)")

```

```
122  
123  
124 if __name__ == "__main__":  
125     main()
```



## 10.3 Résultats



**États fondamentaux : symétrique ( $\psi_0$ ) et antisymétrique ( $\psi_1$ )**



### **Commentaire :**

Le potentiel est symétrique par rapport à  $x = 0$ . Les états propres sont soit symétriques ( $\psi(-x) = \psi(x)$ ) soit antisymétriques ( $\psi(-x) = -\psi(x)$ ).

### **Doubles :**

- $\psi_0$  : état fondamental symétrique
- $\psi_1$  : premier excité antisymétrique, nœud en  $x = 0$
- Faible écart  $\Delta E = E_1 - E_0$  : effet tunnel significatif

La barrière centrale (faible avec  $a = 1$ ) permet un fort couplage entre les deux puits.

# Question 11 Double puits symétrique profond ( $a=400$ )

## 11.1 Énoncé

Refaire Q12 avec  $a = 400$  pour augmenter la hauteur de la barrière.

## 11.2 Code

```
1 # -*- coding: utf-8 -*-
2 """
3 TP4 - Question 13 : Double puits symétrique profond
4
5 Refaire Q12 avec a=400 pour étudier l'effet d'une barrière plus haute.
6 """
7
8 import numpy as np
9 import matplotlib.pyplot as plt
10 import os
11 import sys
12
13 sys.path.append(os.path.dirname(__file__))
14 from q01_q02 import potentiel_double_puits
15 from q04_q05 import construire_hamiltonien, diagonaliser_hamiltonien
16 from q07 import normalize
17
```

```

18
19 def main():
20     """
21     Programme principal - Question 13
22     """
23     # Paramètres
24     L = 20.0
25     n = 1000
26     a = 400.0 # Barrière haute
27     r1 = -2.0
28     r2 = -0.5
29     r3 = 0.5
30     r4 = 2.0
31
32     # Discrétisation
33     delta_x = L / (n - 1)
34     x = np.linspace(-L/2, L/2, n)
35
36     print("== TP4 - Question 13 : Double puits symétrique profond ==")
37     print(f"Paramètres : L = {L}, n = {n}")
38     print(f"x = {delta_x:.6f}")
39     print(f"Double puits : a = {a}, r1 = {r1}, r2 = {r2}, r3 = {r3}, r4 = {r4}\n")
40
41     # Potentiel double puits
42     V = potentiel_double_puits(x, a, r1, r2, r3, r4)
43

```

```

44 # Diagonaliser
45 d, e = construire_hamiltonien(x, V, delta_x)
46 w, v = diagonaliser_hamiltonien(d, e)
47
48 # Normaliser
49 psi_norm = normalize(v, delta_x)
50
51 # Affichage énergies
52 print("Énergies des premiers niveaux :")
53 for p in range(min(10, len(w))):
54     print(f" E_{p} = {w[p]:.8f}")
55
56 print("\n==== Analyse des doublets ===")
57 print("Doublet 1 :")
58 print(f" E_0 = {w[0]:.10f}")
59 print(f" E_1 = {w[1]:.10f}")
60 print(f" E = {w[1] - w[0]:.10e} (levée de dégénérescence)")
61
62 print("Doublet 2 :")
63 print(f" E_2 = {w[2]:.10f}")
64 print(f" E_3 = {w[3]:.10f}")
65 print(f" E = {w[3] - w[2]:.10e}")
66
67 # Graphique 1 : Potentiel et énergies
68 fig, axes = plt.subplots(2, 1, figsize=(12, 10))
69

```

```

70 # Potentiel avec niveaux d'énergie
71 ax1 = axes[0]
72 ax1.plot(x, V, 'k-', linewidth=2, label='V(x)', alpha=0.7)
73 for p in range(min(10, len(w))):
74     if w[p] < 0: # États liés
75         ax1.axhline(y=w[p], color='blue', linestyle='--', linewidth=0.8, alpha=0.5)
76         ax1.text(-9, w[p], f'E_{p}', fontsize=9, color='blue')
77     ax1.axhline(y=0, color='red', linestyle='-', linewidth=1, alpha=0.5, label='E=0')
78 ax1.set_xlabel('Position x', fontsize=12)
79 ax1.set_ylabel('Énergie', fontsize=12)
80 ax1.set_title(f'Double puits symétrique profond (a={int(a)})', fontsize=13, fontweight='bold')
81 ax1.legend(fontsize=11)
82 ax1.grid(True, alpha=0.3)
83 ax1.set_xlim(-5, 5)
84 ax1.set_ylim(-120, 50)
85
86 # Fonctions d'onde du premier doublet
87 ax2 = axes[1]
88 colors = ['blue', 'red']
89 for p in range(2):
90     psi_p = psi_norm[:, p]
91     ax2.plot(x, psi_p, color=colors[p], linewidth=2,
92               label=f'_{{p}}(x) (E={w[p]:.6f})', alpha=0.8)
93     ax2.axhline(y=0, color='k', linestyle='-', linewidth=0.5, alpha=0.3)
94     ax2.axvline(x=0, color='k', linestyle='--', linewidth=0.5, alpha=0.3)
95     ax2.set_xlabel('Position x', fontsize=12)

```

```

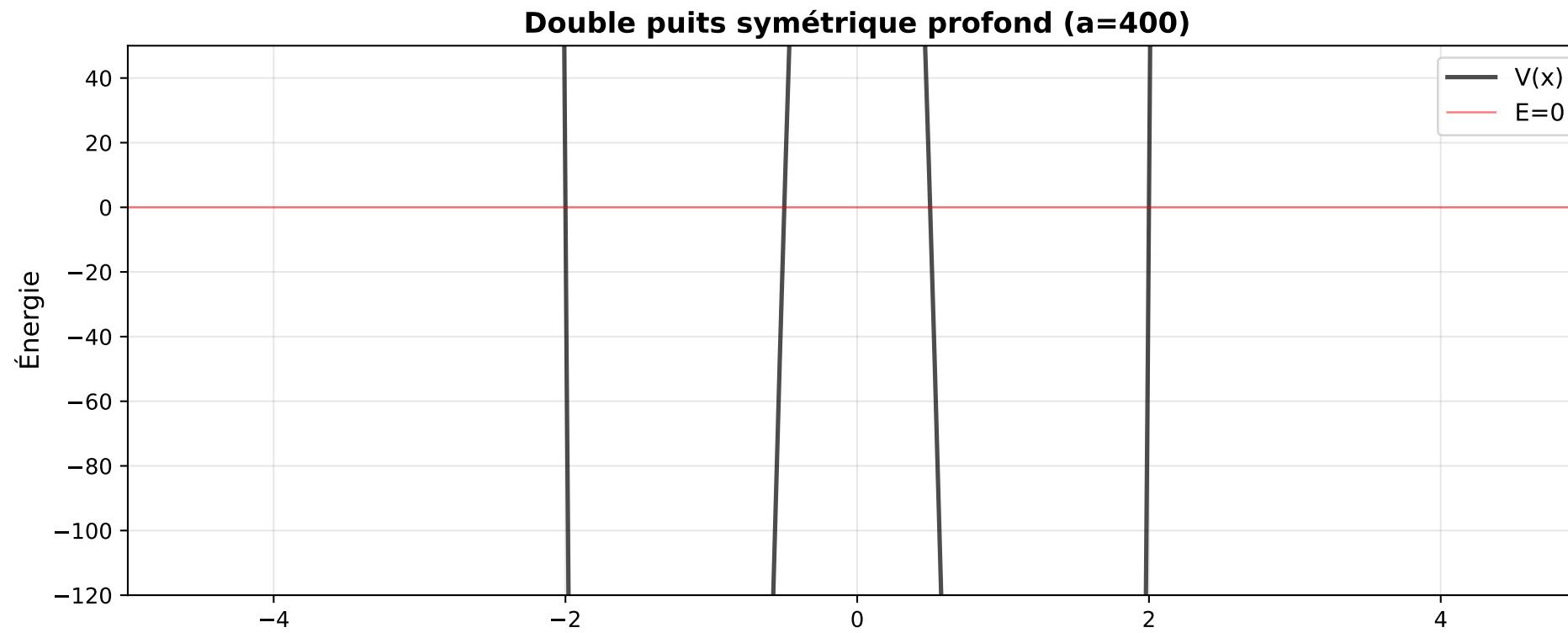
96     ax2.set_ylabel('(x)', fontsize=12)
97     ax2.set_title('États fondamentaux : symétrique (_0) et antisymétrique (_1)',
98                   fontsize=13, fontweight='bold')
99     ax2.legend(fontsize=11)
100    ax2.grid(True, alpha=0.3)
101    ax2.set_xlim(-5, 5)
102
103    plt.tight_layout()
104
105    # Sauvegarde
106    output_dir = os.path.join(os.path.dirname(__file__), '..', 'figures')
107    output_path = os.path.join(output_dir, f'q13_double_puits_sym_a{int(a)}.pdf')
108    plt.savefig(output_path, dpi=150)
109    print(f"\nFigure sauvegardée : {output_path}")
110
111    plt.show()
112
113    print("\n==== Commentaire ===")
114    print("• Avec a=400, la barrière est beaucoup plus haute qu'avec a=1.")
115    print("• L'effet tunnel est fortement supprimé.")
116    print("• Les fonctions d'onde sont très localisées dans chaque puits")
117    print(" avec peu d'amplitude dans la région centrale.")
118    print("• Les doublets sont presque parfaitement dégénérés : E 0.")
119    print("• Limite : deux puits découplés, chacun avec son spectre propre.")
120    print("• Les états ne sont plus _gauche ± _droite mais quasi-isolés.")
121

```

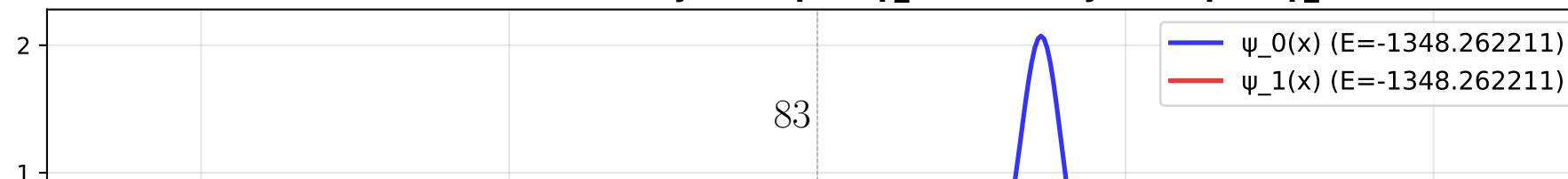
```
122  
123 if __name__ == "__main__":  
124     main()
```



## 11.3 Résultats



**États fondamentaux : symétrique ( $\psi_0$ ) et antisymétrique ( $\psi_1$ )**



### **Commentaire :**

Avec  $a = 400$ , la barrière est bien plus haute. L'effet tunnel est fortement supprimé :

- Les fonctions d'onde sont très localisées dans chaque puits
- Amplitude quasi-nulle dans la région centrale
- Doublets presque parfaitement dégénérés :  $\Delta E \approx 0$
- Limite : deux puits découplés avec spectres indépendants

# Question 12 Double puits asymétrique

## 12.1 Énoncé

Refaire Q12 avec  $r_3 = 0$  au lieu de 0.5 pour briser la symétrie.

## 12.2 Code

```
1 # -*- coding: utf-8 -*-
2 """
3 TP4 - Question 14 : Double puits asymétrique
4
5 Refaire Q12 avec r3=0 au lieu de 0.5 pour briser la symétrie.
6 """
7
8 import numpy as np
9 import matplotlib.pyplot as plt
10 import os
11 import sys
12
13 sys.path.append(os.path.dirname(__file__))
14 from q01_q02 import potentiel_double_puits
15 from q04_q05 import construire_hamiltonien, diagonaliser_hamiltonien
16 from q07 import normalize
17
```

```
18
19 def main():
20     """
21     Programme principal - Question 14
22     """
23     # Paramètres
24     L = 20.0
25     n = 1000
26     a = 1.0
27     r1 = -2.0
28     r2 = -0.5
29     r3 = 0.0 # Asymétrie !
30     r4 = 2.0
31
32     # Discrétisation
33     delta_x = L / (n - 1)
34     x = np.linspace(-L/2, L/2, n)
35
36     print("== TP4 - Question 14 : Double puits asymétrique ==")
37     print(f"Paramètres : L = {L}, n = {n}")
38     print(f"x = {delta_x:.6f}")
39     print(f"Double puits : a = {a}, r1 = {r1}, r2 = {r2}, r3 = {r3}, r4 = {r4}")
40     print(" (r3 = 0 au lieu de 0.5 : symétrie brisée)\n")
41
42     # Potentiel double puits
43     V = potentiel_double_puits(x, a, r1, r2, r3, r4)
```

```

44
45     # Diagonaliser
46     d, e = construire_hamiltonien(x, V, delta_x)
47     w, v = diagonaliser_hamiltonien(d, e)
48
49     # Normaliser
50     psi_norm = normalize(v, delta_x)
51
52     # Affichage énergies
53     print("Énergies des premiers niveaux :")
54     for p in range(min(10, len(w))):
55         print(f"  E_{p} = {w[p]:.8f}")
56
57     print("\n==== Analyse ===")
58     print("Premiers niveaux :")
59     print(f"  E_0 = {w[0]:.10f}")
60     print(f"  E_1 = {w[1]:.10f}")
61     print(f"  E = {w[1] - w[0]:.10e}")
62
63     print("\nNiveaux suivants :")
64     print(f"  E_2 = {w[2]:.10f}")
65     print(f"  E_3 = {w[3]:.10f}")
66     print(f"  E = {w[3] - w[2]:.10e}")
67
68     # Graphique 1 : Potentiel et énergies
69     fig, axes = plt.subplots(2, 1, figsize=(12, 10))

```

```

70
71 # Potentiel avec niveaux d'énergie
72 ax1 = axes[0]
73 ax1.plot(x, V, 'k-', linewidth=2, label='V(x)', alpha=0.7)
74 for p in range(min(10, len(w))):
75     if w[p] < 0: # États liés
76         ax1.axhline(y=w[p], color='blue', linestyle='--', linewidth=0.8, alpha=0.5)
77         ax1.text(-9, w[p], f'E_{p}', fontsize=9, color='blue')
78     ax1.axhline(y=0, color='red', linestyle='-', linewidth=1, alpha=0.5, label='E=0')
79 ax1.set_xlabel('Position x', fontsize=12)
80 ax1.set_ylabel('Énergie', fontsize=12)
81 ax1.set_title(f'Double puits asymétrique (a={a}, r3={r3})', fontsize=13, fontweight='bold')
82 ax1.legend(fontsize=11)
83 ax1.grid(True, alpha=0.3)
84 ax1.set_xlim(-5, 5)
85 ax1.set_ylim(-1.2, 0.5)

86
87 # Fonctions d'onde des deux premiers états
88 ax2 = axes[1]
89 colors = ['blue', 'red']
90 for p in range(2):
91     psi_p = psi_norm[:, p]
92     ax2.plot(x, psi_p, color=colors[p], linewidth=2,
93               label=f'_{{p}}(x) (E={w[p]:.6f})', alpha=0.8)
94     ax2.axhline(y=0, color='k', linestyle='-', linewidth=0.5, alpha=0.3)
95     ax2.axvline(x=0, color='k', linestyle='--', linewidth=0.5, alpha=0.3)

```

```

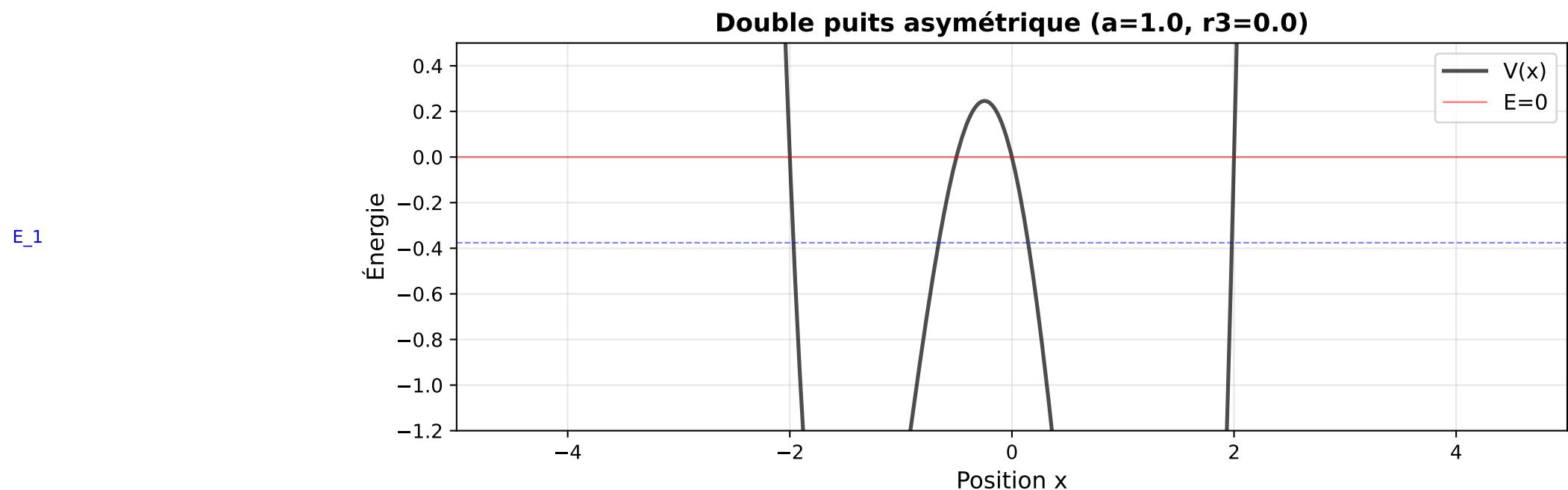
96     ax2.set_xlabel('Position x', fontsize=12)
97     ax2.set_ylabel('(x)', fontsize=12)
98     ax2.set_title('États fondamentaux dans le double puits asymétrique',
99                   fontsize=13, fontweight='bold')
100    ax2.legend(fontsize=11)
101    ax2.grid(True, alpha=0.3)
102    ax2.set_xlim(-5, 5)
103
104    plt.tight_layout()
105
106    # Sauvegarde
107    output_dir = os.path.join(os.path.dirname(__file__), '..', 'figures')
108    output_path = os.path.join(output_dir, f'q14_double_puits_asym_a{int(a)}.pdf')
109    plt.savefig(output_path, dpi=150)
110    print(f"\nFigure sauvegardée : {output_path}")
111
112    plt.show()
113
114    print("\n==== Commentaire ===")
115    print("• La symétrie est brisée : le puits gauche est plus large que le droit.")
116    print("• Les fonctions d'onde ne sont plus symétriques/antisymétriques.")
117    print("• _0 : localisée principalement dans le puits gauche (plus large)")
118    print("• _1 : localisée principalement dans le puits droit")
119    print("• Les niveaux ne sont plus en doublets quasi-dégénérés.")
120    print("• Levée de dégénérescence par brisure de symétrie.")
121    print("• Chaque puits impose sa propre structure de niveaux.")

```

```
122  
123  
124 if __name__ == "__main__":  
125     main()
```



## 12.3 Résultats



### **Commentaire :**

La symétrie est brisée : le puits gauche est plus large que le droit.

### **Conséquences :**

- Les fonctions d'onde ne sont plus symétriques/antisymétriques
- $\psi_0$  : localisée principalement dans le puits gauche (plus large, énergie plus basse)
- $\psi_1$  : localisée principalement dans le puits droit
- Plus de doublets quasi-dégénérés : levée de dégénérescence par brisure de symétrie
- Chaque puits impose sa propre structure de niveaux

# Conclusion

Ce TP a permis de :

1. Implémenter la méthode des différences finies pour résoudre l'équation de Schrödinger 1D
2. Diagonaliser efficacement des matrices tridiagonales avec `scipy`
3. Comparer solutions numériques et analytiques pour le puits infini
4. Comprendre l'impact des paramètres  $L$  et  $n$  sur la précision
5. Étudier l'effet tunnel dans le double puits
6. Observer la levée de dégénérescence par brisure de symétrie

**Limites de la méthode :**

- Conditions aux limites artificielles : nécessite  $L$  grand pour les potentiels non-confinés
- Discrétisation spatiale : nécessite  $\delta x$  petit devant la longueur d'onde
- Compromis  $L$  vs  $n$  : augmenter  $L$  améliore les conditions aux limites mais augmente  $\delta x$  à  $n$  fixé

**Extensions possibles :**

- Potentiels 2D ou 3D (matrices plus complexes)
- Dépendance temporelle : méthode Crank-Nicolson
- Potentiels aléatoires : localisation d'Anderson
- Systèmes à N corps : problème combinatoire