The effect of virtual $B\bar{B}$ meson pairs on the Upsilon spectrum.

Author: Adrián Asensio Magariños
Facultat de Física, Universitat de Barcelona, Diagonal 645, 08028 Barcelona, Spain.

Advisor: Joan Soto Riera

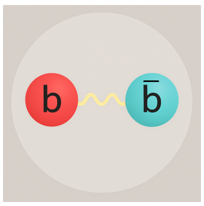
Abstract: This work studies the impact of $B\bar{B}$ meson pairs on the Upsilon Υ spectrum. First we solve the radial Schrodinger equation with a Cornell potential to compute the spectrum, reproducing known levels accurately. Then we analyze the effect of string breaking through both adiabatic and coupled channel approaches. The adiabatic approach uses diagonalization to obtain effective potentials, while the coupled-channel formalism solves the system of two coupled Schrödinger equations in order to model the mixing dynamically. Numerical solvers developed for each method reveal that virtual meson pairs alter the energy states. The obtained results provide theoretical predictions that are relevant for future experiments and lattice QCD simulations.

Keywords: Schrödinger equation, wavefunction, quark, meson, energy spectrum, shooting method
SDGs: 4. Quality Education

I. INTRODUCTION

The Upsilon (Υ) meson family belongs to a group of high-mass particles composed of a bottom quark (b) and its corresponding antiparticle ($\bar{b}$), commonly referred to as a bottomonium states. Discovered at Fermilab in 1977 (the state $1S$), the research and exploration of the Upsilon meson family has been very significant for the understanding of heavy quarkonium systems and the study of the strong interaction, described by Quantum Chromodynamics (QCD).

FIG. 1: Upsilon meson (Υ).



An interesting feature of the Upsilon meson family is that its large mass in the ground state ($\approx 9.46 GeV/c^2$) allows it to exhibit a spectrum of well-defined energy levels. These levels are analogous to the hydrogen atom, but ruled by the non-Abelian nature of the strong force. Characterizing this energy spectrum will be the first part of this study.

Once these states cross a certain mass threshold, a phenomenon known as "string breaking" can occur. During this process, a light quark-antiquark pair is spontaneously created due to quantum fluctuations and then combined with the heavy b and $\bar{b}$ quarks to form a pair of virtual $B\bar{B}$ mesons. The effect of these virtual B mesons pairs on the Upsilon spectrum will be the focus of the second part of the work.

It is important to note that the entire analysis is carried out in the framework of the Non-relativistic QCD (NRQCD) [1], assuming the heavy quark limit $m_b \gg \Lambda_{QCD}$, where $m_b = 4.885\ GeV$ and $\Lambda_{QCD} \sim 0.2\ GeV$. This justifies the use of non-relativistic quantum mechanics and potential models and provides theoretical support to the work. We will focus on the radial dynamics of the system. Quantum spin effects such as spin-spin and spin-orbit interactions, are suppressed by $1/m_b^2$, and those corrections will not be included. This approximation allows us to isolate the impact of the channel coupling mechanism on the spectrum, leaving spin effects outside of our analysis. Throughout this article we use natural units $\hbar = c = 1$, $E,\ m\ [GeV]$.

II. DEVELOPMENT

The main objective of this work is to understand the effect of virtual B meson pairs on the Upsilon spectrum. This section is divided into two parts; the characterization of the spectrum of the Upsilon meson and the study of string breaking.

A. Υ Spectrum.

The first objective is to characterize the spectrum of the Upsilon meson. To this end, we will adopt an interaction potential based on [2], in which the binding energy of the Upsilon meson was fitted using a Cornell Potential:

$$V_\Upsilon(r) = -\frac{a}{r} + \sigma r + c \tag{1}$$

In this kind of potential, FIG.2, the first term represents the Coulomb-like attraction at short distances and the second term models the linear confinement due to the gluon binding force. The numerical values obtained in [2] fit are:

$$a = (0.369 \pm 0.015)$$

$$\sigma = (0.190 \pm 0.003)\ GeV^2$$
$$c = (-1.13 \pm 0.05)\ GeV$$

Once the potential has been defined, we are now ready to solve the radial Schrödinger equation:

$$-\frac{1}{2\mu_\Upsilon}\frac{\mathrm{d^2u(r)}}{\mathrm{dr^2}} + \left(\frac{\mathrm{l(l+1)}}{2\mu_\Upsilon \cdot \mathrm{r^2}} + \mathrm{V_\Upsilon(r)}\right)\mathrm{u(r)} = \mathrm{E\,u(r)}. \quad (2)$$

where $\mu_\Upsilon = \frac{m_b}{2}$ is the reduced mass of the meson, and $m_b = 4.885$ GeV.

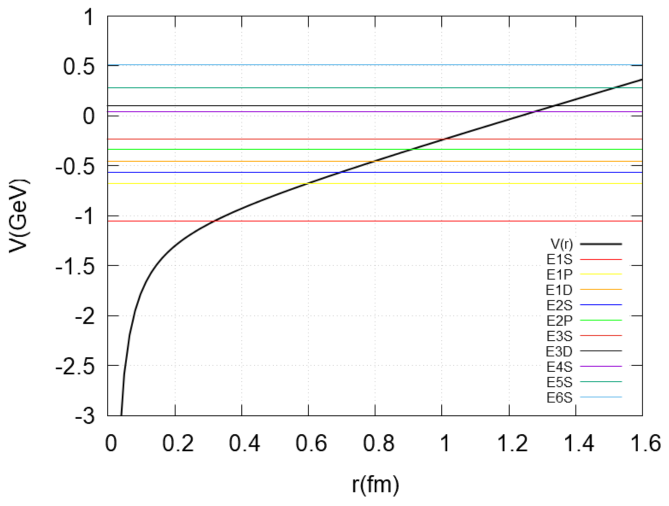
Using a numerical solver for the one-channel Schrödinger equation (provided in the Appendix A of this work) the spectrum of the meson has been computed. This solver is based on the method originally developed in [3], but adapted for the case of the Cornell potential.
The approach is based on the shooting method, where the Schrödinger equation is integrated for trial energies, and the boundary condition at large distances $r \to \infty, u(r) \approx 0$ is monitored. The eigenvalues of the energy are found by locating the energies for which the wavefunction decays properly, using a root-finding method based on bisection.
Once an energy level is determined, the wavefunction is integrated over a fine radial grid and normalized. The code implementation allows for the adjustment of model parameters, orbital angular momentum (l) and the number of nodes of the solution (n_0).

The table below compares the results obtained with this solver to the experimental values reported by Particle Data Group (PDG):

TABLE I: Energy levels of the Upsilon spectrum, via Particle Data Group, and results obtained of solving the Schrödinger equation using the Cornell Potential (eq1) for $c_1 = -1.16\ GeV$, third column, and using $c_2 = 9.42\ GeV$, fourth column.

Υ **Spectrum**	E_{PDG} **(GeV)**	$E^{c_1}_{solver}$ $\pm 0.002\ (GeV)$	$E^{c_2}_{solver}$ $\pm 0.002\ (GeV)$
E1S	9.46	-1.051	9.499
E1P	9.86	-0.676	9.874
E1D	10.16	-0.449	10.101
E1F	not found	-0.265	10.285
E2S	10.02	-0.563	9.987
E2P	10.23	-0.334	10.216
E2D	not found	-0.157	10.393
E2F	not found	-0.001	10.549
E3S	10.36	-0.234	10.316
E3P	not found	-0.053	10.497
E3D	10.57	0.099	10.649
E3F	not found	0.238	10.788
E4S	10.58	0.040	10.590
E4P	not found	0.196	10.746
E4D	not found	0.332	10.882
E4F	not found	0.459	11.009
E5S	10.85	0.283	10.833
E5P	not found	0.423	10.973
E6S	11.00	0.512	11.062

FIG. 2: Plot of Cornell's Potential, $V_\Upsilon(r)$ (GeV), with the corresponding spectrum of the Upsilon meson calculated with the solver of the Schrödinger equation.



B. Effect of virtual $B\bar{B}$ meson pairs on the Υ spectrum

To address the main objective of this study, in this section there will be two approaches: beginning with an adiabatic approximation and followed by the solution of the coupled two-channel Schrödinger equations.

1. *Adiabatic Approximation*

The main idea of this approach is based on the models of quarkonium hybrid and heavy-meson pair coupling potentials, V_{mix}, developed at [4]. From the three models presented, we will work with the Model III presented, where in this model the parameters have been tuned to match the potential obtained from the lattice QCD computations at [5]. Therefore, in our study we adopt the following mixing potential:

$$V_{mix}(r) = 8\Lambda^2_{QCD} \cdot r \cdot \left(\frac{r_0}{r_0 + r}\right)^5 + \frac{0.9}{\Lambda^2_{QCD} \cdot r^3} \cdot \left(\frac{r}{r_0 + r}\right)^5 \quad (3)$$

where $\Lambda_{QCD} = 0.2\ GeV$ and $r_0 = 0.275\ fm$.

To account for the mixing between the quarkonium state and the heavy-meson pair channel, we consider a coupled channel-framework. The interaction is represented by a 2×2 matrix where the off-diagonal terms induce the mixing and we will consider the $B\bar{B}$ mesons as an open channel.

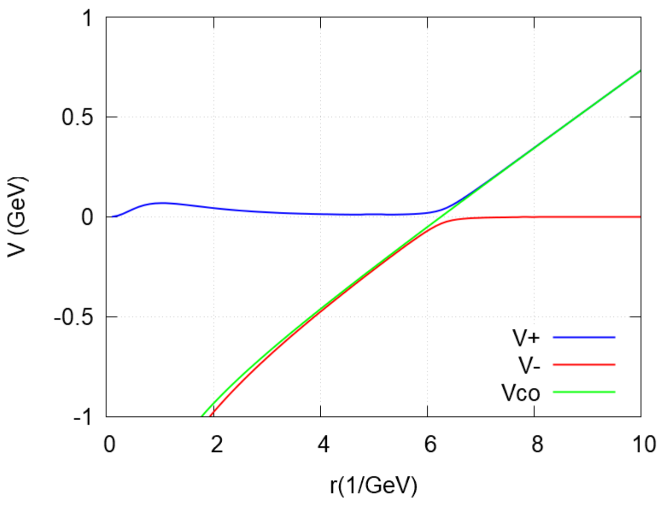
$$V(r) = \begin{pmatrix} V_\Upsilon(r) & V_{mix}(r) \\ V_{mix}(r) & 0 \end{pmatrix} \quad (4)$$

The main objective, by diagonalizing the potential matrix, is to obtain the following effective potentials:

$$\mathrm{V}_\pm(\mathrm{r}) = \frac{\mathrm{V}_\Upsilon(\mathrm{r}) \pm \sqrt{(\mathrm{V}_\Upsilon(\mathrm{r}))^2 + 4\cdot(\mathrm{V}_{\mathrm{mix}}(\mathrm{r}))^2}}{2} \tag{5}$$

where $V_\Upsilon(r)$ and $V_{mix}(r)$ are defined in equations (1) and (3), respectively. A graphical representation of the behavior of these potentials, $V_\pm$, is shown below.

FIG. 3: Plot of the effective potentials, $V_\pm$ (GeV), in comparison with the Cornell potential, V_Υ (GeV), as a function of r (1/GeV). Centered in the string breaking region.



Using the solver that can be found in Appendix A), we can now compute the spectrum of both V_- and V_+:

TABLE II: Results obtained of solving the Schrödinger equation using the effective potentials (V_- and V_+).

V_-	$E_{solver} \pm 0.002$ **(GeV)**	V_+	$E_{solver} \pm 0.02$ **(GeV)**
E1S	-1.105	E1S	0.06
E1P	-0.722	E1P	0.09
E1D	-0.482	E1D	0.13
E1F	-0.288	E1F	0.18
E2S	-0.594	E2S	0.16
E2P	-0.363	E2P	0.22
E2D	-0.183	E2D	0.29
E2F	-0.029	E2F	0.36
E3S	-0.257	E3S	0.31
E3P	-0.082	E3P	0.39
E3D		E3D	0.47
E3F		E3F	0.55
E4S		E4S	0.48
E4P		E4P	0.57
E4D		E4D	0.65
E5S		E5S	0.66
E5P		E5P	0.74
E6S		E6S	0.83

2. *Diabatic approximation.*

To go beyond the adiabatic approximation and fully capture the effect of the mixing between the bottomonium state and the heavy-meson pair channel, we consider the coupled channel formalism.

The physical system is described by a two-component radial wavefunction and the dynamics are governed by two coupled Schrödinger equations. These equations account for the interaction between the two channels through off-diagonal terms in the potential matrix, equation (3).

Solving this system allows us to obtain more accurate corrections to the spectrum of the Upsilon meson by fully accounting for the channel mixing. The coupled radial Schrödinger equation take the following form:

$$\begin{aligned} -\frac{1}{2\mu_\Upsilon}\frac{d^2u_1(r)}{dr^2} + V_{11}(r)u_1(r) + V_{\mathrm{mix}}(r)\,u_2(r) &= E\,u_1(r) \\ -\frac{1}{2\mu_B}\frac{d^2u_2(r)}{dr^2} + V_{22}(r)\,u_2(r) + V_{\mathrm{mix}}(r)\,u_1(r) &= (E+\Delta)\,u_2(r) \end{aligned} \tag{6}$$

In this system, $u_1(r)$ describes the radial wavefunction of the bottomonium ($b\bar{b}$) state, while $u_2(r)$ corresponds to the heavy-meson channel.

The potential $V_{11}(r)$ includes a centrifugal barrier $\frac{l(l+1)}{2\mu_\Upsilon\cdot r^2}$ and the Cornell potential, equation (1). The second channel, represented by $V_{22}(r)$, is treated as an open channel and only includes de centrifugal barrier $\frac{l(l+1)}{2\mu_B\cdot r^2}$, where $\mu_B = \frac{m_B}{2}$ and $m_B = 5.3134$ GeV. The off-diagonal terms $V_{mix}(r)$, equation (3), introduce the mixing between the two channels and $\Delta = 2(m_B - m_b)$ is the threshold shift between channels.

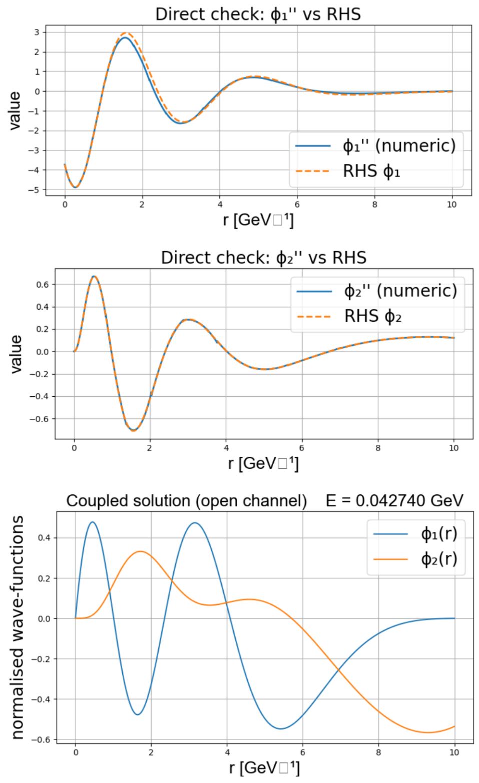
In order to compute the energy corrections from the coupling between the quarkonium and heavy-mesons channel we have developed a numerical solver for the coupled Schrödinger equations. An iterative algorithm is employed to solve the system, and it can be found in Appendix B.

The system of coupled Schrödinger equations is solved using an iterative numerical algorithm that combines integration and root-finding techniques. The first step of this method involves solving the one-channel Schrödinger equation for the Cornell potential (channel 1), using a shooting method with bisection to find the unperturbed energy level for a given number of nodes. Once it is obtained $E_{cornell}$ and depending on the sign of this energy, the nature of the coupling is handled differently:

- If $E_{cornell} < 0$, the system is in a closed-channel region and $u_2(r)$ must decay exponentially. This is achieved by combining the particular and homogeneous solutions of the second equation and removing the noise at large distances due to the interpolation.

- If $E_{cornell} > 0$, the system is in a open-channel region and $u_2(r)$ is initialized with a spherical bessel function $j_l(k,r)$, where $k = \sqrt{2(\mu_B E_{cornell)} + \Delta)}$. This accounts for the correct continuum energy in the heavy-meson channel and ensures the asymptotic oscillatory behavior $u_2(r)$ is physically accurate.

FIG. 4: Sample plots generated by the coupled-channel Schrödinger solver, showing its numerical accuracy and output wavefunctions.



In both cases, a coupled-channel iteration is performed:

1. The radial wavefunction $u_1(r)$ interpolated is used as a source term $(V_{mix}(r) \cdot u_1(r))$ in the equation of $u_2(r)$, which is solved using the updated physical energy $E_2 = E + \Delta$ to account for the different threshold.

2. The resulting $u_2(r)$ is then used to update the equation of $u_1(r)$, including the back-reaction effects.

3. A new energy eigenvalue is computed by searching for a solution with the correct number of nodes, this process of iteration is repeated until energy convergence is achieved.

The iterative scheme ensures the physical consistency of the system, normalization and asymptotic decay conditions are guaranteed. Diagnostic plots are also generated comparing the second derivative of the wavefunctions $u_1''(r)$ and $u_2''(r)$ with the right-hand sides of their respective equation. Finally a figure of $u_1(r)$ and $u_2(r)$ for the new energy eigenvalue is plotted FIG.4.

The implementation is modular and allows for tuning the model parameters, angular momentum l and number of nodes .

The spectrum computed with this approximation is shown in the next table:

TABLE III: Results obtained by solving the two coupled Schrödinger equations.

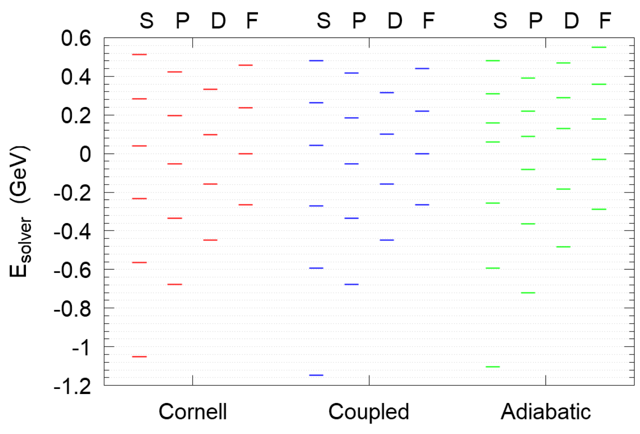
State	E_{solver} **(GeV)**	**State**	E_{solver} **(GeV)**
E1S	-1.296	E3D	0.131
E1P	-0.814	E3F	0.249
E1D	-0.439	E4S	0.065
E1F	-0.262	E4P	0.299
E2S	-0.556	E4D	0.359
E2P	-0.391	E4F	0.469
E2D	-0.177	E5S	0.303
E2F	0.004	E5P	0.517
E3S	-0.254	E6S	0.537
E3P	-0.103		

III. DISCUSSION.

We will start this section discussing the results obtained for the Upsilon spectrum using the Cornell potential. The obtained data show a remarkable agreement with the tabulated energy levels reported by the Particle Data Group (PDG). In particular, most of the states (S, P, D) are accurately computed within a precision of $\pm 0.01\ GeV$, this shows the effectiveness of the potential form used from [2].
However, to align the absolute energy scale of the calculated spectrum with the experimental one, a new additive constant c in the potential was required. This constant term in the potential acts as an energy reference and does not affect the wavefunctions or the space between the energy levels.
Moreover, the numerical solver predicts additional excited states for $l > 2$ and higher n excitations, that have not been experimentally observed so far. These predictions provide interesting candidates for future experimental verification and could be useful for future measurements or lattice QCD studies.

We now move on to explore how the inclusion of the mixing with virtual $B\bar{B}$ pairs modify the spectrum. The adiabatic and diabatic (coupled-channel) approximations both capture essential features of the interaction between the bottomonium and the heavy-meson channel, in particular the string breaking phenomena.

FIG. 5: Energy levels, E_{solver} (GeV), for the Upsilon Spectrum "Cornell" (red) and for the two approximations with the effect of the BB meson, "coupled" (blue) and "adiabatic" (green).



In the adiabatic approximation by diagonalizing (4) we obtain two effective potentials, $V_{\pm}(r)$. The spectrum of $V_{-}(r)$ shows an overall downward energy shift compared to the Cornell potential, reflecting an effective screening from virtual $B\bar{B}$ pairs. In contrast levels from $V_{+}(r)$ appear at much higher energies, although numerical solutions exist for a given number of nodes, these states are likely to correspond to higher radial excitations of the Cornell potential. Moreover, we observe additional states that do not correspond to any states found on the others spectra such as states $2S$ and $1P$ as we can see in FIG.6.

The coupled-channel approximation incorporates mixing dynamically and produces a spectrum that aligns well with the Cornell results at low energies, while introducing visible shifts at higher energies. These deviations, consistent with the onset of the string breaking, indicate a more realistic treatment of the meson-pair coupling.

All the numerical methods developed in this work are not limited to the bottomonium system. In particular we could have extended to study the charmonium spectrum, where similar mixing effects with virtual $D\bar{D}$ meson pairs can occur. Given that the charm quark mass is lower ($m_c = 1.496\ GeV$), mixing effects and threshold proximity may be even more pronounced. However the non-relativistic approximation becomes less accurate in this regime, so in the charmonium system lies near the boundary of the theoretical framework used in this study.

IV. CONCLUSIONS

- In this work, two non-trivial numerical solvers have been successfully developed, the first one with the objective of solving the radial Schrödinger equation for arbitrary potential and the second one to address the more complex case of coupled differential equations. The implementation applies a high level of numerical and physical rigor, using techniques as node-counting, energy root-finding and asymptotic validations. This has enabled the identification and analysis of bound states under various potential configurations.
- The spectra obtained both the Cornell potential and the diabatic approximation (via coupled-channel equations) show significant agreement with the tabulated experimental data.
- Although the adiabatic approximation provides useful information below threshold, it does not offer an accurate description of the dynamics beyond the threshold, where coupled channel and open channel effects become essential. Nevertheless, this approximation is still widely used in recent studies [6] and [7].

Acknowledgments

I would like to thank my advisor for offering this challenge and for the guidance during the study. I would also like to thank my parents and Iria for their patience and support. Thanks to my friends Pau, Nico and Nadir.

[1] N. Brambilla, A. Pineda, J. Soto and A. Vairo, Rev. Mod. Phys. **77** (2005), 1423 doi:10.1103/RevModPhys.77.1423 [arXiv:hep-ph/0410047 [hep-ph]].
[2] Pau Peñaranda Gavlak, Dipòsit Digital de la Universitat de Barcelona (2018), https://hdl.handle.net/2445/125152.
[3] W. Lucha and F. F. Schoberl, Int. J. Mod. Phys. C **10** (1999), 607-620 doi:10.1142/S0129183199000450 [arXiv:hep-ph/9811453 [hep-ph]].
[4] J. Tarrús Castellà, JHEP **06** (2024), 107 doi:10.1007/JHEP06(2024)107 [arXiv:2401.13393 [hep-ph]].
[5] G. S. Bali *et al.* [SESAM], Phys. Rev. D **71** (2005), 114513 doi:10.1103/PhysRevD.71.114513 [arXiv:hep-lat/0505012 [hep-lat]].
[6] E. Braaten and R. Bruschini, Phys. Lett. B **863** (2025), 139386 doi:10.1016/j.physletb.2025.139386 [arXiv:2409.08002 [hep-ph]].
[7] N. Brambilla, A. Mohapatra, T. Scirpa and A. Vairo, [arXiv:2411.14306 [hep-ph]].

L'efecte de parelles de mesons B virtuals en l'espectre de l'Upsilon.

Author: Adrián Asensio Magariños
Facultat de Física, Universitat de Barcelona, Diagonal 645, 08028 Barcelona, Spain.

Advisor: Joan Soto Riera

Resum: Aquest treball estudia l'impacte de les parelles de mesons $B\bar{B}$ virtuals en l'espectre de l'Upsilon (Υ). Primer, es resol l'equació radial de Schrödinger amb un potencial de Cornell per calcular l'espectre, reproduint amb precisió els nivells coneguts. A continuació, s'analitza l'efecte de "string breaking" mitjançant les aproximacions adiabàtica i de canals acoblats. L'aproximació adiabàtica fa servir diagonalització per obtenir potencials efectius, mentre que el formalisme de canals acoblats resol el sistema de dues equacions de Schrödinger per modelar la barreja dinàmica. Els solvers numèrics desenvolupats mostren que les parelles de mesons virtuals alteren els estats d'energia. Aquests resultats ofereixen prediccions teòriques rellevants per a futurs experiments i simulacions de lattice QCD .

Paraules clau: Equació de Schrödinger, funció d'ona, quark, mesó, espectre energètic, mètode de tir.
ODSs: 4. Educació de qualitat

Objectius de Desenvolupament Sostenible (ODSs o SDGs)

1. Fi de la es desigualtats		10. Reducció de les desigualtats	
2. Fam zero		11. Ciutats i comunitats sostenibles	
3. Salut i benestar		12. Consum i producció responsables	
4. Educació de qualitat	X	13. Acció climàtica	
5. Igualtat de gènere		14. Vida submarina	
6. Aigua neta i sanejament		15. Vida terrestre	
7. Energia neta i sostenible		16. Pau, justícia i institucions sòlides	
8. Treball digne i creixement econòmic		17. Aliança pels objectius	
9. Indústria, innovació, infraestructures			

Appendix A: Python Code: Schrödinger Solver with Cornell Potential

The following Python script numerically solves the one-channel Schrödinger equation for the Cornell potential, as used in this work.

```
# Cornell Potential Solver using the Radial
    Schr dinger Equation
#
# Author : Adrian Asensio
# Date   : 2025-05
#
# Quick use:
#     $ python solver_cornell.py
#
# Description:
#     This program numerically solves the radial
     Schr dinger equation for a
#     quark-antiquark system using the Cornell
    potential:
#
#         V(r) = -A/r + b*r + c
#
#     The user provides quark masses, angular
    momentum (l), and the number
#     of desired nodes in the wavefunction. The
    program searches for bound
#     state energies using a shooting method
    with automatic node counting,
#     and normalizes the resulting radial
    wavefunction u(r).
#
# Behavior:
#     - If a bound state energy is found, the
    wavefunction is solved and normalized.
#     - A robust auto-detection removes the
    unphysical tail of the wavefunction.
#     - The code checks that the asymptotic
    decay is physically consistent.
#
# Outputs:
#     - Energy eigenvalue for the selected bound
     state.
#     - Plot of the normalized radial
    wavefunction u(r).
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
from scipy.optimize import bisect

# --- Cornell potential parameters ---
A = 0.369
b = 0.19
c = -1.13
hbar = 1.0

# --- User inputs ---
print(" Physical parameters:")
m1 = float(input(" Mass of the first quark (GeV/
    c^2): "))
m2 = float(input(" Mass of the second quark (GeV
    /c^2): "))
l = int(input(" Angular momentum l: "))
n_nodes = int(input(" Desired number of nodes: "
    ))
E1 = float(input(" Lower energy bound (GeV): "))
E2 = float(input(" Upper energy bound (GeV): "))
# --- Reduced mass ---
mu = (m1 * m2) / (m1 + m2)
print(f"Reduced mass:    = {mu:.6f} GeV/c^2")

# --- Radial domain ---
r_min, r_max = 0.001, 10
N = 2000
r_eval = np.linspace(r_min, r_max, N)

# --- Effective potential ---
def V(r):
    return -A / r + b * r + c

def schrodinger(r, y, E):
    u, uprime = y
    Veff = V(r) + hbar**2 * l * (l + 1) / (2 *
        mu * r**2)
    d2u = 2 * mu / hbar**2 * (Veff - E) * u
    return [uprime, d2u]

# Shooting method for a given energy
def shoot(E):
    sol = solve_ivp(
        lambda r, y: schrodinger(r, y, E),
        [r_min, r_max],
        [0.0, 1.0],
        t_eval=[r_max]
    )
    return sol.y[0, -1]

# --- Root finding ---
E_vals = np.linspace(E1, E2, 500)
u_boundary = []

for E in E_vals:
    try:
        val = shoot(E)
        u_boundary.append(val)
    except Exception:
        u_boundary.append(np.nan)

# Search for bound states with given number of
    nodes
def find_states(n_nodes, E_vals):
    states = []
    signs = np.sign([shoot(E) for E in E_vals])
    for i in range(len(signs) - 1):
        if signs[i] != signs[i + 1]:
            E_low = E_vals[i]
            E_high = E_vals[i + 1]
            try:
                E_root = bisect(shoot, E_low,
                    E_high, xtol=1e-8)
                states.append(E_root)
                if len(states) > n_nodes:
                    break
            except ValueError:
                continue
    return states

# --- Find energies ---
energies = find_states(n_nodes, E_vals)

if len(energies) > n_nodes:
    E_found = energies[n_nodes]
    print(f" Energy for state with {n_nodes}
        nodes: E = {E_found:.8f} GeV")

    # --- Solve full wave function ---
    sol_final = solve_ivp(
        lambda r, y: schrodinger(r, y, E_found),
        [r_min, r_max],
        [0.0, 1.0],
        t_eval=r_eval
    )
    u_final = sol_final.y[0]

    # --- Automatically detect the correct
        physical tail ---
    small_threshold = 1e-4
    window = 20  # number of points to check
        stability
    relative_tolerance = 0.05  # 5% relative
        variation allowed

    tail_index = None

    for i in range(len(r_eval) - window):
        segment = u_final[i:i+window]
        if np.all(np.abs(segment) <
            small_threshold):
            rel_var = np.max(np.abs(np.diff(
                segment))) / np.max(np.abs(
                segment))
            if rel_var < relative_tolerance:
                tail_index = i
                break

    if tail_index is not None:
        # Truncate the wave function to zero
        u_final[tail_index:] = 0.0

        # Normalize only up to that point
        r_eval_cut = r_eval[:tail_index]
        u_final_cut = u_final[:tail_index]

        norm = np.sqrt(np.trapz(u_final_cut**2,
            r_eval_cut))
        u_final /= norm
    else:
        norm = np.sqrt(np.trapz(u_final**2,
            r_eval))
        u_final /= norm

    # --- Asymptotic check (tail in last 100
        points) ---
    asymptotic_tolerance = 1e-2
    tail = np.abs(u_final[-100:])
    if np.any(tail > asymptotic_tolerance):
        print(" PHYSICAL ERROR: The wave
            function does NOT decay to zero at
            the tail.")
    else:
        print(" The wave function correctly
            decays to zero at the tail.")

else:
    print("Not enough bound states found in the
        specified energy range.")
    u_final = np.zeros_like(r_eval)

# --- Plot normalized wave function ---
plt.figure(figsize=(8, 5))
plt.plot(r_eval, u_final, label=fr"$u(r)$, $E =
    {E_found:.8f}$ GeV")
plt.title(f"Normalized Wave Function - {n_nodes}
     nodes, $l = {l}$ (robust auto-cut)")
plt.xlabel("r [fm]")
plt.ylabel("u(r) (normalized)")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

Listing 1: Schrödinger solver for the Cornell potential

Appendix B: Python Code: two-channel coupled Schrödinger equations solver.

The following Python script numerically solves the two-channel coupled Schrödinger equations.

```
# C o u p l e d channel  solver for the Cornell
    potential (closed and open channels).

#Author : Adrian Asensio
#Date   : 2025-05

#Quick use

# $ python solver_cornell_coupled.py

#The program asks for the physical parameters,
    computes the uncoupled Cornell
#state, and then decides automatically:

# * If the Cornell energy is negative
    closed channel iteration with
  # exponential tail suppression.
# * If the Cornell energy is positive      open
    channel iteration starting
  # from a spherical-Bessel guess for      .

#The code produces diagnostic plots of the
    differential-equation residuals
#and the final, normalised wave-functions
    and       .

import numpy as np
from scipy.integrate import solve_ivp
from scipy.optimize import bisect
from scipy.interpolate import interp1d,
    CubicSpline
from scipy.special import spherical_jn
import matplotlib.pyplot as plt

#
    --------------------------------------------------

# Global constants
#
    --------------------------------------------------

a, b, c = 0.369, 0.19, -1.13                 #
    Cornell parameters
ro0 = 0.275 / 0.1973                        #
    $r_o$ in  G e V
A     = 0.2                                  #
    Coupling strength
hbar  = 1.0                                  #
        = 1 (natural units)

#
    --------------------------------------------------

# Potentials
```

```
#
    ---------------------------------------------------------------------------------------------------------

def V_cornell(r: float) -> float:
    """Original Cornell potential."""
    return -a / r + b * r + c


def V11(r: float, l: int, mu1: float) -> float:
    return V_cornell(r) + l * (l + 1) / (2 * mu1
         * r ** 2)


def V22(r: float, l: int, mu2: float) -> float:
    return l * (l + 1) / (2 * mu2 * r ** 2)


def V12(r: float) -> float:
    term1 = 8 * A ** 2 * r * (ro0 / (ro0 + r))
         ** 5
    term2 = (0.9 / (A ** 2 * r ** 3)) * (r / (
        ro0 + r)) ** 5
    return term1 + term2

#
    ---------------------------------------------------------------------------------------------------------
# Normalise
#
    ---------------------------------------------------------------------------------------------------------

def normalise(f: np.ndarray, r: np.ndarray) ->
    np.ndarray:
    """Return the L  -normalised array    fdr
        = 1."""
    norm = np.sqrt(np.trapz(f ** 2, r))
    return f / norm if norm > 0 else f

#
    ------------------------------------------------------------------------------------
# Uncoupled Cornell channel (      only)
#
    ------------------------------------------------------------------------------------

def schrodinger_cornell(r, y, E, l, mu1):
    u, up = y
    d2u = 2 * mu1 / hbar ** 2 * (V11(r, l, mu1)
        - E) * u
    return [up, d2u]


def shoot_cornell(E, l, mu1, r_min, r_max):
    u0, up0 = r_min ** (l + 1), (l + 1) * r_min
         ** l
    sol = solve_ivp(lambda r, y:
        schrodinger_cornell(r, y, E, l, mu1),
                    [r_min, r_max], [u0, up0],
                        t_eval=[r_max])
    return sol.y[0, -1]


def find_cornell_state(E_grid, l, mu1, r_min,
    r_max, n_nodes):
    sign   = np.sign([shoot_cornell(E, l, mu1,
        r_min, r_max) for E in E_grid])
    roots  = []
    for i in range(len(sign) - 1):
        if sign[i] != sign[i + 1]:
            try:
                root = bisect(lambda E:
                    shoot_cornell(E, l, mu1,
                    r_min, r_max),
                              E_grid[i], E_grid[
                                  i + 1], xtol=1
                                  e-8)
                roots.append(root)
                if len(roots) > n_nodes:
                    break
            except ValueError:
                pass
    return roots[n_nodes] if len(roots) >
        n_nodes else None


def solve_phi1_cornell(E, r_eval, l, mu1):
    u0, up0 = r_eval[0] ** (l + 1), (l + 1) *
        r_eval[0] ** l
    sol = solve_ivp(lambda r, y:
        schrodinger_cornell(r, y, E, l, mu1),
                    [r_eval[0], r_eval[-1]], [u0
                        , up0], t_eval=r_eval)
    return sol.y[0]

#
    ---------------------------------------------------------------------------------------------------------
# Channel-2 solver (E < 0)
#
    ---------------------------------------------------------------------------------------------------------

def schrodinger_phi2(r, y, phi1_i, E, l, mu2):
    phi2, phi2p = y
    d2phi2 = 2 * mu2 * ((V22(r, l, mu2) - E) *
        phi2 + V12(r) * phi1_i(r))
    return [phi2p, d2phi2]


def schrodinger_phi2_hom(r, y, E, l, mu2):
    phi2, phi2p = y
    d2phi2 = 2 * mu2 * (V22(r, l, mu2) - E) *
        phi2
    return [phi2p, d2phi2]


def phi2_total(phi1_arr, r_eval, E, l, mu2):
    phi1_i = interp1d(r_eval, phi1_arr, kind="
        cubic", fill_value="extrapolate")
    y0 = [r_eval[0] ** (l + 1), (l + 1) * r_eval
        [0] ** l]
    sol = solve_ivp(lambda r, y:
        schrodinger_phi2(r, y, phi1_i, E, l, mu2
        ),
                    [r_eval[0], r_eval[-1]], y0,
                        t_eval=r_eval)
    return sol.y[0]


def phi2_homogeneous(r_eval, E, l, mu2):
    y0 = [r_eval[0] ** (l + 1), (l + 1) * r_eval
        [0] ** l]
    sol = solve_ivp(lambda r, y:
        schrodinger_phi2_hom(r, y, E, l, mu2),
                    [r_eval[0], r_eval[-1]], y0,
                        t_eval=r_eval)
    return sol.y[0]


def damp_growing_mode(phi_tot, phi_hom, window
    =500):
```

```
    tail = slice(-window, None)
    c = np.dot(phi_tot[tail], phi_hom[tail]) /
        np.dot(phi_hom[tail], phi_hom[tail])
    print(f"   cancelling growing mode with
        factor c = {c:.3e}")
    return phi_tot - c * phi_hom


def phi2_physical_Eneg(phi1_arr, r_eval, E, l,
    mu2):
    phi_tot = phi2_total(phi1_arr, r_eval, E, l,
         mu2)
    phi_hom = phi2_homogeneous(r_eval, E, l, mu2
        )
    phi_phys = damp_growing_mode(phi_tot,
        phi_hom)

    # smooth exponential cut-off
    r_cut = 6
    phi_phys *= np.exp(-((r_eval - r_cut) * (
        r_eval > r_cut))**2)
    return phi_phys

#
    ----------------------------------------------------------------------------

# Shared       solver
#
    ----------------------------------------------------------------------------

def schrodinger_phi1(r, y, phi2_i, E, l, mu1):
    phi1, phi1p = y
    d2phi1 = 2 * mu1 * ((V11(r, l, mu1) - E) *
        phi1 + V12(r) * phi2_i(r))
    return [phi1p, d2phi1]


def shoot_phi1(E, phi2_i, l, mu1, r_min, r_max):
    y0 = [r_min ** (l + 1), (l + 1) * r_min ** l
        ]
    sol = solve_ivp(lambda r, y:
        schrodinger_phi1(r, y, phi2_i, E, l, mu1
        ),
                    [r_min, r_max], y0, t_eval=[
                        r_max])
    return sol.y[0, -1]


def find_phi1_state(E_grid, phi2_i, l, mu1,
    r_min, r_max, n_nodes):
    sign   = np.sign([shoot_phi1(E, phi2_i, l,
        mu1, r_min, r_max) for E in E_grid])
    roots  = []
    for i in range(len(sign) - 1):
        if sign[i] != sign[i + 1]:
            try:
                root = bisect(lambda E:
                    shoot_phi1(E, phi2_i, l, mu1
                    , r_min, r_max),
                              E_grid[i], E_grid[
                                  i + 1], xtol=1
                                  e-8)
                roots.append(root)
                if len(roots) > n_nodes:
                    break
            except ValueError:
                pass
    return roots[n_nodes] if len(roots) >
        n_nodes else None
```

```
#
    ----------------------------------------------------------------------------

# Closed-channel iteration (E < 0)
#
    ----------------------------------------------------------------------------

def iterate_closed(phi1_0, E_0, r, n_iter, l,
    mu1, mu2, E_grid, n_nodes,delta_E):
    phi1 = normalise(phi1_0.copy(), r)
    E    = E_0
    for k in range(n_iter):
        print(f" iteration {k + 1}")
        phi2 = phi2_physical_Eneg(phi1, r, E +
            delta_E, l, mu2)
        print(f"   max|      | = {np.max(np.abs(
            phi2)):.3e}")

        phi2_i = interp1d(r, phi2, kind="cubic",
             fill_value="extrapolate")
        E_new  = find_phi1_state(E_grid, phi2_i,
             l, mu1, r[0], r[-1], n_nodes)
        if E_new is None:
            print("   no new root found
                stopping")
            break
        print(f"   new energy  E = {E_new:.8f}
            GeV")

        y0   = [r[0] ** (l + 1), (l + 1) * r[0]
            ** l]
        sol  = solve_ivp(lambda rr, y:
            schrodinger_phi1(rr, y, phi2_i,
            E_new, l, mu1),
                         [r[0], r[-1]], y0,
                             t_eval=r)
        phi1 = normalise(sol.y[0], r)
        E    = E_new
    return phi1, phi2, E


#
    ----------------------------------------------------------------------------

# Open-channel iteration (E > 0)
#
    ----------------------------------------------------------------------------

def phi2_bessel(r, E, l, mu2):
    k = np.sqrt(2 * mu2 * E)
    return np.sqrt(2 / np.pi) * spherical_jn(l,
        k * r)


def solve_phi1(E, r, phi2_i, l, mu1):
    y0  = [r[0] ** (l + 1), (l + 1) * r[0] ** l]
    sol = solve_ivp(lambda rr, y:
        schrodinger_phi1(rr, y, phi2_i, E, l,
        mu1),
                        [r[0], r[-1]], y0, t_eval=r)
    return sol.y[0]


def solve_phi2_open(phi1_arr, r, E, l, mu2):
    phi1_i = CubicSpline(r, phi1_arr,
        extrapolate=True)
    sol = solve_ivp(lambda rr, y:
        schrodinger_phi2(rr, y, phi1_i, E, l,
        mu2),
                        [r[0], r[-1]], [0.0, 0.0],
                            t_eval=r)
    return sol.y[0]


def iterate_open(E_0, r, n_iter, l, mu1, mu2,
    E_grid, n_nodes,delta_E):
    print(" initialising                with a
        spherical-Bessel function")
    phi2 = phi2_bessel(r, E_0+delta_E, l, mu2)
    E    = E_0
    for k in range(n_iter):
        print(f"\n iteration {k + 1}")
        phi2_i = CubicSpline(r, phi2,
            extrapolate=True)
        E_new  = find_phi1_state(E_grid, phi2_i,
             l, mu1, r[0], r[-1], n_nodes)
        if E_new is None:
            print("   no bound state for
                    aborting")
            break
        print(f"   new energy  E = {E_new:.8f}
            GeV")
        E   = E_new
        phi1 = normalise(solve_phi1(E, r, phi2_i
            , l, mu1), r)
        phi2 = solve_phi2_open(phi1, r, E +
            delta_E, l, mu2)
        print(f"   max|      | = {np.max(np.abs(
            phi2)):.2e}")
    return phi1, phi2, E

#
    ----------------------------------------------------------------------

# Diagnostics and plots
#
    ----------------------------------------------------------------------

def diagnostic_plots(r, phi1, phi2, E, l, mu1,
    mu2, title,delta_E):
    spl1 = CubicSpline(r, phi1); spl2 =
        CubicSpline(r, phi2)
    d2_1 = spl1(r, 2);            d2_2 = spl2(r,
         2)

    rhs1 = 2 * mu1 * (V11(r, l, mu1) - E) * phi1
         + 2 * mu1 * V12(r) * phi2
    rhs2 = 2 * mu2 * (V22(r, l, mu2) - (E+
        delta_E)) * phi2 + 2 * mu2 * V12(r) *
        phi1

    #      ''
    plt.figure(figsize=(8, 4))
    plt.plot(r, d2_1, label="     '' (numeric)",
         linewidth=2)
    plt.plot(r, rhs1, "--", label="RHS      ",
        linewidth=2)
    plt.xlabel("r [ G e V  ]"); plt.ylabel("
        value")
    plt.title("Direct check:      '' vs RHS")
    plt.legend(); plt.grid(); plt.tight_layout()
        ; plt.show()

    #      ''
    plt.figure(figsize=(8, 4))
    plt.plot(r, d2_2, label="     '' (numeric)",
         linewidth=2)
    plt.plot(r, rhs2, "--", label="RHS      ",
        linewidth=2)
    plt.xlabel("r [ G e V  ]"); plt.ylabel("
        value")
    plt.title("Direct check:      '' vs RHS")
    plt.legend(); plt.grid(); plt.tight_layout()
        ; plt.show()

    # wave-functions
    plt.figure(figsize=(8, 5))
    plt.plot(r, phi1, label="     (r)")
    plt.plot(r, phi2, label="     (r)")
    plt.xlabel("r [ G e V  ]"); plt.ylabel("
        normalised wave-functions")
    plt.title(title)
    plt.legend(); plt.grid(); plt.tight_layout()
        ; plt.show()

#
    ----------------------------------------------------------------------

# Main program
#
    ----------------------------------------------------------------------

def main():
    print("\n=== Input physical parameters ===")
    m1 = float(input("Mass of the first quark  (
        GeV/c^2): "))
    m2 = float(input("Mass of the second quark (
        GeV/c^2): "))
    mB = float(input("Mass of the B meson      (
        GeV):      "))
    l  = int  (input("Orbital angular momentum l
        :          "))
    n_nodes = int(input("Desired number of
        radial nodes:      "))
    E_min = float(input("Energy search lower
        bound (GeV):     "))
    E_max = float(input("Energy search upper
        bound (GeV):     "))

    mu1 = m1 * m2 / (m1 + m2)
    mu2 = mB / 2.0
    delta_E = 2.0 * (mB - m2)

    r_min, r_max, N = 0.001, 14, 4000
    r      = np.linspace(r_min, r_max, N)
    E_grid = np.linspace(E_min, E_max, 500)

    # 1) Uncoupled Cornell
    print("\nSolving the uncoupled Cornell
        problem ...")
    E_cornell = find_cornell_state(E_grid, l,
        mu1, r_min, r_max, n_nodes)
    if E_cornell is None:
        print("No state with that node count in
            the requested range.")
        return

    phi1_corn = normalise(solve_phi1_cornell(
        E_cornell, r, l, mu1), r)
    print(f"Uncoupled Cornell energy  E = {
        E_cornell:.8f} GeV")

    # 2) Choose iteration scheme by the sign of
        E_cornell
    if E_cornell < 0.0:
        print("\nE < 0      closed channel
            iteration")
        phi1_f, phi2_f, E_f = iterate_closed(
            phi1_corn, E_cornell, r,
                                                      n_iter
                                                         =7,
                                              l=l
                                              ,
                                              mu1
                                              =
                                              mu1
                                              ,
                                              mu2
                                              =
                                              mu2
                                              ,
                                              E_grid
                                              =
                                              E_grid
                                              ,
                                              n_nodes
                                              =
                                              n_nodes
                                              ,
                                              delta_E
                                              =
                                              delta_E
                                              )
        print(f"\nFinal corrected energy  E = {
            E_f:.8f} GeV"
              f"   ( E  = {E_f - E_cornell:+.8f}
                  GeV)")
        diagnostic_plots(r, phi1_f, phi2_f, E_f,
             l, mu1, mu2,
                         title=f"Coupled
                             solution (closed
                             channel)  E = {E_f
                             :.6f} GeV",delta_E=
                             delta_E)
    else:
        print("\nE > 0      open channel
            iteration")
        phi1_f, phi2_f, E_f = iterate_open(
            E_cornell, r,
                                              n_iter
                                                 =5,
                                              l=l,
                                                 mu1
                                                 =
                                                 mu1
                                                 ,
                                                 mu2
                                                 =
                                                 mu2
                                                 ,
                                              E_grid
                                                 =
                                                 E_grid
                                                 ,
                                                 n_nodes
                                                 =
                                                 n_nodes
                                                 ,
                                                 delta_E
                                                 =
                                                 delta_E
                                                 )
        print(f"\nFinal coupled-channel energy
            E = {E_f:.8f} GeV")
        diagnostic_plots(r, phi1_f, phi2_f, E_f,
             l, mu1, mu2,
                         title=f"Coupled
                             solution (open
                             channel)   E = {
                             E_f:.6f} GeV",
                             delta_E=delta_E)


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

Listing 2: 2-channel coupled Schrödinger equations solver.