

Nom étudiant: Samuel Ogulluk

Numéro étudiant: 21501479

**LU3PY126 FOAD**

**TP 4 Valeurs propres et vecteurs propres : Résolution de l'équation de Schrödinger par un calcul de différences finies**

---

<b>Table des matières</b>		
1	Formes des potentiels	3
2	Visualisation des potentiels étudiés	4
3	Résolution par MEF	5
4	Puit infini	6
5	Simulation	7
6	Énergies propres numériques	9
7	Détermination des fonctions d'ondes	10
8	Comparaison des résultats obtenus	11
9	Energie propre d'un potentiel harmonique	12
10	Etude de la convergence algorithmique	14
11	Potentiel harmonique et fonction d'onde	15
12	Energie propres associées au double puit	16
13	Double puit symétrique profond	18
14	Double puit symétrique dissymétrique	19

# Introduction

Au cours de ce TP, nous nous intéresserons à l'utilisation de méthodes formelles pour résoudre des problèmes de mécanique quantique.

Ainsi, nous verrons notamment les points suivants :

- Etude de trois formes de potentiels
- Résolution de l'équation de Schrödinger stationnaire par méthode des éléments finis
- Etude des énergies propres et fonctions d'ondes associées à un potentiel
- Comparaison de fonctions d'ondes

Les codes présentés sont en C++ et Python pour la visualisation et sont joints à ce compte-rendu ainsi que disponibles dans le dépôt suivant : [Dépôt Github](#)

# Question 1 Formes des potentiels

On commence par s'intéresser à trois potentiels tels que soit  $x \in [-L/2, L/2]$  :

1. Un puit de potentiel :

$$V(x) = 0$$

2. Un potentiel harmonique :

$$V(x) = \frac{m}{2}\omega^2 x^2$$

3. Un double-puit de potentiel :

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

On a donc 2 fonctions de type  $(\text{double} \mapsto \text{double})$  pour les deux premiers potentiels.

Pour le double-puit de potentiel, on vient ajouter les coefficients en argument de la fonction.

## TP4/q1q2potentiels.cpp

```
3 double V_puits(double) {  
4     return 0.0;  
5 }  
6  
7 double V_harmonique(double x) {  
8     return x * x/2; // k=1, m=1 -> omega^2 = 1  
9 }  
10  
11 double V_double(double x, double a, double r1, double  
    ↪ r2, double r3, double r4) {  
12     return a * (x - r1) * (x - r2) * (x - r3) * (x -  
    ↪ r4);  
13 }
```

## Question 2 Visualisation des potentiels étudiés

On peut ainsi visualiser les potentiels obtenus à l'aide de Python et matplotlib :

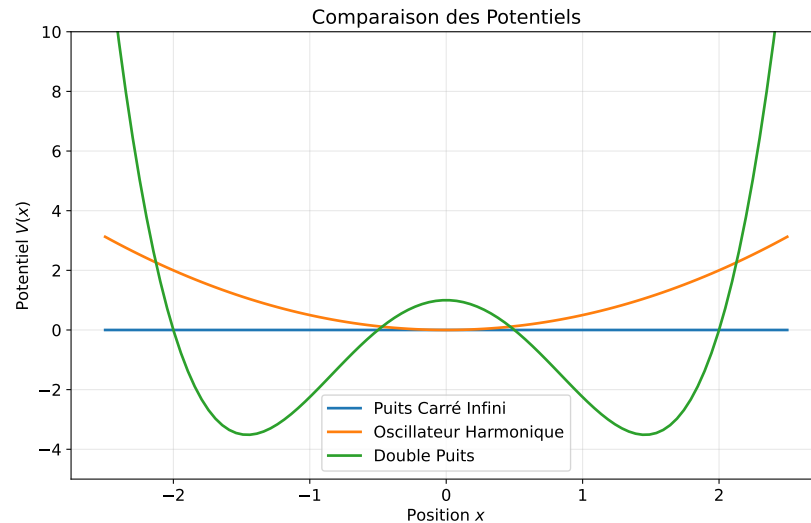


FIGURE 1 – Potentiels en fonction de  $x$

On constate ainsi que les potentiels sont tous pairs et bien défini sur  $[-L/2; L/2]$ .

### TP4/q1q2potentiels.cpp

```
15 void solve_q1_q2_potentiels() {
16     const int n = 100;
17     const double L = 5.0;
18     const double dx = L / (n - 1);
19
20     const double a = 1.0;
21     const double r1 = -2.0, r2 = -0.5, r3 = 0.5, r4 =
        ↪ 2.0;
22
23     ofstream
24     ↪ fichier("resultats/q1_q2_potentials.res");
25
26     for (int i = 0; i < n; i++) {
27         const double x = -L / 2.0 + i * dx;
28         fichier << x << " "
29             << V_puits(x) << " "
30             << V_harmonique(x) << " "
31             << V_double(x, a, r1, r2, r3, r4) <<
32             ↪ endl;
33     }
34
35     fichier.close();
36     cout << "Fichier q1_q2_potentials.res généré." <<
37     ↪ endl;
38 }
```

## Question 3    Résolution par MEF

On vient maintenant discrétiser l'équation de Schrödinger stationnaire via une MEF. On a ainsi transformé le système en une équation matricielle aux valeurs propres pour trouver numériquement les énergies et les fonctions d'onde.

Sens physique de cette transformation :

- La probabilité de trouver la particule en dehors de cet espace est nulle.

Conséquence sur le potentiel effectif :

- Ajout de murs de potentiel infinis aux bornes du domaine
- L'extension de la fonction d'onde est confinée et centrée dans l'intervalle

## Question 4    Puit infini

On vient reprendre les fonctions de la question 1 et définir une fonction potentiel que l'on changera au besoin.

## Question 5 Simulation

On vient maintenant créer un programme permettant de déterminer numériquement les énergies propres et fonctions d'ondes associées à un potentiel. Ainsi, le programme procède comme suit :

1. **Discrétisation et initialisation** : Définir l'intervalle  $[-L/2, L/2]$  divisé en  $n$  segments de longueur  $\delta x$ . Initialiser les vecteurs  $x$  et  $V$  (via `VectorXd`) avec les paramètres  $L = 5$  et  $n = 100$ .
2. **Construction de la matrice** : Instancier la matrice  $H$  de taille  $n \times n$  (initialisée à zéro) et remplir les éléments diagonaux et sous-diagonaux.
3. **Diagonalisation et export** : Calculer les valeurs et vecteurs propres, puis écrire les résultats dans des fichiers dédiés.

### TP4/q6puits.cpp

```
3 void save_puits_results(int n, double L, const
  ↳ VectorXd& x, const VectorXd& energies, const
  ↳ MatrixXd& ondes) {
4   VectorXd energies_theo(n);
5   for (int p = 0; p < n; p++) {
6     double k = M_PI * (p + 1) / L;
7     energies_theo[p] = k * k;
8   }
9
10  std::ofstream
  ↳ f_val("resultats/q6_puits_energies.res");
11  MatrixXd E_out(n, 2);
12  E_out.col(0) = energies; E_out.col(1) =
  ↳ energies_theo;
13  f_val << E_out; f_val.close();
14
15  std::ofstream
  ↳ f_vec("resultats/q6_puits_ondes.res");
16  MatrixXd R(n, n + 1);
17  R.col(0) = x; R.block(0, 1, n, n) = ondes;
18  f_vec << R; f_vec.close();
19 }
```

## TP4/q6puits.cpp

```
21 void solve_q6_puits() {
22     int n = 100; double L = 5.0;
23     double dx = L / (n - 1);
24     double idx2 = 1.0 / (dx * dx);
25
26     VectorXd x(n); MatrixXd H(n, n); H.setZero();
27     for (int i = 0; i < n; i++) {
28         x[i] = -L / 2.0 + i * dx;
29         H(i, i) = 2.0 * idx2 + V_puits(x[i]);
30         if (i > 0) H(i, i - 1) = -idx2;
31         if (i < n - 1) H(i, i + 1) = -idx2;
32     }
33
34     VectorXd energies; MatrixXd ondes;
35     solve(H, energies, ondes);
36     save_puits_results(n, L, x, energies, ondes / sqrt(dx));
37 }
```



## Question 6 Énergies propres numériques

On simule ainsi avec  $n = 100$  et  $L = 5$  et sachant que dans le cas du puits de potentiel, les énergies propres ont pour expression :

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

On constate ainsi que la simulation parvient très bien à décrire la dynamique des premières énergies propres, mais voit son erreur augmenter drastiquement pour les ordres plus élevés. Ceci peut s'expliquer par

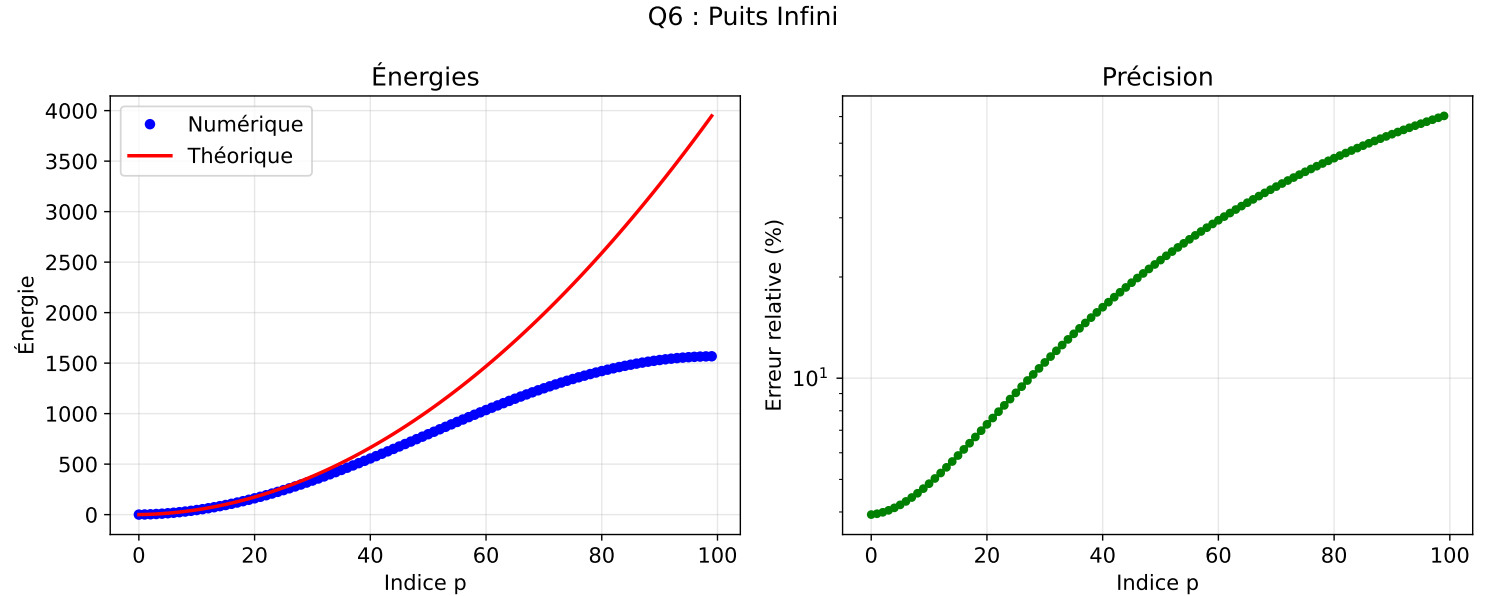


FIGURE 1 – Énergies propres théoriques et prédites et précision de la simulation

## Question 7 Détermination des fonctions d'ondes

On vient ensuite tracer les premières fonctions d'ondes. Pour ce faire, on commence par enregistrer les fonctions d'ondes :

### TP4/q6puits.cpp

```
15  std::ofstream
    ↪ f_vec("resultats/q6_puits_ondes.res");
16  MatrixXd R(n, n + 1);
17  R.col(0) = x; R.block(0, 1, n, n) = ondes;
18  f_vec << R; f_vec.close();
```

Où, "ondes" est

### TP4/q6puits.cpp

```
34  VectorXd energies; MatrixXd ondes;
35  solve(H, energies, ondes);
36  save_puits_results(n, L, x, energies, ondes /
    ↪ sqrt(dx));
```

Puis on trace à l'aide de matplotlib le résultat.

On remarque que le résultat correspond aux attendus, la fonction d'onde s'annule toujours aux extrémités et s'annule  $p$  fois entre les deux.

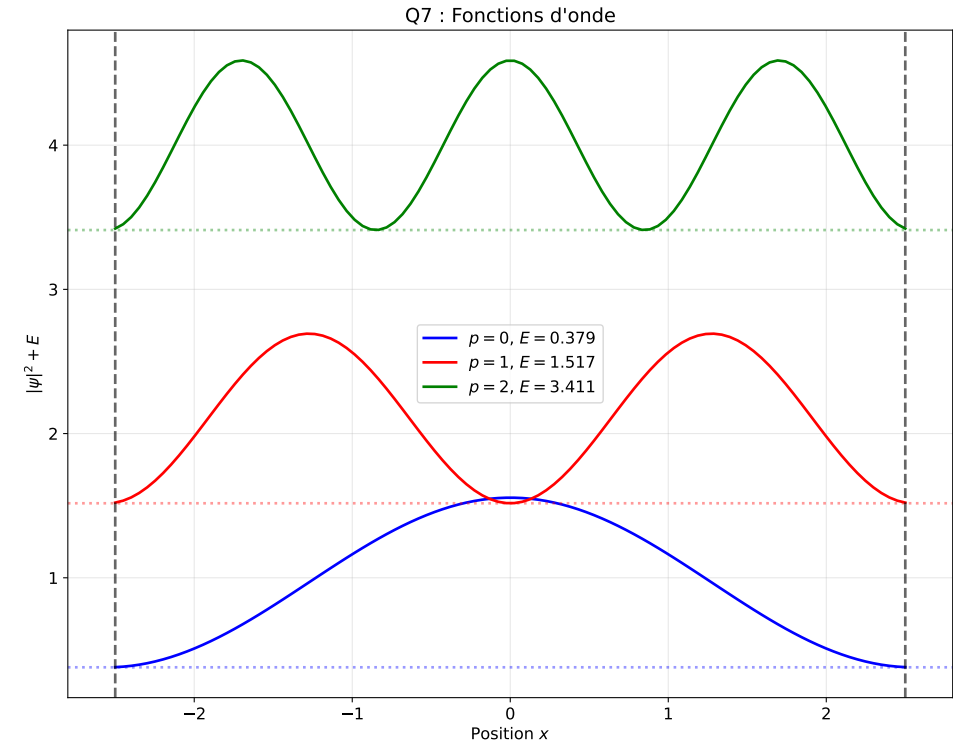


FIGURE 2 – Fonctions d'ondes associées au puit de potentiel

## Question 8 Comparaison des résultats obtenus

On peut ensuite tracer une comparaison entre les fonctions d'ondes numériques et leur valeurs théoriques (dans ce rare cas où l'expression analytique est connue) :

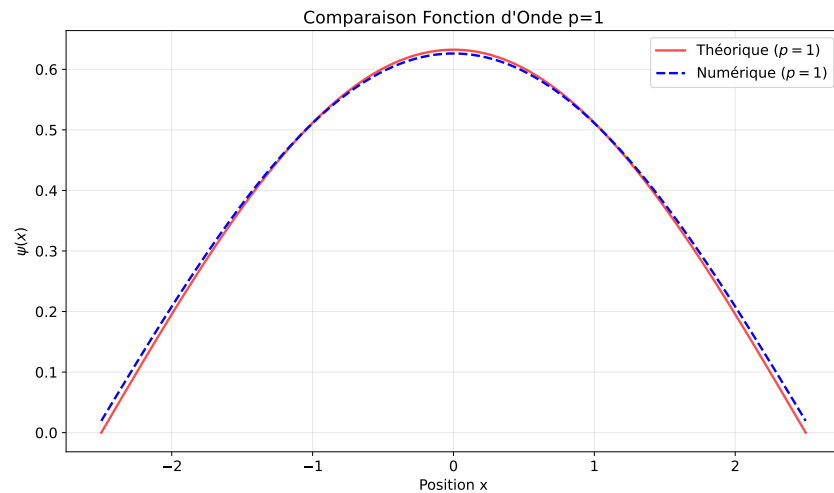


FIGURE 2 – Fonctions d'ondes pour  $p=1$

On constate ainsi que pour  $p=1$ , les erreurs sont très faibles, la dynamique est respectée.

Cependant, pour un  $p$  plus élevé, ici  $p=55$  on constate que la fonction d'onde théorique et numériques présentent de larges différences :

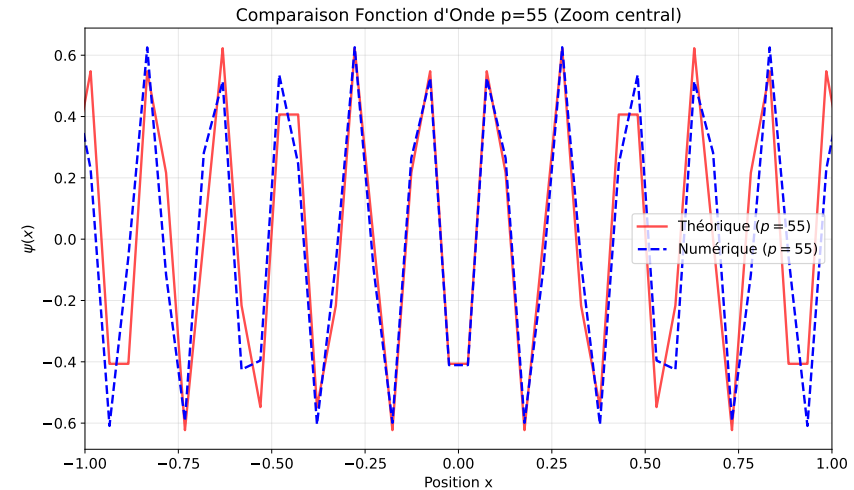


FIGURE 3 – Fonctions d'ondes pour  $p=55$

Une des explications possible pourrait être analogique au critère de Shannon-Nyquist, ici, pour des ordres trop élevés, l'échantillonnage pourrait ne pas être suffisant.

## Question 9 Énergie propre d'un potentiel harmonique

On s'intéresse maintenant au potentiel harmonique, en venant reproduire les simulations précédentes. On trace donc les énergies obtenues, afin de les comparer à leur valeur théoriques :

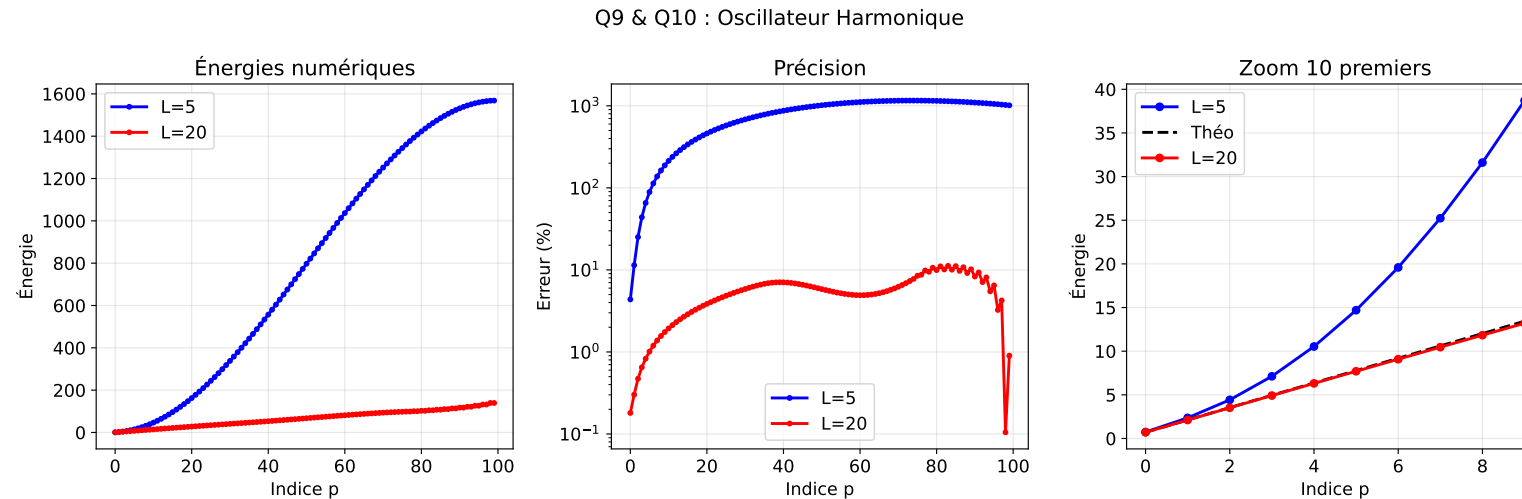


FIGURE 3 – Énergies propres obtenues en fonction de l'indice  $p$

On constate ainsi qu'encore une fois, plus  $p$  est petit, plus la simulation est correcte. Cependant, on constate que l'erreur présente un plateau, voire diminue lorsque  $p$  est très important .

## TP4/q9harmonique.cpp

```

3 void save_harm_results(double L, int n, double dx,
    ↪ const VectorXd& x, const VectorXd& energies, const
    ↪ MatrixXd& ondes) {
4     VectorXd eth(n);
5     double hbar_w = sqrt(2.0);
6     for (int p = 0; p < n; p++) eth[p] = (p + 0.5) *
    ↪ hbar_w;
7
8     string suff = "_n" + to_string(n) + "_L" +
    ↪ to_string((int)L);
9     ofstream f_val(("resultats/q9_harm_energies" +
    ↪ suff + ".res").c_str());
10    MatrixXd E_out(n, 2);
11    E_out.col(0) = energies; E_out.col(1) = eth;
12    f_val << E_out; f_val.close();
13
14    ofstream f_vec(("resultats/q9_harm_ondes" + suff +
    ↪ ".res").c_str());
15    MatrixXd R(n, n + 1);
16    R.col(0) = x; R.block(0, 1, n, n) = ondes;
17    f_vec << R; f_vec.close();
18
19    ofstream f_pot(("resultats/q9_harm_pot" + suff +
    ↪ ".res").c_str());
20    for(int i=0; i<n; i++) f_pot << x[i] << " " <<
    ↪ V_harmonique(x[i]) << endl;
21    f_pot.close();
22 }

```

## TP4/q9harmonique.cpp

```

24 void solve_harmonique_pour_L(double L, int n) {
25     double dx = L / (n - 1), idx2 = 1.0 / (dx * dx);
26     VectorXd x(n); MatrixXd H(n, n); H.setZero();
27
28     for (int i = 0; i < n; i++) {
29         x[i] = -L / 2.0 + i * dx;
30         H(i, i) = 2.0 * idx2 + V_harmonique(x[i]);
31         if (i > 0) H(i, i - 1) = -idx2;
32         if (i < n - 1) H(i, i + 1) = -idx2;
33     }
34
35     VectorXd energies; MatrixXd ondes;
36     solve(H, energies, ondes);
37     save_harm_results(L, n, dx, x, energies, ondes /
    ↪ sqrt(dx));
38 }
39
40 void solve_q9_harmonique() {
41     solve_harmonique_pour_L(5.0, 100);
42     solve_harmonique_pour_L(20.0, 100);
43 }

```

## Question 10 Etude de la convergence algorithmique

On s'intéresse ensuite à l'effet de  $L$  sur les énergies. Ainsi, on peut constater sur les figures 3 et 4, que plus  $L$  est élevé, plus la simulation parvient à décrire précisément les énergies.

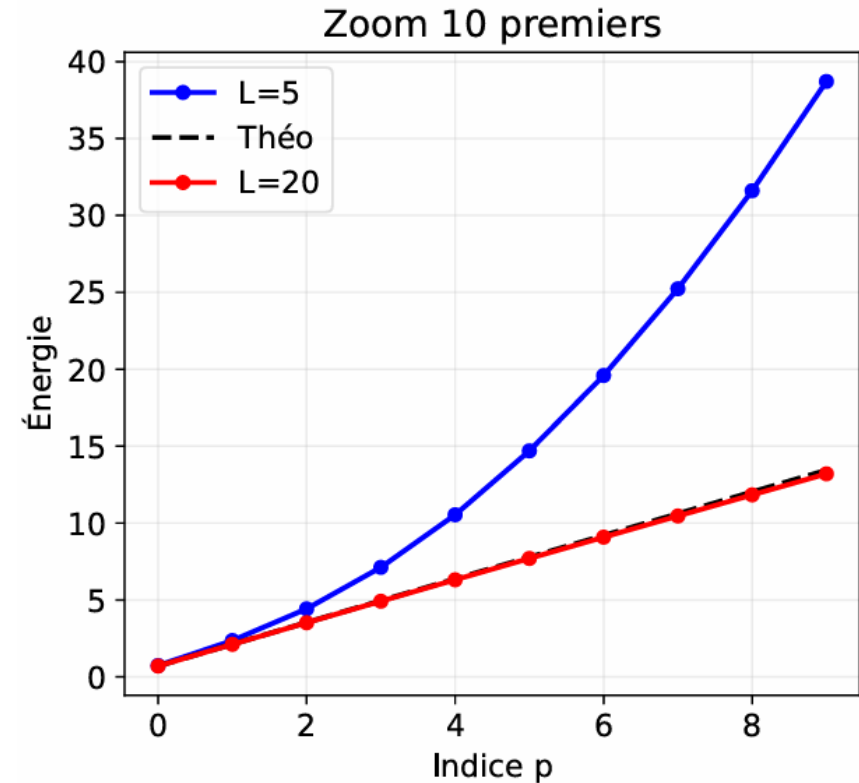


FIGURE 4 – Énergies en fonction de  $p$

## Question 11 Potentiel harmonique et fonction d'onde

On vient maintenant s'intéresser au lien entre potentiel, ordre d'excitation et fonction d'onde. On constate ainsi les points suivants :

- La largeur augmente avec  $p$  car les points de rebroussement classiques, où  $V(x) = E_p$ , s'éloignent de l'origine à mesure que l'énergie augmente.
- Pour  $p = 0$ , la densité est maximale au centre, alors que pour  $p$  élevé, la particule passe plus de temps près des parois du potentiel, se rapprochant du comportement classique.
- La fonction d'onde au carré ne s'annule pas aux limites du potentiel mais décroît exponentiellement. La probabilité de présence dans la zone classiquement interdite n'est donc pas nulle, c'est l'effet tunnel .

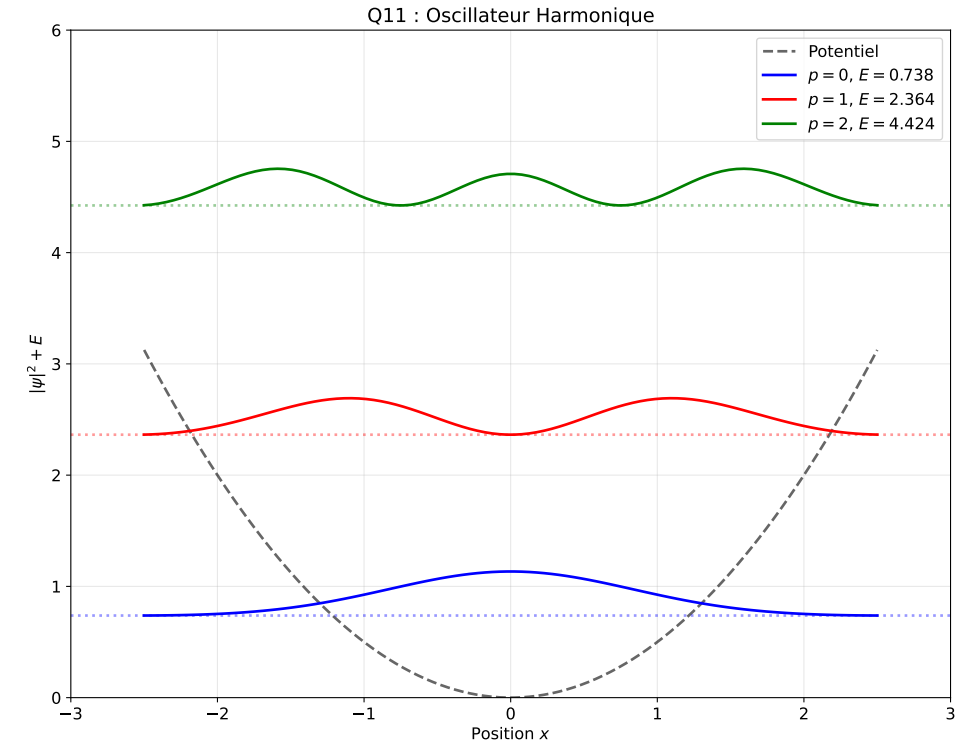


FIGURE 5 – Potentiel harmonique et fonctions d'ondes

## Question 12 Energie propres associées au double puit

On s'intéresse maintenant à un double-puit de potentiel :

$$V(x) = (x + 2)(x + 0.5)(x - 0.5)(x - 2)$$

Un tel potentiel peut modéliser une particule entre deux positions d'équilibre stables.

- Inversion de  $NH_3$ , transfert de protons (ADN) et qubits supraconducteurs.
- Les niveaux  $E_0$  (symétrique) et  $E_1$  (antisymétrique) forment un doublet serré, illustrant la levée de dégénérescence par effet tunnel.
- Analogue aux orbitales liantes/antiliantes, avec une délocalisation de la particule sur les deux puits malgré la barrière.

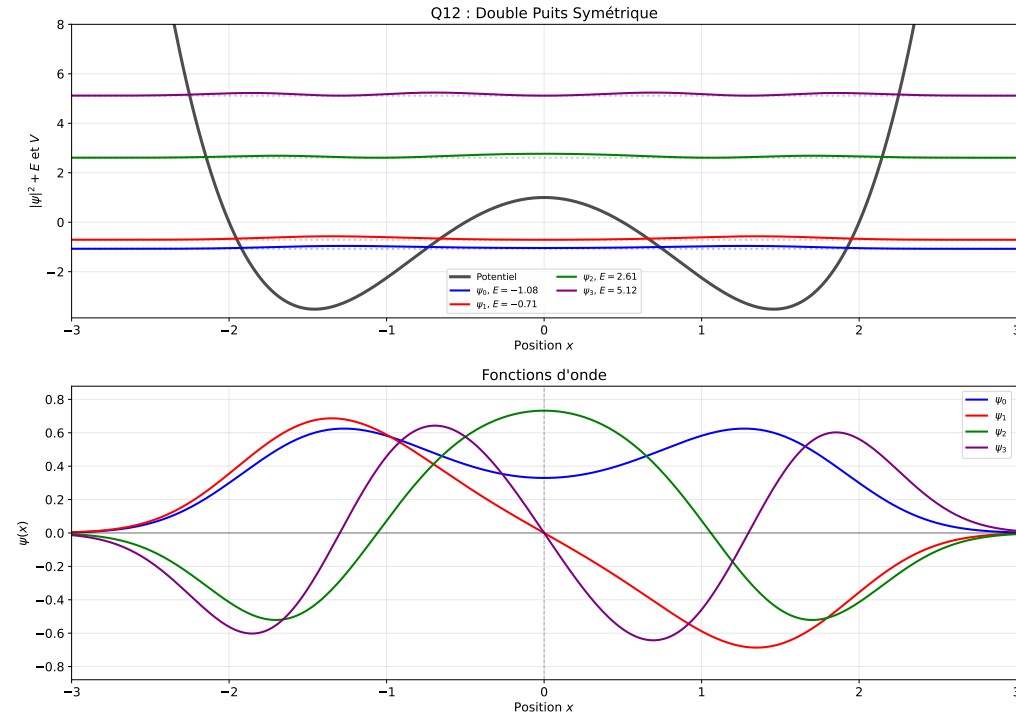


FIGURE 6 – Double-puit de de potentiel et fonctions d'ondes associées



## TP4/q12doublepuits.cpp

```

3 void save_double_results(const VectorXd& x, const
  ↳ VectorXd& energies, const MatrixXd& ondes, string
  ↳ label, double a, double r1, double r2, double r3,
  ↳ double r4) {
4   string suff = " " + label;
5   ofstream f_e(("resultats/q12_double_energies" +
  ↳ suff + ".res").c_str());
6   f_e << energies; f_e.close();
7
8   ofstream f_o(("resultats/q12_double_ondes" + suff
  ↳ + ".res").c_str());
9   MatrixXd R(x.size(), x.size() + 1);
10  R.col(0) = x; R.block(0, 1, x.size(), x.size()) =
  ↳ ondes;
11  f_o << R; f_o.close();
12
13  ofstream f_p(("resultats/q12_double_pot" + suff +
  ↳ ".res").c_str());
14  for(int i=0; i<x.size(); i++) f_p << x[i] << " "
  ↳ << V_double(x[i], a, r1, r2, r3, r4) << endl;
15  f_p.close();
16 }

```

## TP4/q12doublepuits.cpp

```

18 void solve_double_puits_config(int n, double L, double
  ↳ a, double r1, double r2, double r3, double r4,
  ↳ string label) {
19   double dx = L / (n - 1), idx2 = 1.0 / (dx * dx);
20   VectorXd x(n); MatrixXd H(n, n); H.setZero();
21
22   for (int i = 0; i < n; i++) {
23     x[i] = -L / 2.0 + i * dx;
24     H(i, i) = 2.0 * idx2 + V_double(x[i], a, r1,
  ↳ r2, r3, r4);
25     if (i > 0) H(i, i - 1) = -idx2;
26     if (i < n - 1) H(i, i + 1) = -idx2;
27   }
28
29   VectorXd energies; MatrixXd ondes;
30   solve(H, energies, ondes);
31   save_double_results(x, energies, ondes / sqrt(dx),
  ↳ label, a, r1, r2, r3, r4);
32 }
33
34 void solve_q12_double_puits() {
35   int n = 1000; double L = 20.0;
36   solve_double_puits_config(n, L, 1.0, -2.0, -0.5,
  ↳ 0.5, 2.0, "q12_sym");
37   solve_double_puits_config(n, L, 400.0, -2.0, -0.5,
  ↳ 0.5, 2.0, "q13_deep");
38   solve_double_puits_config(n, L, 1.0, -2.0, -0.5,
  ↳ 0.0, 2.0, "q14_asym");
39 }

```

## Question 13 Double puit symétrique profond

L'augmentation du paramètre  $a$  modifie radicalement le comportement du système :

- Les énergies  $E_0, E_1$  et  $E_2, E_3$  apparaissent désormais par paires quasi dégénérées. La barrière est si haute que l'effet tunnel est fortement réduit.
- Les fonctions d'onde sont beaucoup plus étroites et confinées au fond des puits. Elles se comportent localement comme des oscillateurs harmoniques indépendants.
- Les valeurs d'énergie sont nettement plus basses (très négatives) et l'écart entre les doublets est plus grand que dans le cas  $a = 1$ .

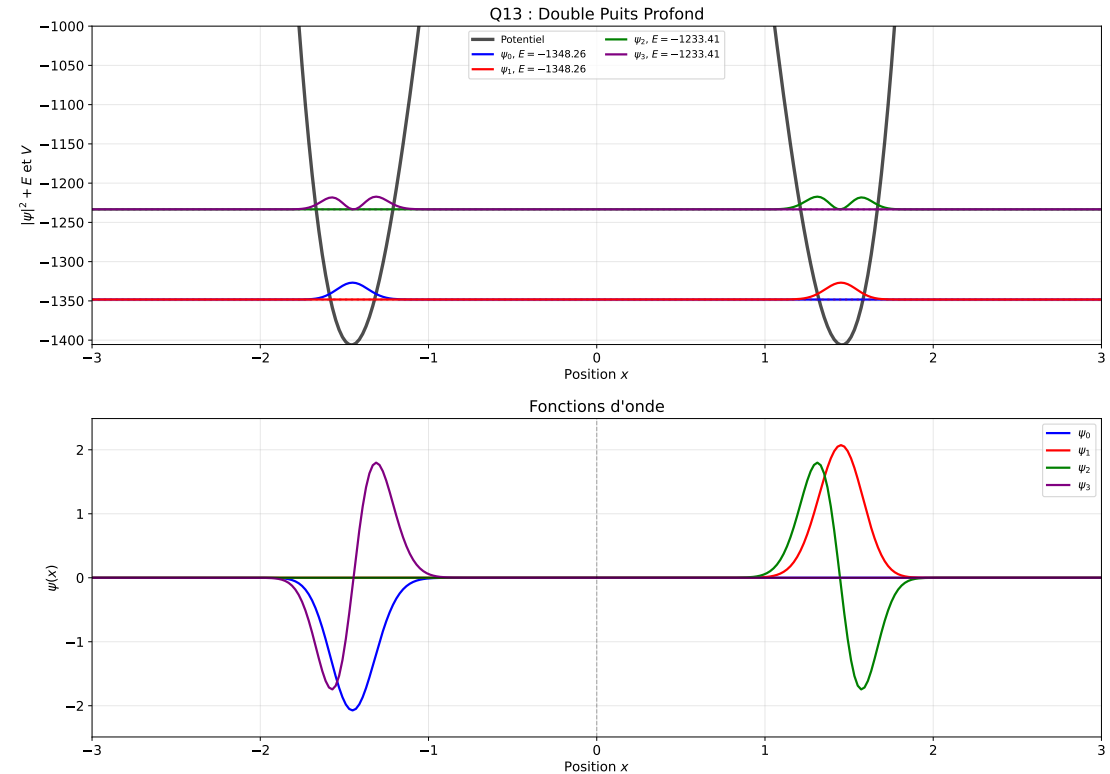


FIGURE 7 – Puits de potentiel profond et fonctions d'ondes associées

## Question 14 Double puit symétrique dissymétrique

En déplaçant la racine  $r_3$  à 0, on obtient un double puits dissymétrique. L'observation des résultats montre les points suivants :

- Les fonctions d'onde ne présentent plus de symétrie par rapport à l'origine. La parité n'est plus un bon nombre quantique pour ce système.
- Les états de plus basse énergie se concentrent dans le puits le plus bas. On observe une transition d'un état délocalisé (cas symétrique) vers un état plus localisé.
- L'écart entre  $E_0$  et  $E_1$  augmente significativement. La structure en doublets serrés disparaît car les puits ne sont plus résonnants entre eux.

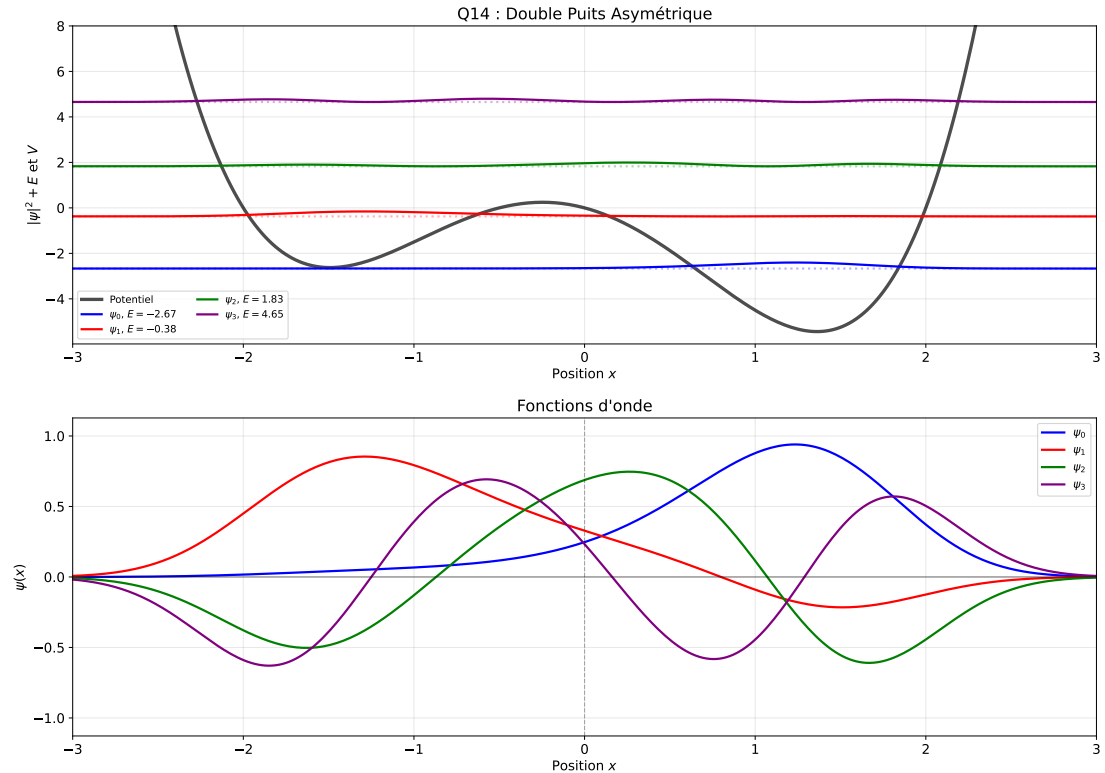


FIGURE 8 – fonctions d'ondes pour un puit dissymétrique

# Conclusion

— Energie :

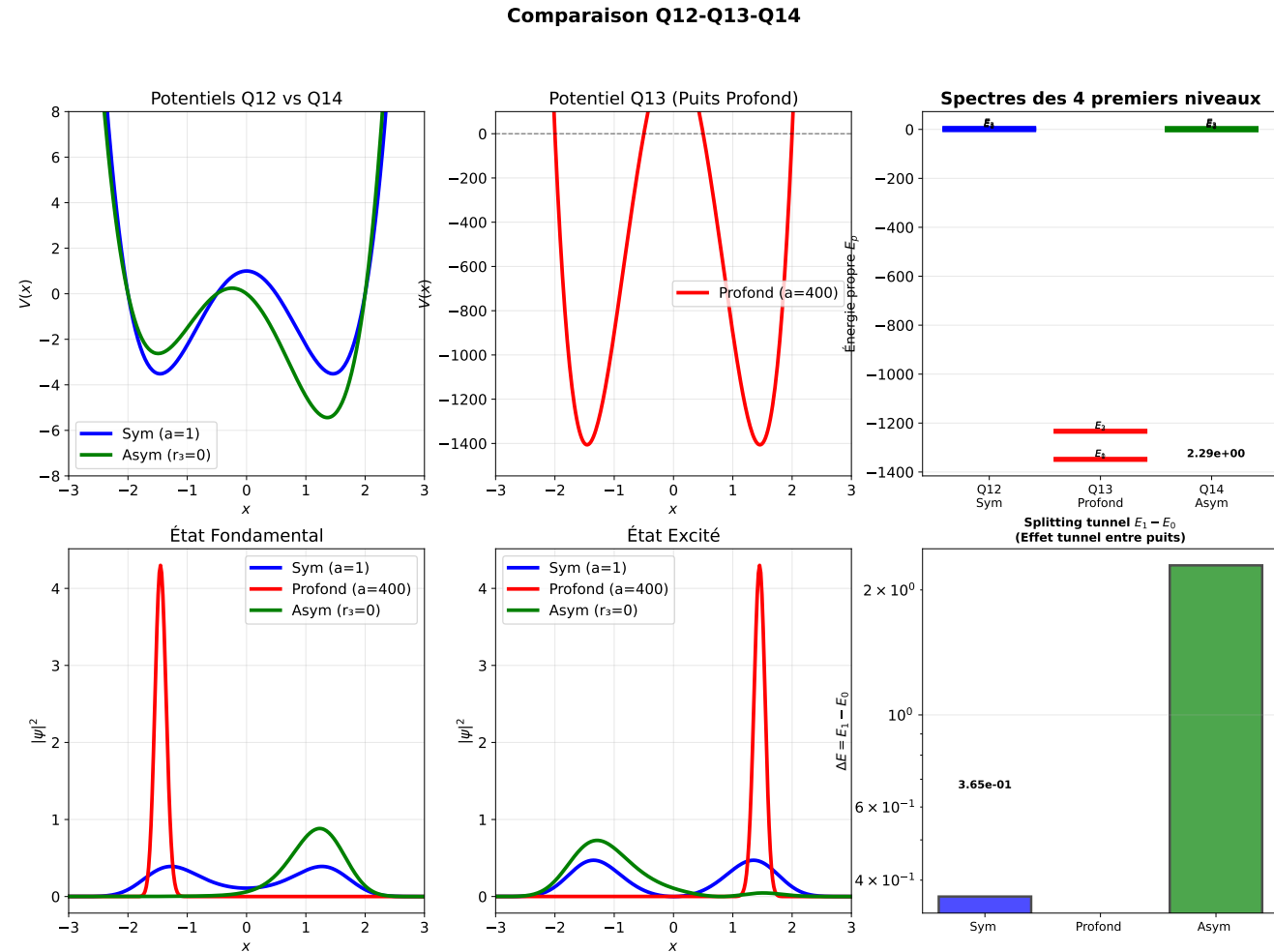
$E_{PP}$  est très bas tandis que  $E_{sym}$  et  $E_{assym}$  se situent près de 0.

— Localisation :

$V_{assym}$  concentre la particule dans le puits le plus profond, Pour  $V_{sym}$  la présence est partagée sur les deux puits.

— Effet tunnel :

$\Delta E = E_1 - E_0$  maximale pour le système asymétrique Q14 et quasi nul pour le puit profond.



# Annexe

Le fichier visualisation.py est disponible dans le dépôt indiqué plus haut (d'une longueur de 300 lignes, il prendrait trop de place en annexe de ce TP).