Animacja paczek falowych Rydberga: Symulacja ewolucji czasowej stanów koherentnych dla n=320

Mateusz Budzyński

February 8, 2026

0.0.1 Link do githuba: https://github.com/Budyn8/kwanty_coding/

1 Równanie funkcji falowej

W celu ułatwienia obliczeń należy przejść do współrzędnych biegunowych, gdzie dwuwymiarowy operator Laplasjanu (∇^2) przyjmuje postać:

$$\nabla^2 = \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2}{\partial \phi^2}$$

Równanie funkcji falowej niezależnej od czasu w układzie biegunowym:

$$-\frac{\hbar^2}{2\mu}\nabla^2\Psi + V\Psi = E\Psi$$

Zakładając separację zmiennych w postaci:

$$\Psi(r,\phi) = R(r)\Phi(\phi)$$

2 Separacja Zmiennych

Pełną funkcję falową $\Psi(r,\phi,t)$ można rozbić na składowe:

2.0.1 Równanie Radialne (używając podstawienia $u(r) = rR(r)$):

$$-\frac{\hbar^2}{2\mu}\frac{d^2u(r)}{dr^2} + V_{eff}(r)u(r) = Eu(r)$$

2.0.2 Równanie Kątowe:

$$\frac{d^2\Phi(\phi)}{d\phi^2} = -m^2\Phi(\phi)$$

3 Rozwiązania Równań i Przybliżenie Stanów Kołowych

Dla bardzo wysokich liczb kwantowych ($n \gg 1$), funkcje falowe atomu wodoru dążą do granicy klasycznej. W przypadku **stanów kołowych** (ang. *circular states*), gdzie orbitalna liczba kwantowa jest maksymalna ($l = n - 1$), rozwiązania równania $R(r)$ można przybliżyć do znacznie prostszej formy.

3.0.1 Rozwiązanie Kątowe

Rozwiązaniem równania kątowego jest funkcja wykładnicza o czysto urojonym argumencie:

$$\Phi(\phi) = e^{im\phi}$$

Dla stanów kołowych przyjmujemy $m = n - 1$.

3.0.2 Przybliżenie Radialne dla $n \approx 320$

Dokładne rozwiązanie równania radialnego wykorzystuje wielomiany Laguerre'a, jednak dla $n = 320$ są one numerycznie niemożliwe do bezpośredniego obliczenia (osiągają wartości rzędu 10^{600}). Stosujemy więc przybliżenie wynikające z asymptotycznej postaci funkcji dla maksymalnego l i jednostek atomowych ($Z = 1$):

$$R(r) \approx N \cdot r^{n-1} e^{-\frac{r}{n}}$$

W symulacji, aby uniknąć błędów przepełnienia (overflow), wartości te obliczono w skali logarytmicznej:

$$\ln(R(r)) \approx (n-1)\ln(r) - \frac{r}{n} + \ln(N)$$

3.0.3 Stała Normalizacyjna w skali logarytmicznej

Aby funkcja falowa była poprawnie znormalizowana (całkowite prawdopodobieństwo równe 1), należy uwzględnić stałą normalizacyjną N. Ze względu na operowanie na bardzo wysokich liczbach kwantowych, zastosowano przybliżenie Stirlinga dla silni, zapisane w formie logarytmicznej:

$$\ln(N) \approx -\frac{1}{2}\left[2n\ln(2n) - 2n + \frac{1}{2}\ln(4\pi n)\right] + (n + \frac{1}{2})\ln\left(\frac{2}{n}\right)$$

4 Sumaryczna Postać Paczki Falowej (Superpozycja)

Paczka falowa widoczna na animacji nie jest pojedynczym stanem stacjonarnym, lecz **superpozycją** wielu stanów własnych. Każdy składnik sumy jest modyfikowany przez wagę Gaussa w_n oraz czynnik ewolucji czasowej:

$$\Psi(r, \phi, t) = \sum_n w_n \cdot R_n(r) e^{i(n-1)\phi} e^{-iE_n t}$$

Gdzie energia (w j.u.) wynosi $E_n = -\frac{1}{2n^2}$.

Równanie wagi (w_n): W symulacji zastosowano rozkład Gaussa:

$$w_n = \exp\left(-\frac{(n-\bar{n})^2}{4\sigma^2}\right)$$

[ ]:
```
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.animation import FuncAnimation, FFMpegWriter
from IPython.display import HTML

sp.init_printing(use_latex='mathjax')
```

[ ]:
```
# import matplotlib
# matplotlib.rcParams['animation.embed_limit'] = matplotlib.rcParams['animation.
 ↪embed_limit'] * 10
# matplotlib.use('WebAgg')
```

[ ]:
```
n, n_avg, sigma = sp.symbols(r'n \bar{n} sigma')
phi, r, t = sp.symbols('phi r t')
R = sp.symbols('R(r)')
```

[ ]:
```
weight_expr = sp.exp(-(n - n_avg)**2 / (4*sigma**2))
weight_expr
```

[ ]:
```
angular_phase_expr = sp.exp(sp.I * (n - 1) * phi)
angular_phase_expr
```

[ ]:
```
time_phase_expr = sp.exp(sp.I * t / (2 * n**2))
time_phase_expr
```

[ ]:
```
Psi_total_expr = weight_expr * R * angular_phase_expr * time_phase_expr
Psi_total_expr
```

[ ]:
```
log_norm_expr = -1/2 * ((2*n)*sp.log(2*n) - (2*n) + 1/2*sp.log(2*sp.pi*2*n)) +␣
 ↪(n + 1/2)*sp.log(2/n)

log_radial_part_expr = (n - 1) * sp.log(r + 1e-10) - (r / n)

# COMBINED LOG EXPONENT
# Dostajemy całą stałą normalizującą przy czym dodajemy zamiast mnożyć bo są␣
 ↪zlogarytmowane
# Normalnie byśmy je mnożyli
combined_log_expr = log_radial_part_expr + log_norm_expr
combined_log_expr
```

[ ]:
```
# Stałe z artykułu
N_AVG = 320
SIGMA = 2.5
RESOLUTION = 300

Psi_total_lambda = sp.lambdify((r, phi, t, n, R), Psi_total_expr.subs({sigma :␣
 ↪SIGMA, n_avg : N_AVG}), 'numpy')
combined_log_lambda = sp.lambdify((r, n), combined_log_expr, 'numpy')

# poziomy energetyczne które będziemy uwzględniać na naszym wykresie
# ('3 * standardowe odczylenie zgarnia ~99.7% pakietów energetycznych' ~ Gemini)
n_range = np.arange(int(N_AVG - 4*SIGMA), int(N_AVG + 4*SIGMA) + 1)

# Wzięcie odległości R dla których prawdopodobieństwo ma sens (nie jest ~0)
r_min = N_AVG**2 - (40 * N_AVG)
r_max = N_AVG**2 + (40 * N_AVG)

r_vals = np.linspace(r_min, r_max, RESOLUTION)

phi_vals = np.linspace(0, 2*np.pi, RESOLUTION)

R, PHI = np.meshgrid(r_vals, phi_vals)

# Kepler Period T = 2 * pi * n^3
T_kepler = 2 * np.pi * (N_AVG**3)
# Przybliżony czas na powrót do stanu początkowego
T_rev = (N_AVG / 3) * T_kepler

def update(frame_fraction):
    ax.clear() # Clear the previous mountain

    Psi_total = np.zeros_like(R, dtype=np.complex128)

    X = R * np.sin(PHI)
    Y = R * np.cos(PHI)

    curr_t = frame_fraction*T_rev

    for n in n_range:
        c_1 = combined_log_lambda(R, n)
        c_max = np.max(c_1)

        Psi_total += Psi_total_lambda(R, PHI, curr_t, n, np.exp(c_1 - c_max))

    Prob_Density = np.abs(Psi_total)**2

    Z = Prob_Density/np.max(Prob_Density)
```

```
    # Redraw the surface
    surf = ax.plot_surface(X, Y, Z, cmap='CMRmap',
                           linewidth=0, antialiased=True,
                           rcount=500, ccount=300)

    ax.set_title(f"Rydberg Wave Packet (n={N_AVG}, FPS={FPS})\nTime =␣
 ↪{frame_fraction:.3f} $T_{{rev}}$")
    ax.set_zlim(0, np.max(Z) * 1.1)
    return surf,

# Stałe do animacji, przy SMOOTHNESS = 0.0117 zajmuje dość długo (nie wiem ile)
# ale jak się zmniejszy smoothness do np 0.1 to będzie się kompilpowało szybcie␣
 ↪ale kosztem słabszej animacji

# Liczba klatek na sekundę
FPS = 15

# Czas w jednostkach T_rev (czasu odrodzenia)
# 1 oznacza powrót do stanu początkowego, nie dokońca bo tak naprawdę to 0.99␣
 ↪ale blisko tego
FROM_TF = 0
TO_TF = 0.9883 # 0.9883 <- aby fajnie się zapętlało

# Parametr kontrolujący gęstość próbkowania czasu
SMOOTHNESS = 0.1 # 0.0117 <- aby animacja była ładna (zwiększając zmniejsza␣
 ↪się ilość odległość między t_i a t_(i-1))

fig, ax = plt.subplots(subplot_kw={"projection": "3d"}, figsize=(10, 8))

frames = np.linspace(FROM_TF, TO_TF, int((TO_TF - FROM_TF)/SMOOTHNESS * FPS))

ani = FuncAnimation(fig, update, frames=frames, interval=1000/FPS)

plt.close()
HTML(ani.to_jshtml())
```

5 Bibliografia

- **Gaeta, Z. D., & Stroud Jr, C. R.** (1990). *Classical and quantum-mechanical dynamics of a quasiclassical state of the hydrogen atom.* Physical Review A, 42(11), 6308–6319.