Python Quantum Mechanical Linear Harmonic Oscillator: Wave Functions and Roots (n = 0–10) by James Pate Williams, Jr., BA, BS, MSwE, PhD and Microsoft's Copilot an AI Agent and Tool

One of the earliest studied quantum mechanical systems with discrete eigenvalues and bound states is the quantum mechanical linear harmonic oscillator [1] [2] [3] [4] [5]. The Schrödinger equation is [2]:

$$-\frac{\hbar^2}{2m}\frac{d^2u}{dx^2}+\frac{1}{2}Kx^2u=Eu$$

Where the *h-bar* symbol is the reduced Planck's constant, *m* is the mass, *K* is the Hooke's Law constant, *E* is the total energy of the linear harmonic oscillator, *x* is the one-dimensional displacement and *u* is the wave function. The Schrödinger equation is derived from the classical Hamiltonian operator:

$$H=T+V=\frac{p^2}{2m}+\frac{1}{2}Kx^2=E$$

Merzbacher [4] uses Schrödinger's *psi* instead of Schiff's *u*. Also, Merzbacher utilizes *mu*, the reduced mass. Boehm takes an entirely different approach using ladder operators (creation and annihilation operators).

The well-known eigenfunctions are:

$$u_n(x)=N_nH_n(x)e^{-\frac{1}{2}\alpha^2x^2}$$

Where:

$$N_n=\left(\frac{\alpha}{\pi^{\frac{1}{2}}2^nn!}\right)^{\frac{1}{2}}$$

$$\alpha^4=\frac{mK}{\hbar^2}$$

We assume *alpha* is equal to one in the experimental Python source code. In the Python programs *Psi* is the complete wave function not just the Hermite polynomial.

References

[1] L. Pauling and J. E. B. Wilson, "Introduction to Quantum Mechanics with Applications to Chemistry Kindle Edition," [Online]. Available: https://amazon.com. [Accessed 24 May 2026].

[2] L. I. Schiff, "Chapter 4 Discrete Eigenvalues: Bound States Section 13 Linear Hamonic Oscillator," in *Quantum Mechanics Third Edition*, New York, McGraw-Hill Book Company, 1968, pp. 66-76.

[3] A. Boehm, "II Foundations of Quantum Mechanics - The Harmonic Oscillator," in *Quantum Mechanics*, New York, Springer-Verlog New York Inc. , 1979, pp. 9-63.

[4] E. Merzbacher, "5. The Linear Harmonic Operator," in *Quantum Mechanics Second Edition*, New York, John Wiley & Sons Inc., 1970, pp. 53-79.

[5] A. Messiah, "Chapter XII The Harmonic Oscillator," in *Quantum Mechanics Two Volumes Bound as One*, Mineola, Dover Publications, Inc., 2014, pp. 433-440.

Appendix A: Roots and Wavefunctions

The n = 0 wave function does not have any roots.

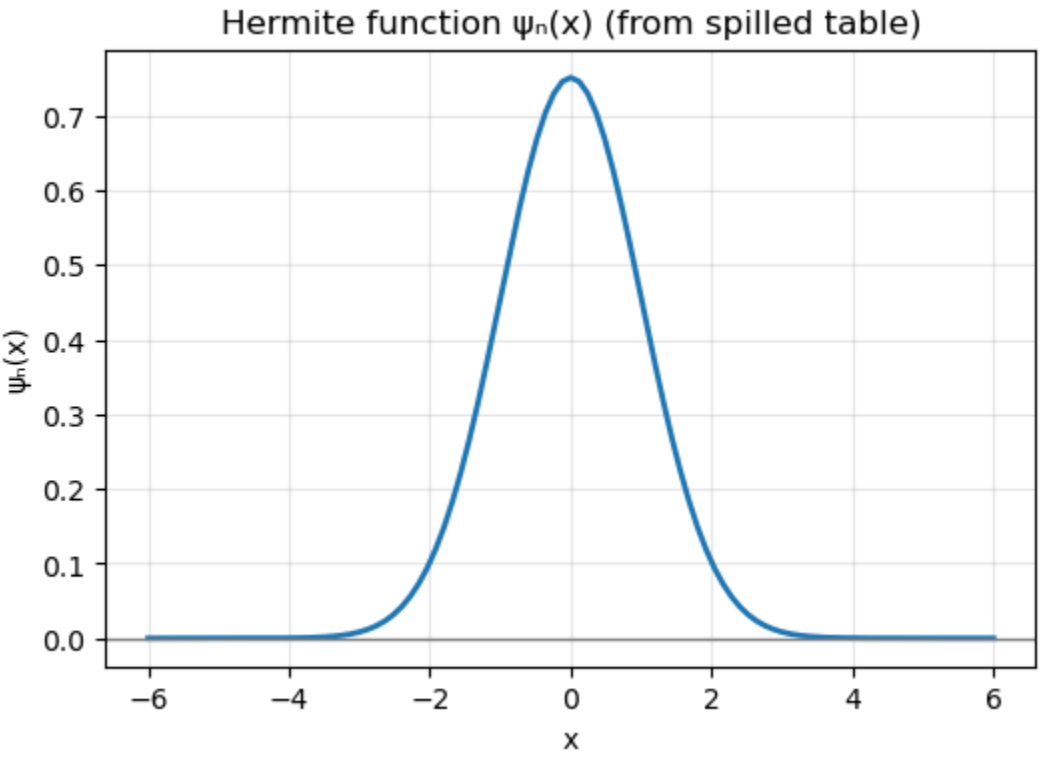

Figure 1 n = 0 Wave Function

The n = 1 Wave Function has the trivial root of 0. Note: the number of roots is equal to n, the quantum number.

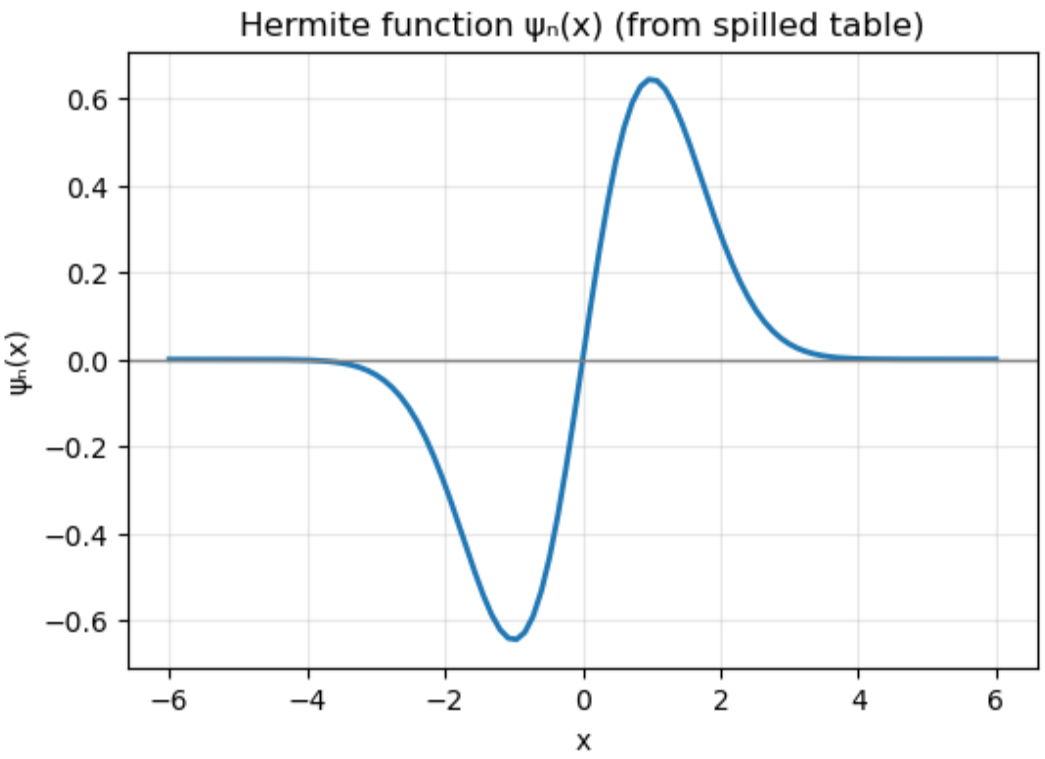

Figure 2 n = 1 Wave Function

roots(H_2)

-0.707106781

0.707106781

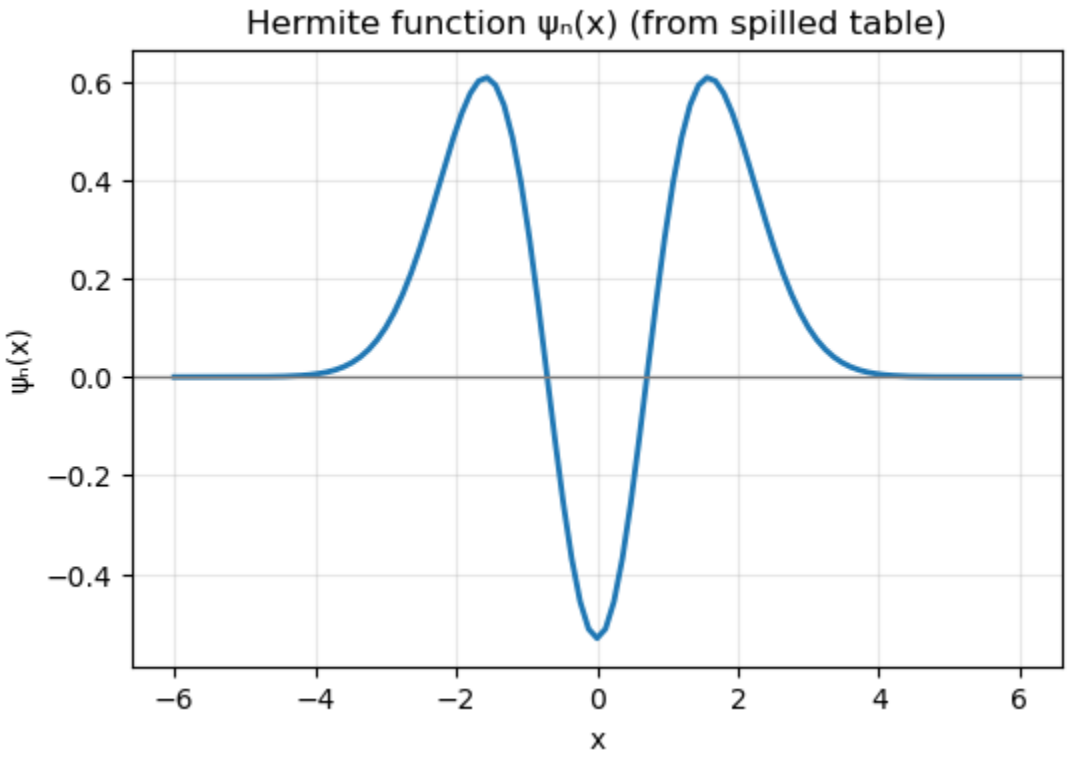

Figure 3 n = 2 Wave Function

roots(H_3)

-1.224744871

-4.24982E-17

1.224744871

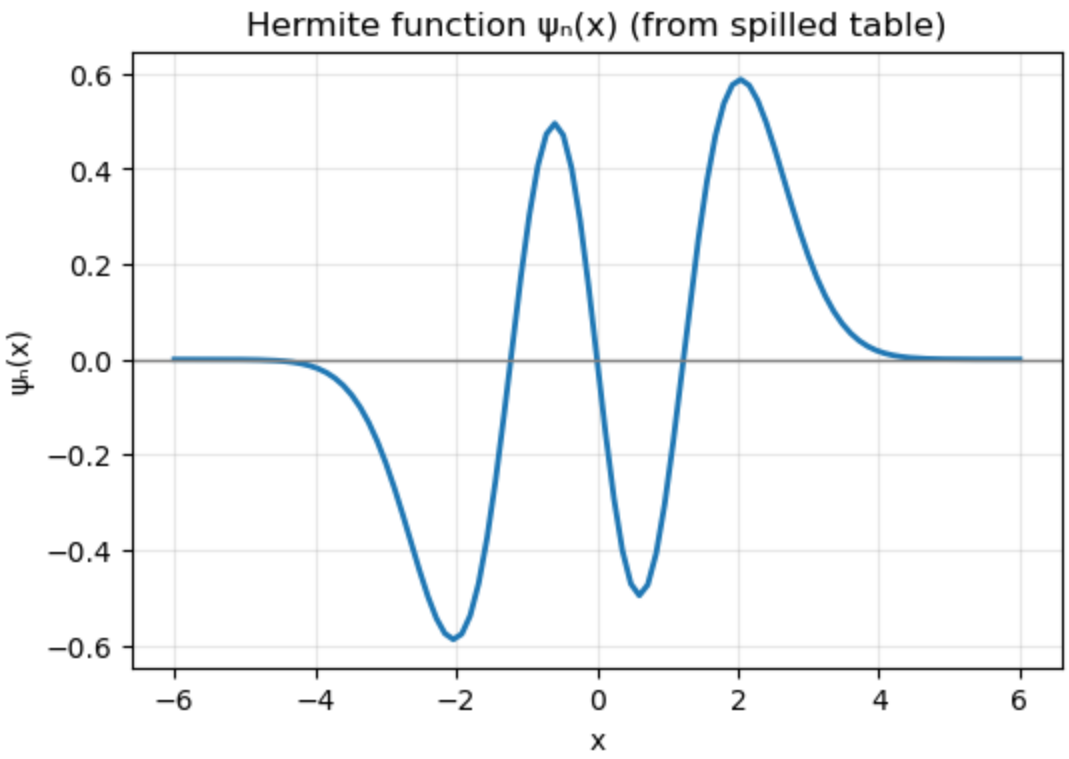

Figure 4 n = 3 Wave Function

roots(H_4)

-1.650680124

-0.524647623

0.524647623

1.650680124

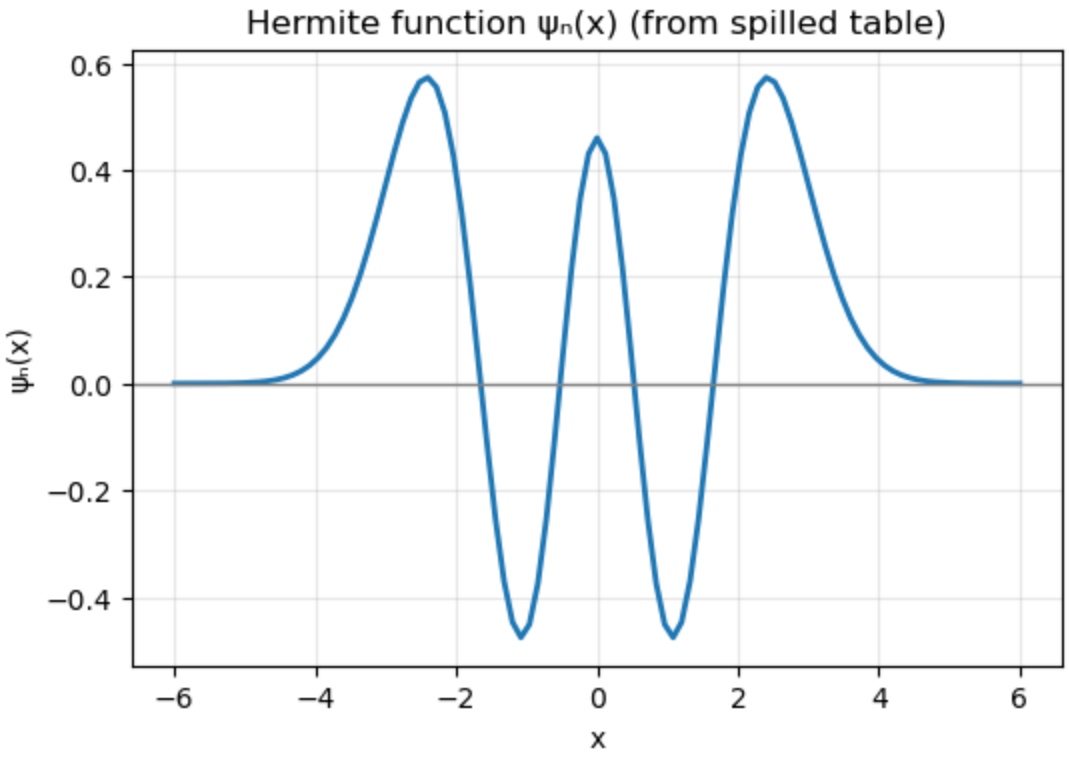

Figure 5 n = 4 Wave Function

roots(H_5)

-2.02018287
-0.958572465
5.15692E-17
0.958572465
2.02018287

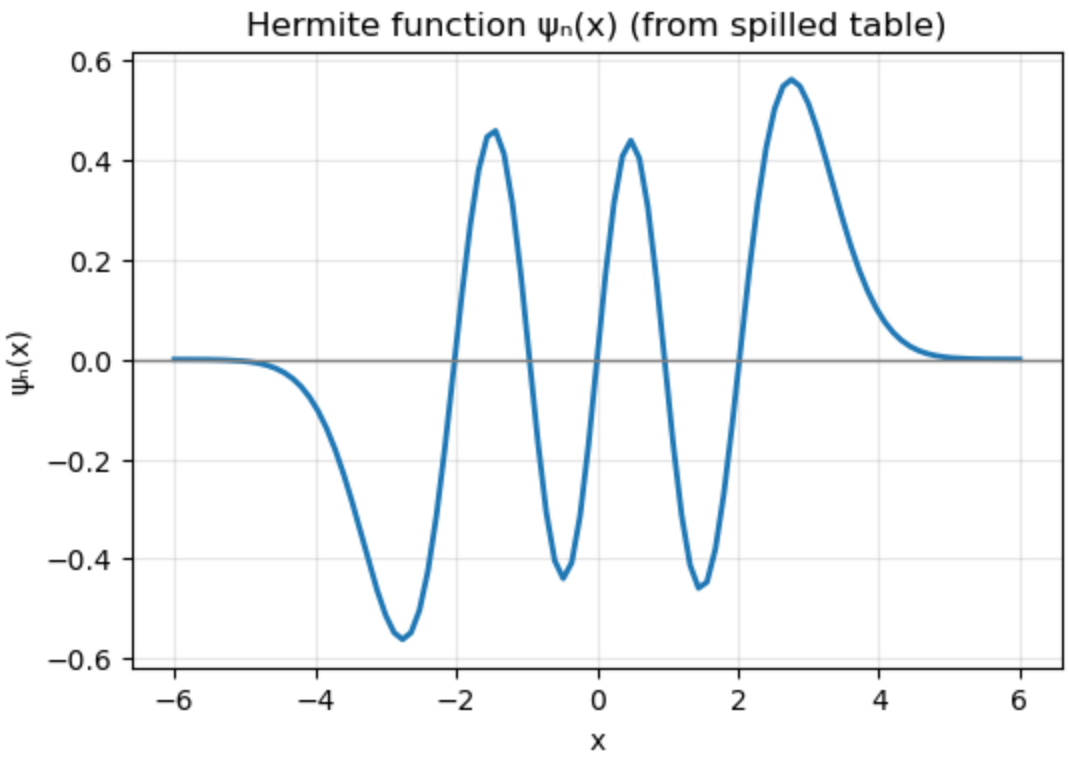

Figure 6 n = 5 Wave Function

roots(H_6)

-2.350604974
-1.335849074
-0.436077412
0.436077412
1.335849074
2.350604974

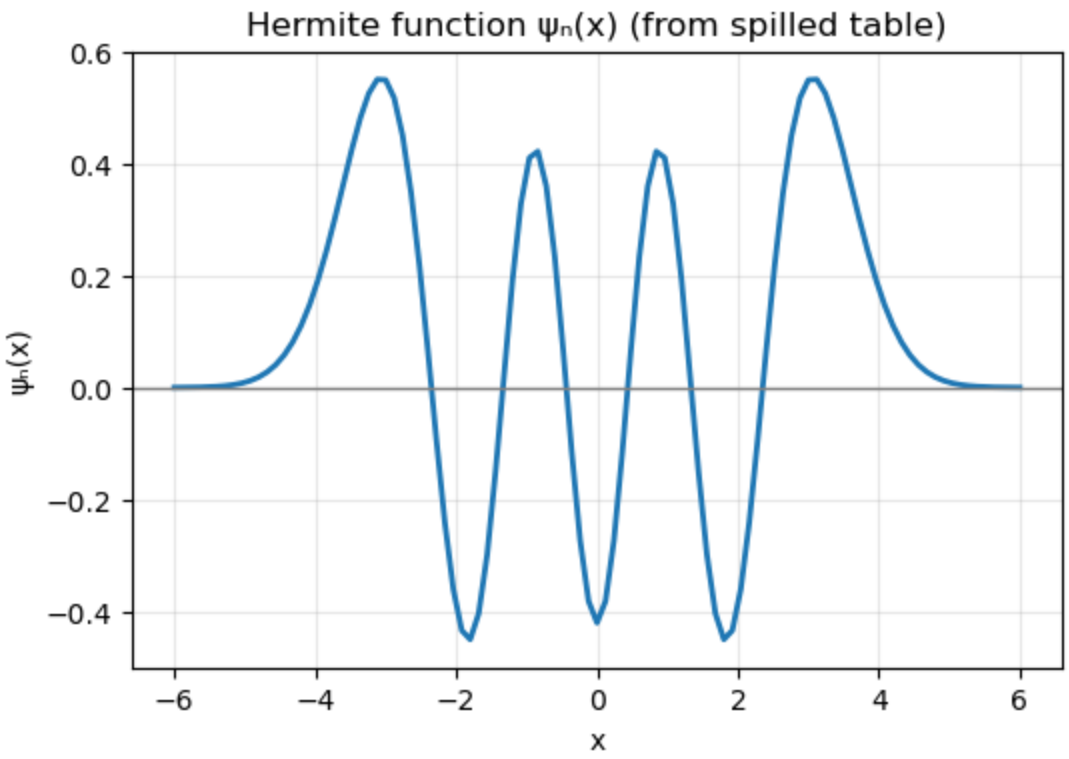

Figure 7 n = 6 Wave Function

roots(H_7)

-2.651961357
-1.673551629
-0.816287883
1.24386E-16
0.816287883
1.673551629
2.651961357

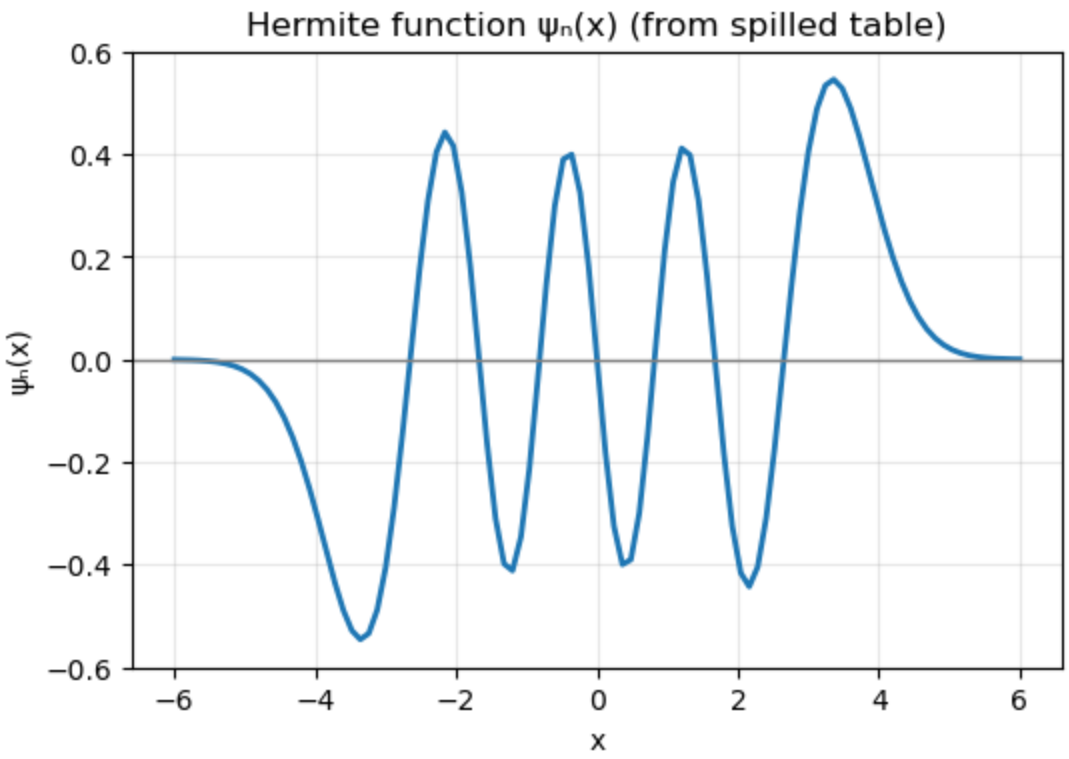

Figure 8 n = 7 Wave Function

roots(H_8)

-2.93063742
-1.981656757
-1.157193712
-0.38118699
0.38118699
1.157193712
1.981656757
2.93063742

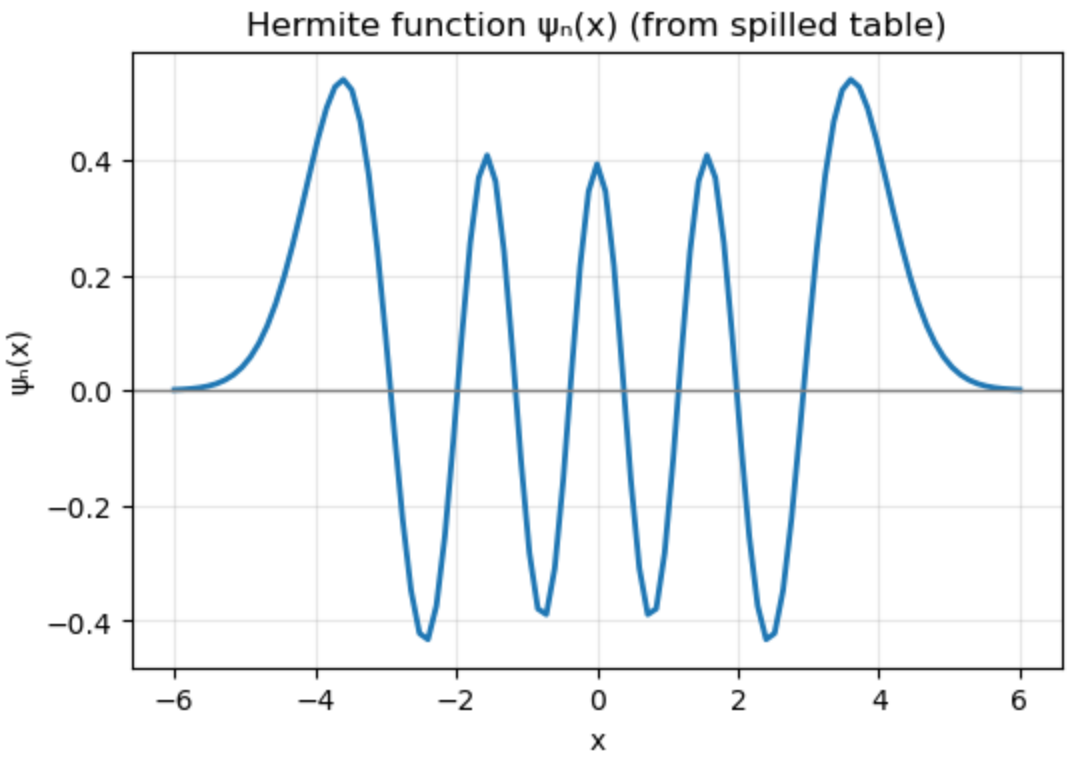

Figure 9 n = 8 Wave Function

roots(H_9)

-3.190993202
-2.266580585
-1.468553289
-0.723551019
-2.62003E-17
0.723551019
1.468553289
2.266580585
3.190993202

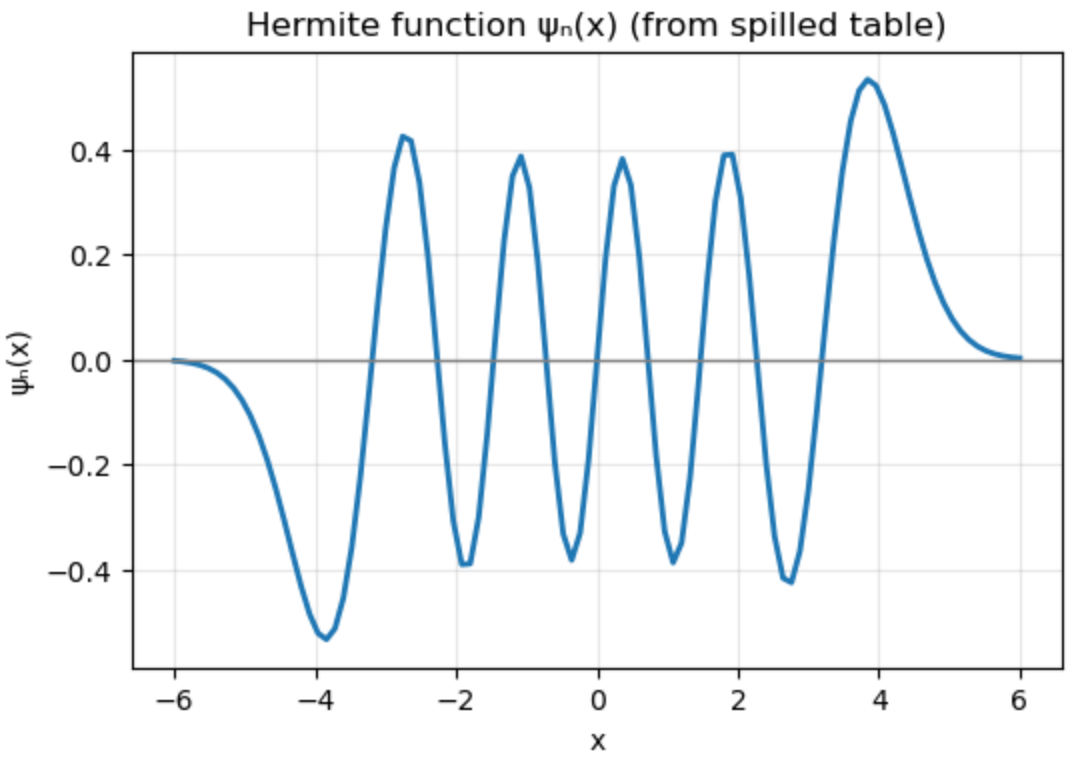

Figure 10 n = 9 Wave Function

roots(H_10)

-3.436159119
-2.532731674
-1.756683649
-1.03661083
-0.342901327
0.342901327
1.03661083
1.756683649
2.532731674
3.436159119

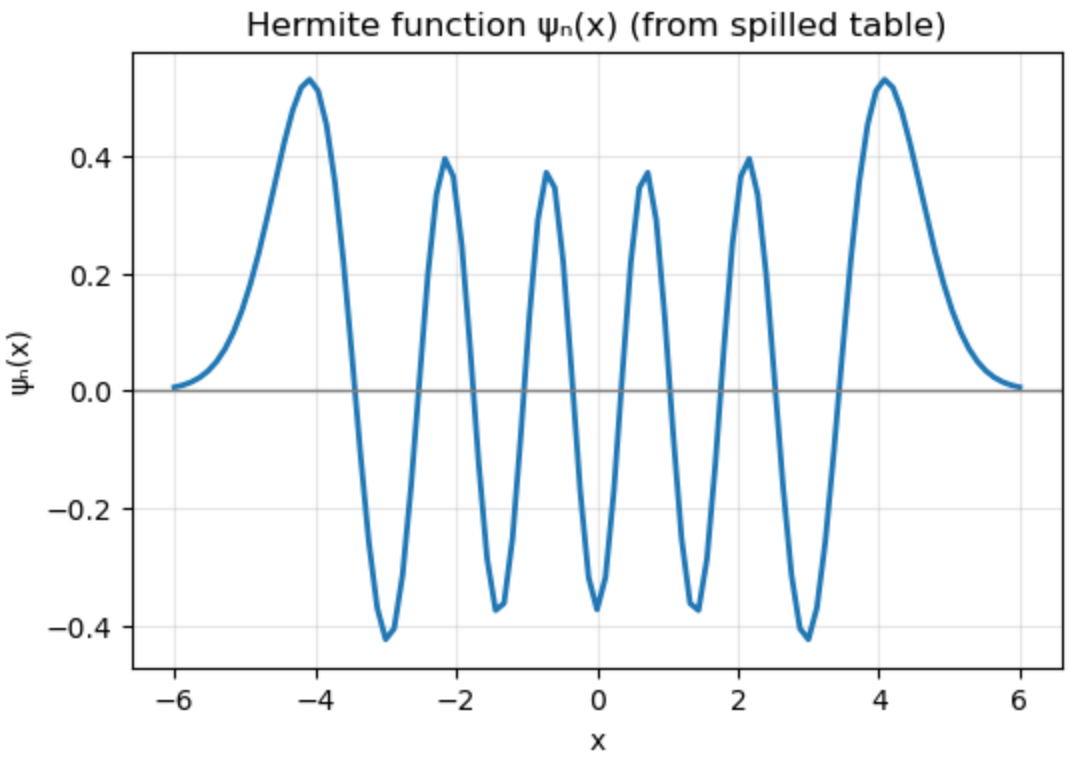

Figure 11 n = 10 Wave Function

Appendix B: Python Source Code

This source code was essentially written by Microsoft's Copilot which is a very knowledgeable artificial intelligence tool.

In cell A1:

```
import numpy as np
import math
from numpy.polynomial import Hermite

# --- User input: degree in cell D1 ---
n_raw = xl("D1")          # Excel -> Python (may come in as a
small table-like object)
n = int(np.array(n_raw).ravel()[0])  # robustly extract the
scalar and convert to int

# --- Polynomial and sampling grid ---
p = Hermite.basis(n)                  # physicists' Hermite
basis polynomial of degree n
[5](https://numpy.org/doc/stable/reference/generated/numpy.polyn
omial.hermite.Hermite.basis.html)[1](https://docs.scipy.org/doc/
/numpy-1.9.3/reference/routines.polynomials.hermite.html)
xs = np.linspace(-6.0, 6.0, 101)

# --- Normalized Hermite function psi_n(x) ---
# normalization: 1/sqrt(2^n * n! * sqrt(pi))
[3](https://people.math.sc.edu/Burkardt/m_src/hermite_polynomial
/hermite_polynomial.html)
norm = 1.0 / math.sqrt((2.0**n) * math.factorial(n) *
math.sqrt(math.pi))
ys = norm * np.exp(-0.5 * xs**2) * p(xs)

# --- Roots (of H_n), padded to match length of xs for Excel
spill ---
roots = p.roots()
roots_col = np.empty(len(xs), dtype=object)
roots_col[:] = ""
roots_col[:len(roots)] = roots

#out = [[f"x (n={n})", "ψ_n#(x)", "roots(H_n)"], ...]
out = [[f"x (n={n})", "ψ_n(x)", "roots(H_n)"]] +
np.column_stack((xs, ys, roots_col)).tolist()
out
```

Place desired n (0 to 10), quantum number, in cell D1.
In cell E1:

```
import numpy as np
import matplotlib.pyplot as plt

n_raw = xl("D1")
n = int(np.array(n_raw).ravel()[0])

tbl = xl("A1:C203", headers=True)

x = np.array(tbl.iloc[:, 0])   # first column, regardless of its
name
y = np.array(tbl.iloc[:, 1])   # second column

fig, ax = plt.subplots(figsize=(6, 4))
ax.plot(x, y, lw=2)
ax.axhline(0, color="gray", lw=1)
ax.set_title("Hermite function ψₙ(x) (from spilled table)")
ax.set_xlabel("x")
ax.set_ylabel("ψₙ(x)")
ax.grid(True, alpha=0.3)

fig
```