

# Numerical Approaches for Quantum Mechanics

Raghunathan Ramakrishnan  
[ramakrishnan@tifrh.res.in](mailto:ramakrishnan@tifrh.res.in)

Tata Institute of Fundamental Research  
TIFR Center for Interdisciplinary Sciences  
Hyderabad, INDIA

30 December 2021



TIFR Centre for  
Interdisciplinary  
Studies

The presentation and the codes are available for download at <https://github.com/raghurama123/NumQM>

# Purpose of this presentation

- ❖ To give a flavour for numerical modelling of quantum mechanical problems.
- ❖ To demonstrate how a bit of programming can clarify important concepts.
- ❖ To highlight the connections between concepts learned in quantum mechanics and the mathematics encountered in numerical computation.
- ❖ To provide sample programs that you can download, modify and enhance your understanding.

# How to work the notebooks interactively

github.com/raghurama123/NumQM\_Basic

Animations Add files via upload 14 hours ago

img Add files via upload 4 days ago

notebooks Add files via upload 14 hours ago

pdf Add files via upload 14 hours ago

LICENSE Initial commit 4 days ago

README.md Update README.md 14 hours ago

**numerical-methods**

Readme MIT License 12 stars 2 watching 3 forks

**Releases**

No releases published Create a new release

**Packages**

No packages published Publish your first package

**Languages**

Jupyter Notebook 100.0

**README.md**

The presentation and the notebooks are prepared for the lecture Numerical Approaches for Quantum Mechanics on 30 December 2021 as a part of the program [National Initiative on Undergraduate Science \(NIUS\)](#). NIUS is an initiative of the [Homi Bhabha Centre for Science Education, TIFR](#).

Click the slides below for the presentation

## Numerical Approaches for Quantum Mechanics

Raghunathan Ramakrishnan  
[ramakrishnan@tifrh.res.in](mailto:ramakrishnan@tifrh.res.in)

Tata Institute of Fundamental Research  
TIFR Center for Interdisciplinary Sciences  
Hyderabad, INDIA

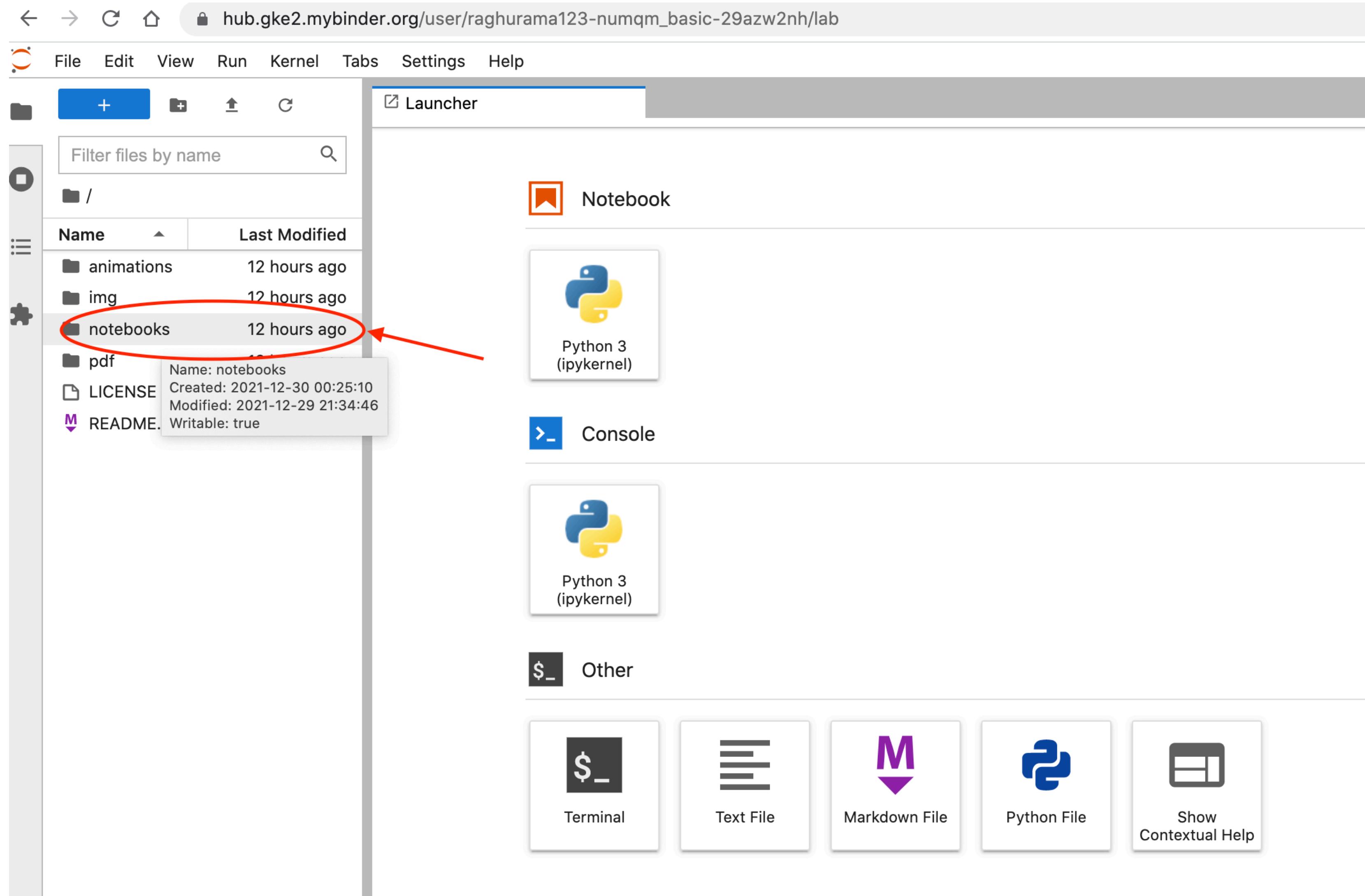
30 December 2021

 TIFR Centre for  
Interdisciplinary  
Studies

The presentation and the codes are available for download at <https://github.com/raghurama123/NumQM>

You can access the Ipython notebooks interactively at  
[https://mybinder.org/v2/gh/raghurama123/NumQM\\_Basic/HEAD](https://mybinder.org/v2/gh/raghurama123/NumQM_Basic/HEAD)

# How to work the notebooks interactively



# How to work the notebooks interactively

hub.gke2.mybinder.org/user/raghurama123-numqm\_basic-29azw2nh/lab/tree/notebooks/NumQM\_Basic\_01.ipynb

File Edit View Run Kernel Tabs Settings Help

+ Filter files by name / notebooks /

Name	Last Modified
NumQM_Basic_01.ipynb	12 hours ago
NumQM_Basic_02.ipynb	12 hours ago
NumQM_Basic_03.ipynb	12 hours ago
NumQM_Basic_04.ipynb	12 hours ago
NumQM_Basic_05.ipynb	12 hours ago
NumQM_Basic_06.ipynb	12 hours ago

Launcher NumQM\_Basic\_01.ipynb

[1]:

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

def psi(n,L,x):
    psi=np.sqrt(2.0/L)*np.sin(n*np.pi*x/L)
    return psi

L=1.0

x=np.linspace(0.0, L, 51)

f1=np.zeros(51)
f2=np.zeros(51)

for i in range(51):
    f1[i]=psi(1,L,x[i])
    f2[i]=psi(2,L,x[i])

plt.plot(x,f1)
plt.plot(x,f2)
plt.legend(['n = 1','n = 2'])

plt.xlabel("$x$", fontsize=14)
plt.ylabel("$\psi_n(x)$", fontsize=14)
plt.savefig('PIB.png')
plt.show()
```

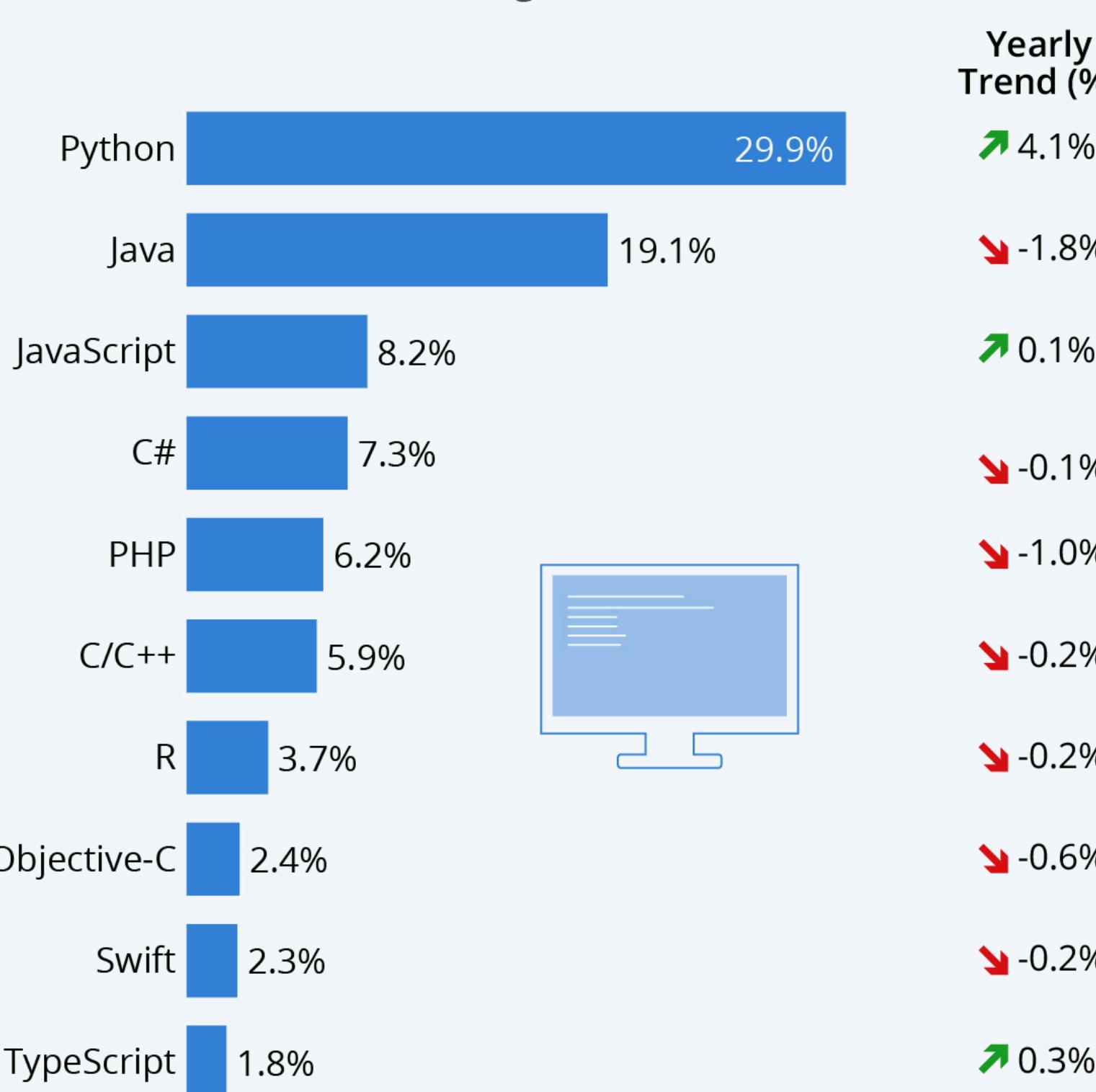
# Why Numerical Approaches?

- ❖ Except for simple cases, quantum mechanical problems are mathematically intensive.
- ❖ Numerical modelling provides quick solutions to analytically complicated problems.
- ❖ When combined with visualisation of results, computer modelling provides deep insights.
- ❖ One can perform virtual experiments that may not even be feasible to perform in a laboratory.
- ❖ Free software can help fast-track coding efforts.
- ❖ Possible to combine to symbolic computation with numerical computation.

# Choosing a programming language

## Python Remains Most Popular Programming Language

Popularity of each programming language based on share of tutorial searches in Google



Yearly trend compares percent change from Feb 2019 to Feb 2020

Sources: GitHub, Google Trends



statista

Q: What's the best programming language to learn for science student with no previous programming experience

A: Python

Python is

- ❖ free
- ❖ easy to reference in the internet
- ❖ has a lot of libraries for visualisation, numerical methods, and data-analysis
- ❖ more libraries means less coding effort so that one can focus on the research problem at hand



I want to have Python in my computer but I  
don't know how to install it. What to do?



*Don't be shy to ask around.*

Take help from friends, teachers, or research scholars in your institute.

# Mathematics and Numerical Methods

Equations	Root-finding (Newton-Raphson, simplex, etc.)
Functions	Optimization: minimization (Newton-Raphson, simplex, etc.)
Function approximation	Least-squares regression (Gaussian elimination, LU decomposition, etc.), interpolation
Eigenvalue problems	Similarity transformation (Jacobi method), Iterative method (power method, Lanczos, etc.)
Differentiation/Integration	Finite-differences, trapezoidal method, quadratures, etc.
Differential equations	Euler method, Runge-Kutta, etc.

# Let's begin with a plot

```
In [1]: import numpy as np
import matplotlib.pyplot as plt

def psi(n,L,x):
    psi=np.sqrt(2.0/L)*np.sin(n*np.pi*x/L)
    return psi

L=1.0

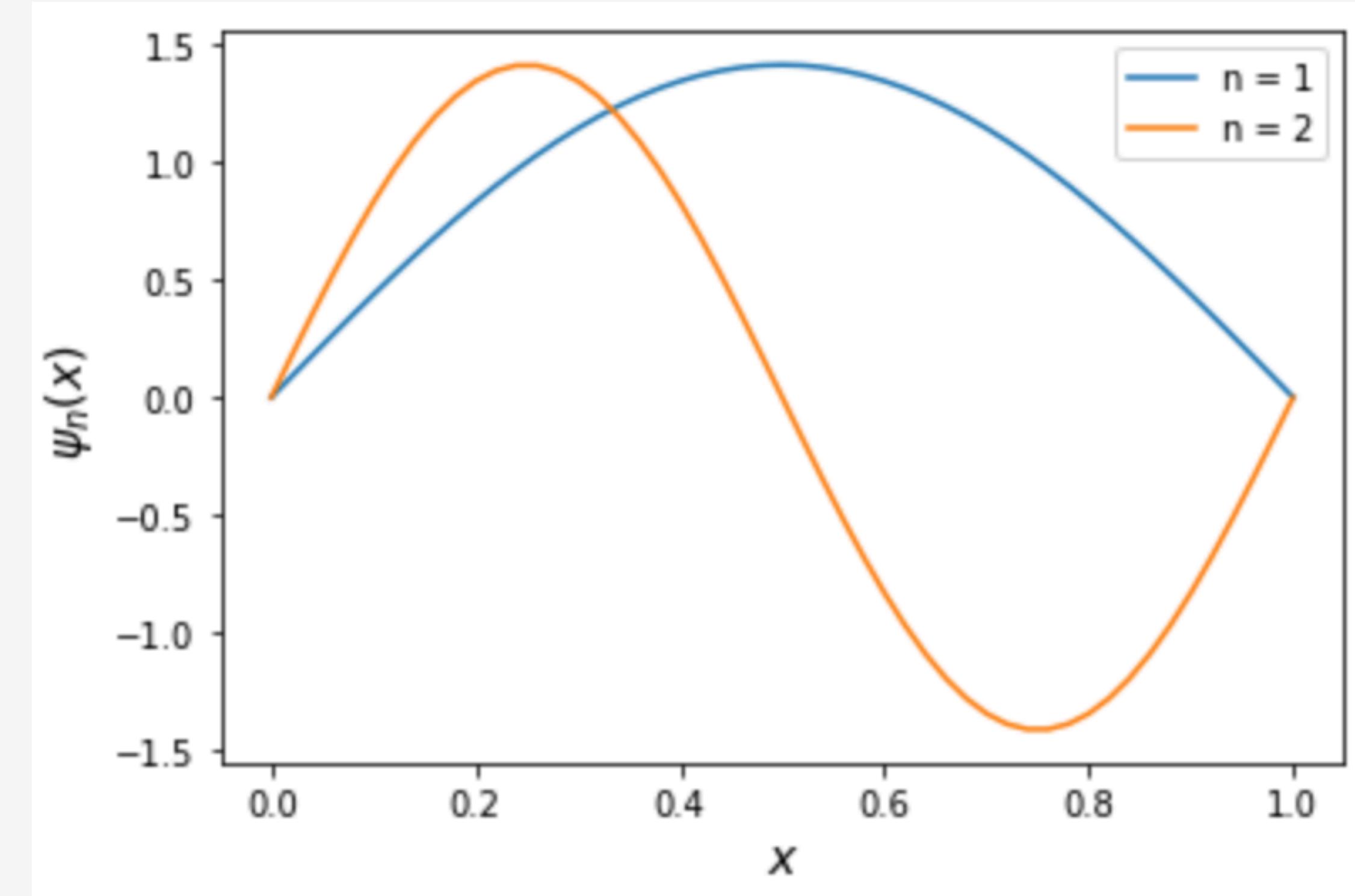
x=np.linspace(0.0, L, 51)

f1=np.zeros(51)
f2=np.zeros(51)

for i in range(51):
    f1[i]=psi(1,L,x[i])
    f2[i]=psi(2,L,x[i])

plt.plot(x,f1)
plt.plot(x,f2)
plt.legend(['n = 1', 'n = 2'])

plt.xlabel("$x$", fontsize=14)
plt.ylabel("$\psi_n(x)$", fontsize=14)
plt.savefig('PIB.png')
plt.show()
```



# Non-stationary states and their time-evolution

Recall:

- ❖ Eigenstates of a system are stationary. When a system is in one of the eigenstates, all quantum mechanical observables (expectation values,  $\langle n | \hat{O} | n \rangle$ ) are conserved in time.
- ❖ Linear combinations of eigenfunctions are not stationary. Consider  $\phi(x) = [\psi_m(x) + \psi_n(x)]/\sqrt{2}$

$$\phi(x, t) = \exp\left(-it\hat{H}/\hbar\right) [\psi_m(x) + \psi_n(x)]/\sqrt{2} = \left[ \exp\left(-itE_m/\hbar\right) \psi_m(x) + \exp\left(-itE_n/\hbar\right) \psi_n(x) \right]/\sqrt{2}$$

$$\begin{aligned} |\phi(x, t)|^2 &= \phi^*(x, t)\phi(x, t) \\ &= \left( |\psi_m(x)|^2 + |\psi_n(x)|^2 \right)/2 + \text{Real} \left[ \exp(-it\omega_{mn}) \psi_n^*(x) \psi_m(x) \right]; \quad \omega_{mn} = (E_m - E_n)/\hbar \\ &= \left( |\psi_m(x)|^2 + |\psi_n(x)|^2 \right)/2 + \cos(\omega_{mn}t) \psi_n^*(x) \psi_m(x) \end{aligned}$$

# Let's watch the wave function evolve in time

```
In [2]: t=np.linspace(0.0, 1.0, 51)

hbar=1                      # in atomic units
mass_e=1                      # in atomic units

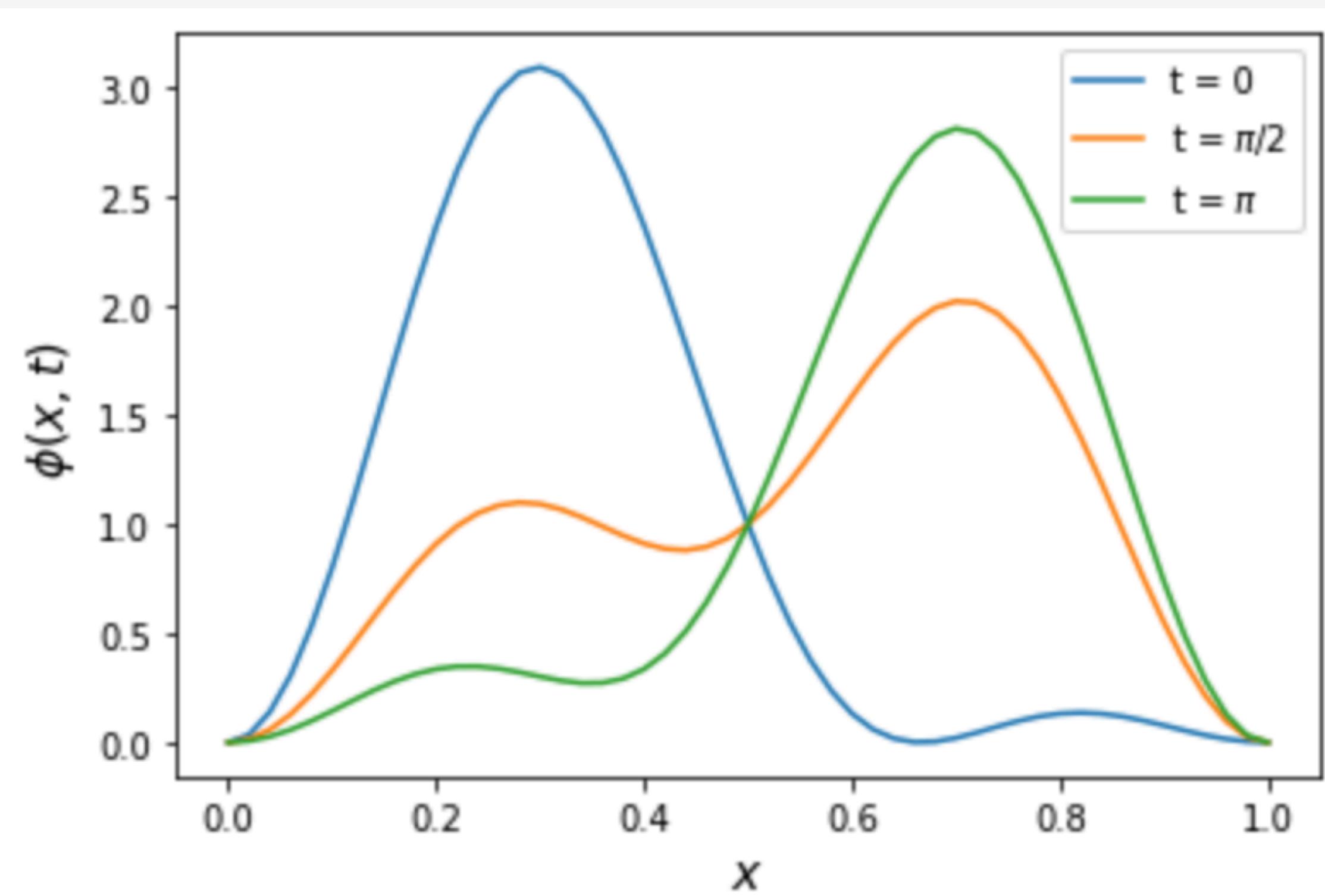
h=2*np.pi * hbar

E1=1**2 * h**2/(8*mass_e*L**2)
E2=2**2 * h**2/(8*mass_e*L**2)
w21 = (E2-E1)/hbar

def phi(t):
    phi=np.zeros(51)
    phi=(f1**2 + f2**2)/2+np.cos(w21*t)*f1*f2
    return phi

plt.plot(x,phi(0))
plt.plot(x,phi(np.pi/2))
plt.plot(x,phi(np.pi))
plt.legend(['t = 0','t = $\pi$/2','t = $\pi$'])

plt.xlabel("$x$", fontsize=14)
plt.ylabel("$\phi(x,t)$", fontsize=14)
plt.savefig('PIB_2.png')
plt.show()
```



# Movie time!

```
In [3]: import os
import imageio
filenames = []
for it in range(100):
    t=it*0.01

    plt.plot(x,phi(t))

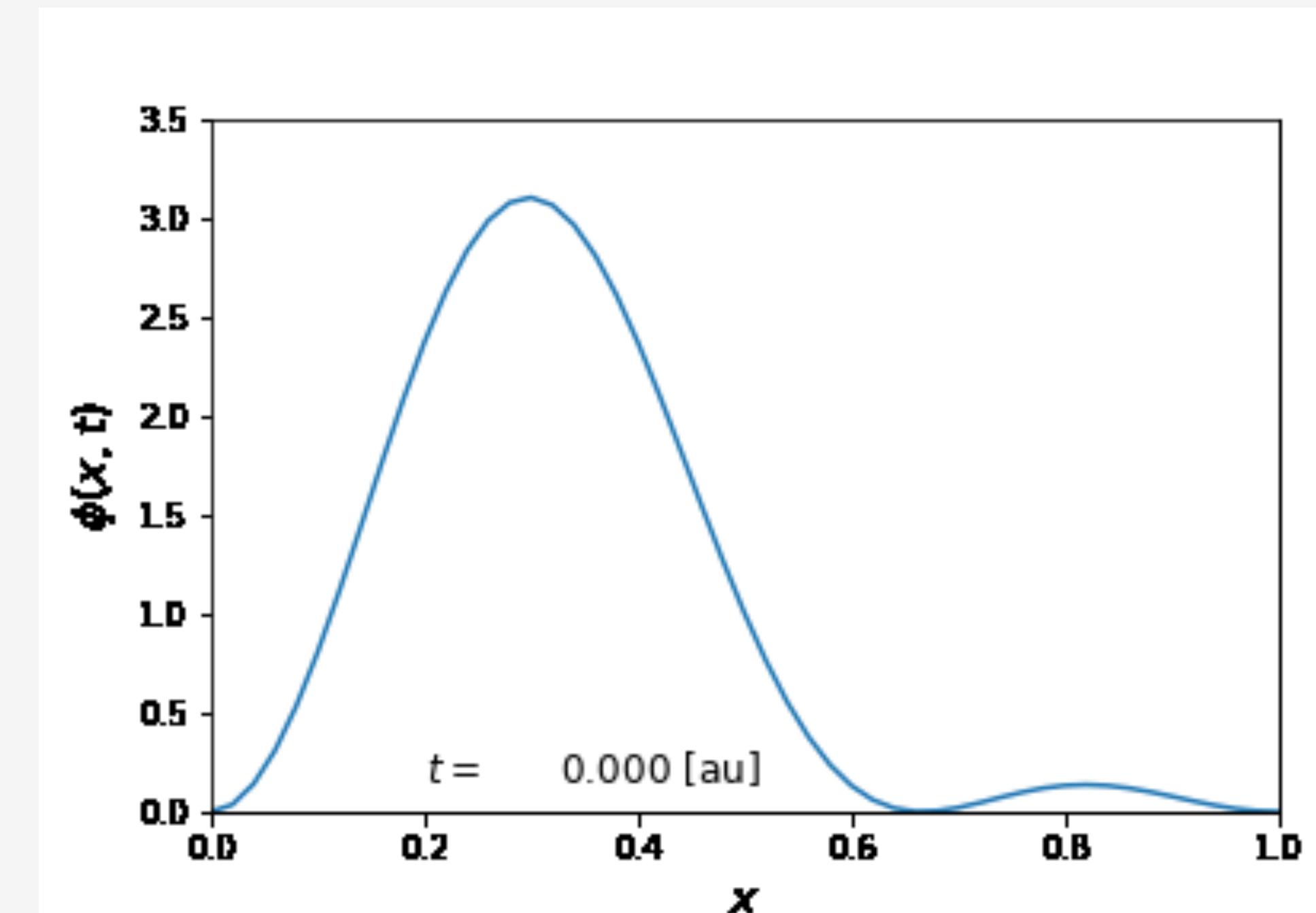
    plt.xlim(0, L)
    plt.ylim(0, 3.5)
    plt.xlabel("$x$", fontsize=14)
    plt.ylabel("$\phi(x,t)$", fontsize=14)

    filename='_tmp_'+str(it).zfill(5)+'.png'
    filenames.append(filename)
    plt.savefig(filename)

    plt.close()

# build animated gif
with imageio.get_writer('PIB_t.gif', mode='I') as writer:
    for filename in filenames:
        image = imageio.imread(filename)
        writer.append_data(image)

# remove temporary files
for filename in set(filenames):
    os.remove(filename)
```



# Homework

Exercise 1: It appears from the animation shown in the previous slide that the initial wave function seems to get back its shape after evolving for some time. This time period is called as the wave packet revival time. Can you derive an analytic expression for the revival time and compare it with the simulation?

Hint: You can generate the animation yourself using the interactive notebooks at [https://mybinder.org/v2/gh/raghurama123/NumQM\\_Basic/HEAD](https://mybinder.org/v2/gh/raghurama123/NumQM_Basic/HEAD)

Does wave packet revival remind you of any classical phenomenon, where the system comes back to its initial configuration after a bit of time-evolution?



# Wavefunctions in $x$ and $p$ representations

Suppose a system is in a state denoted by the ket  $|n\rangle$ , then the corresponding wave function of the system is obtained by representing the ket in the position or momentum eigen kets.

$$\psi_n(x) = \langle x | n \rangle; \quad \phi_n(p) = \langle p | n \rangle$$

The definition of the basis kets  $|x\rangle$  and  $|p\rangle$  has to be taken as one of the postulates of quantum mechanics. They are related to one another by Fourier transformation.

	$x$ -representation	$p$ -representation
$ x_0\rangle$	$\langle x   x_0 \rangle = \delta(x - x_0)$	$\langle p   x_0 \rangle = \frac{1}{\sqrt{2\pi\hbar}} \exp(-ipx_0/\hbar)$
$ p_0\rangle$	$\langle x   p_0 \rangle = \frac{1}{\sqrt{2\pi\hbar}} \exp(ip_0x/\hbar)$	$\langle p   p_0 \rangle = \delta(p - p_0)$

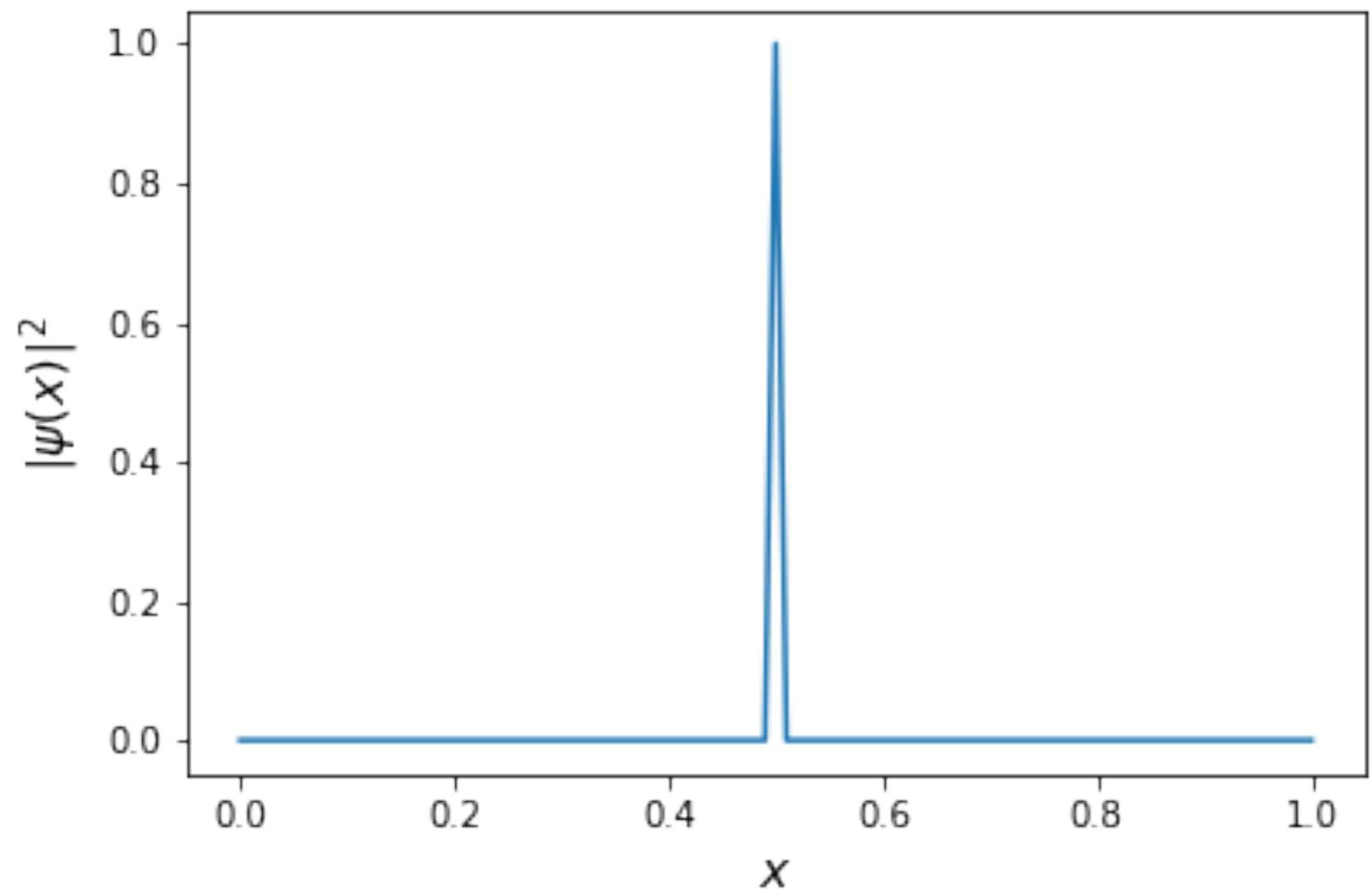
# Changing representations

$$\begin{aligned}\phi_n(p) &= \langle p | n \rangle \\&= \langle p | \hat{I} | n \rangle \\&= \langle p | \int dx | x \rangle \langle x | n \rangle \\&= \int dx \langle p | x \rangle \langle x | n \rangle \\&= \int dx \langle p | x \rangle \psi_n(x) \\&= \frac{1}{\sqrt{2\pi\hbar}} \int dx \exp(-ipx/\hbar) \psi_n(x)\end{aligned}$$

$$\begin{aligned}\psi_n(x) &= \langle x | n \rangle \\&= \langle x | \hat{I} | n \rangle \\&= \langle x | \int dp | p \rangle \langle p | n \rangle \\&= \int dp \langle x | p \rangle \langle p | n \rangle \\&= \int dp \langle x | p \rangle \phi_n(p) \\&= \frac{1}{\sqrt{2\pi\hbar}} \int dp \exp(ipx/\hbar) \phi_n(p)\end{aligned}$$

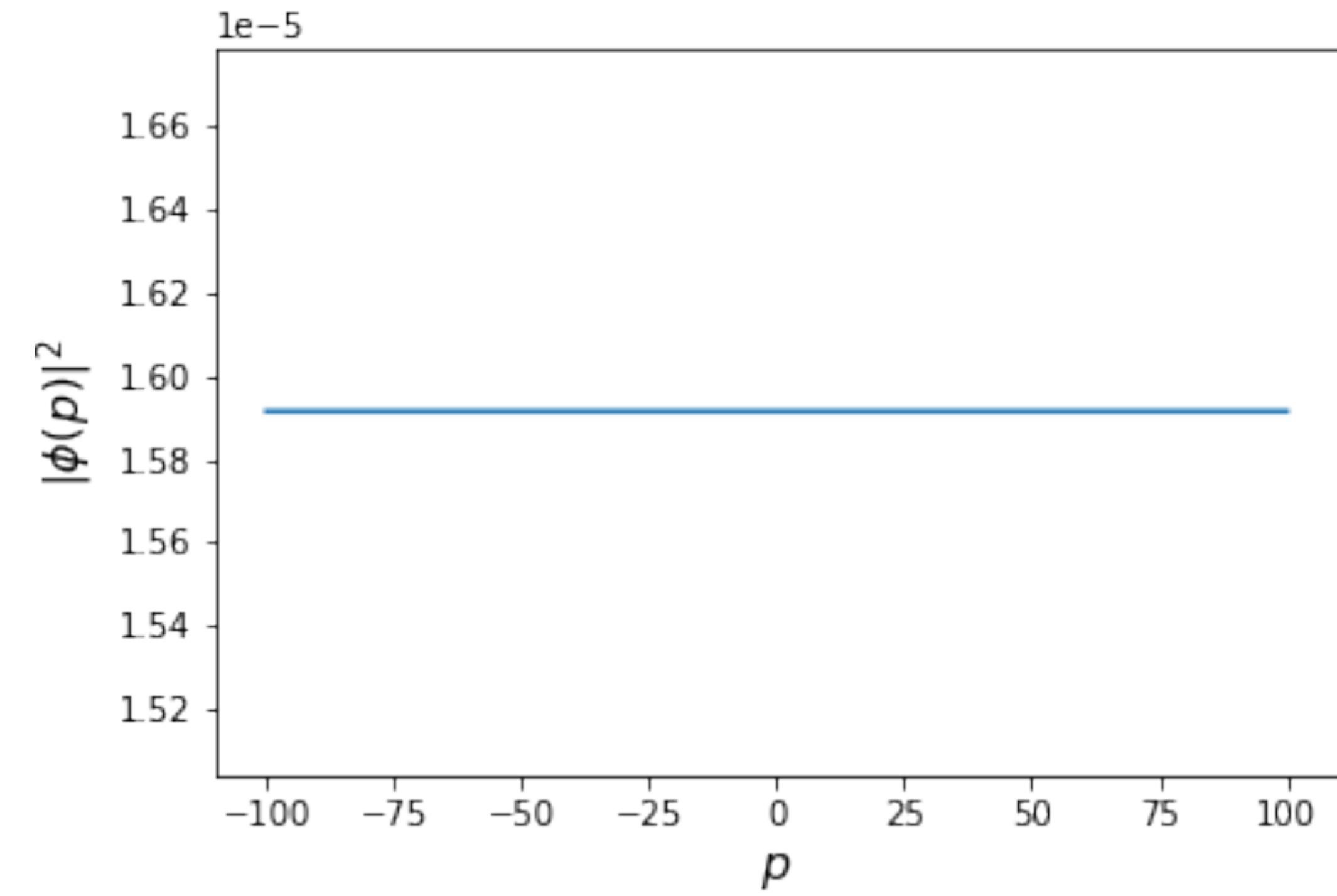
$\hat{I} = \int dx | x \rangle \langle x | = \int dp | p \rangle \langle p |$  is called as the completeness relation

# Fourier transform of one position eigenstate



full-localisation at one point in the  $x$ -representation

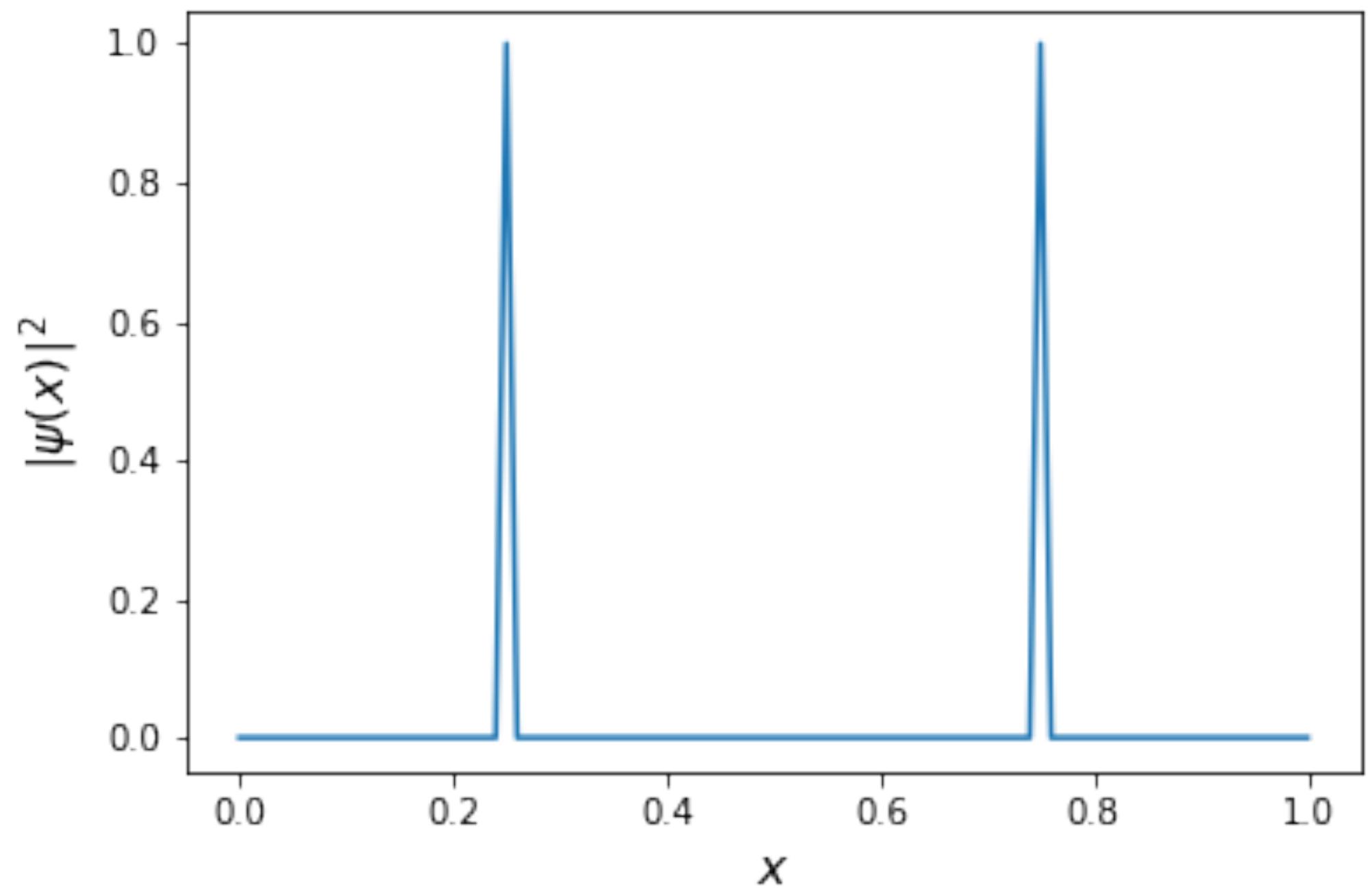
$$\psi(x) = \langle x | x = 0.5 \rangle = \delta(x - 0.5)$$



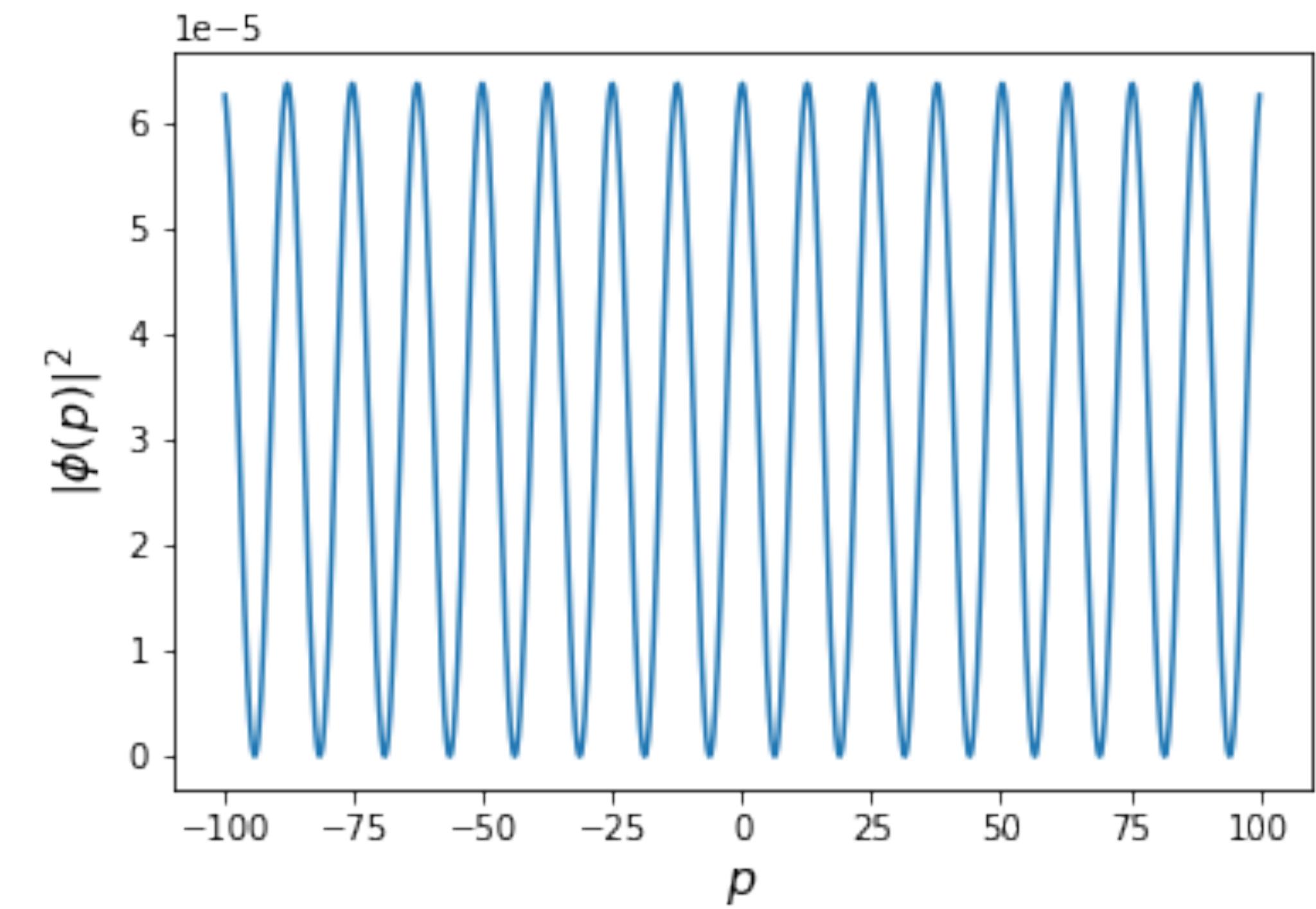
completely delocalised in the  $p$ -representation

$$\phi(p) = \langle p | x = 0.5 \rangle = \frac{1}{\sqrt{2\pi\hbar}} \exp(-ipx/\hbar)$$

# Adding one more position eigenstate

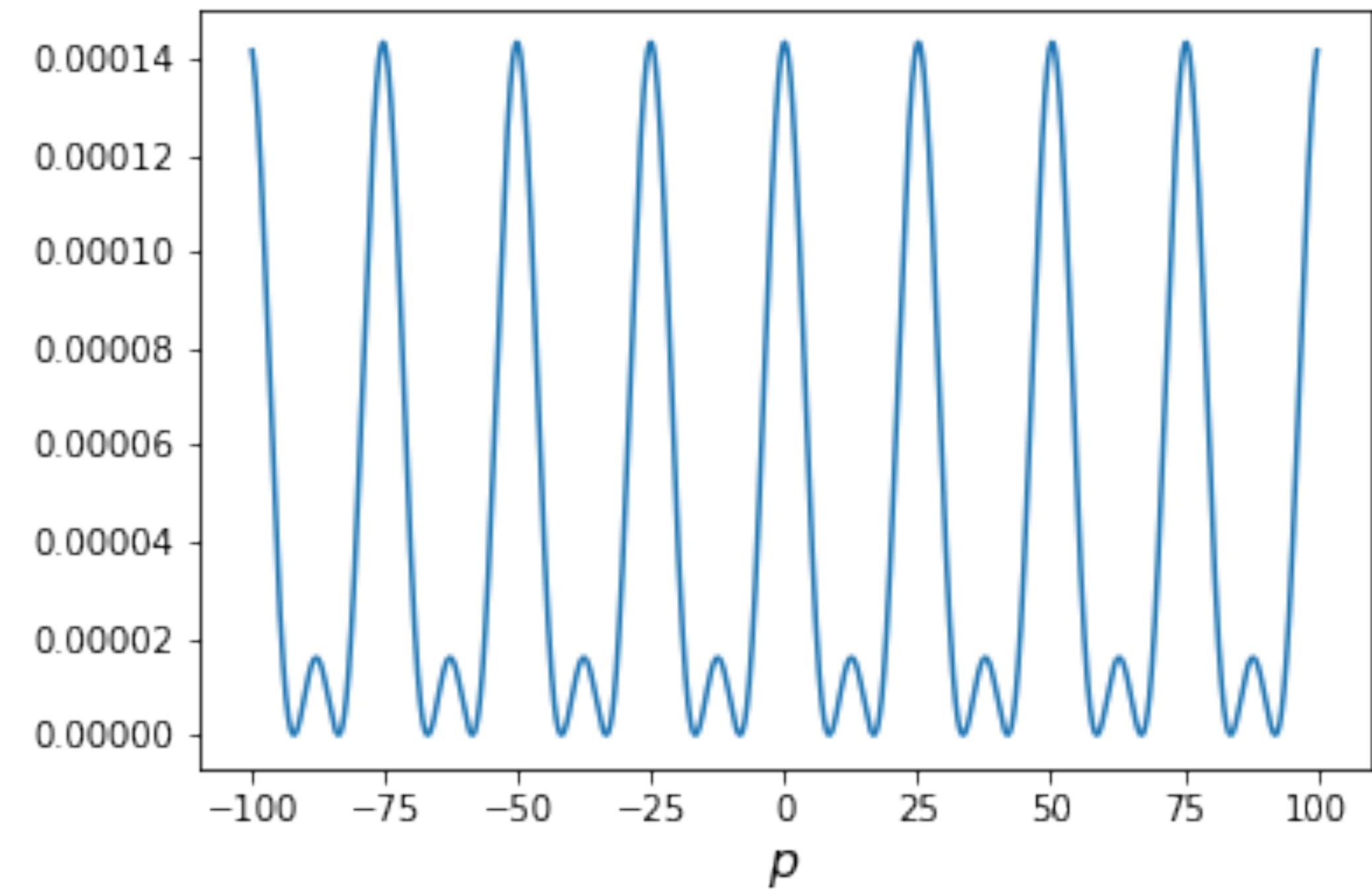
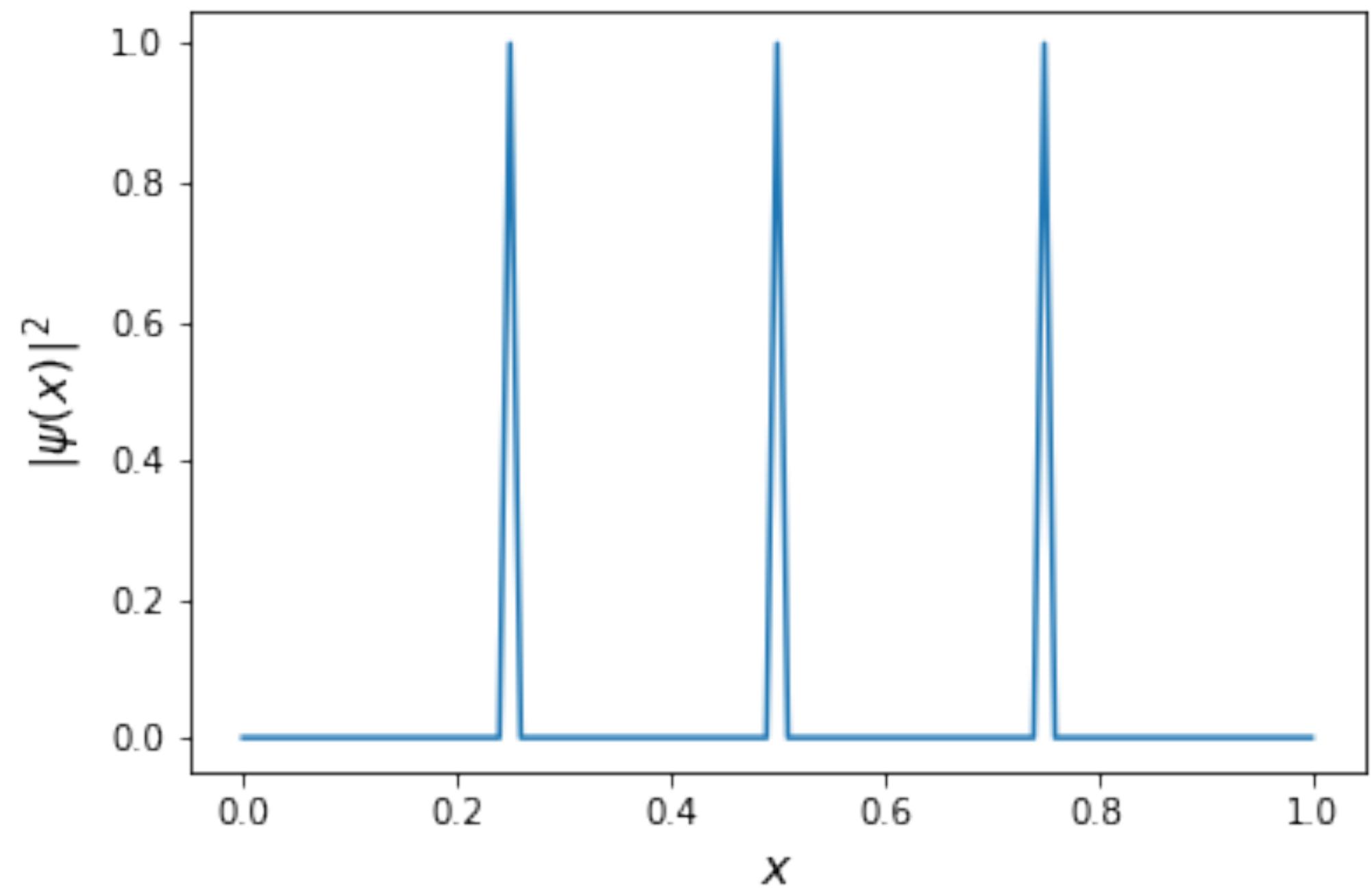


$$\psi(x) = \langle x | 0.25 \rangle + \langle x | 0.75 \rangle$$

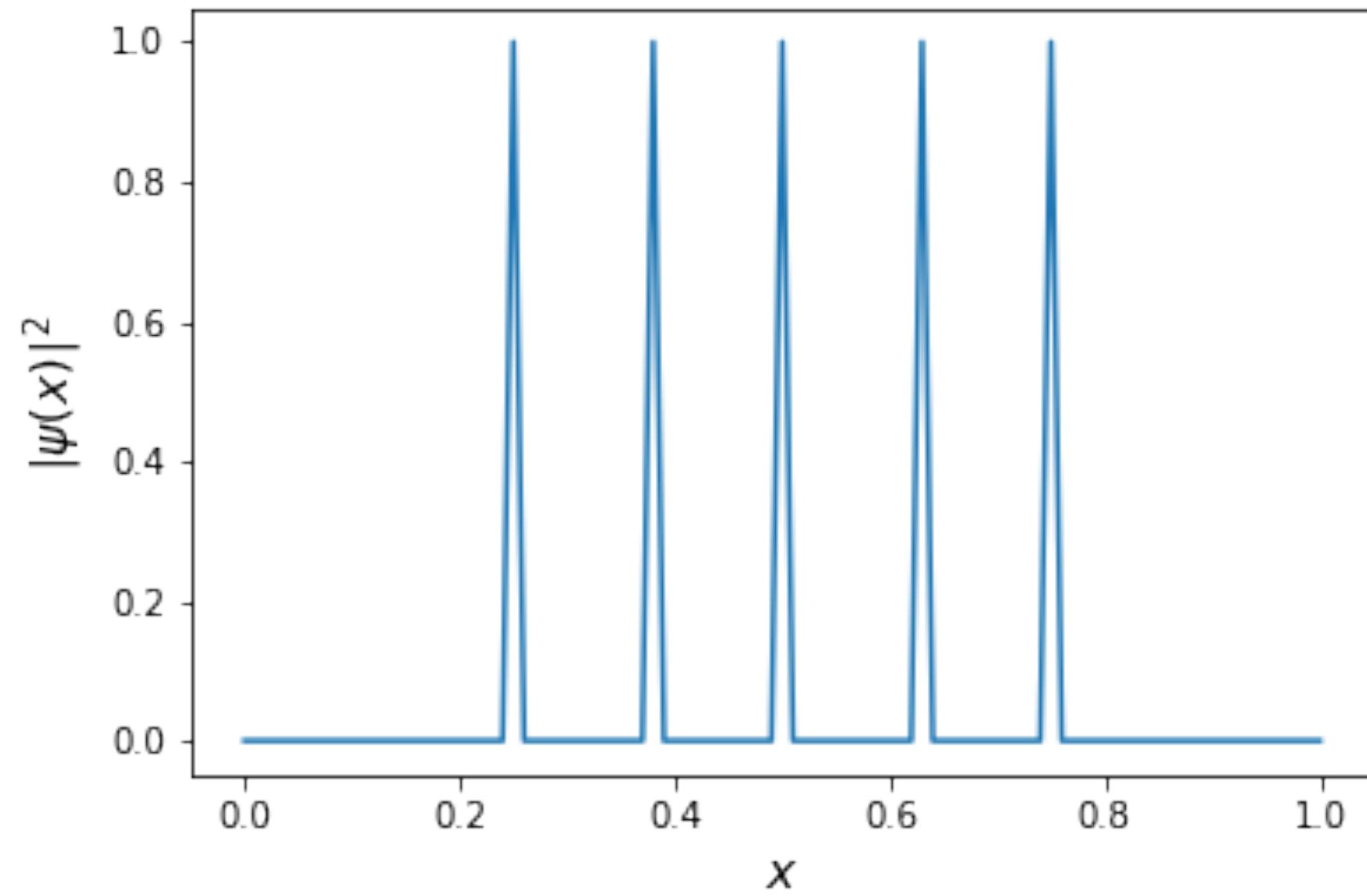


$$\phi(p) = \frac{1}{\sqrt{2\pi\hbar}} \int dx \exp(-ipx/\hbar) \psi_n(x)$$

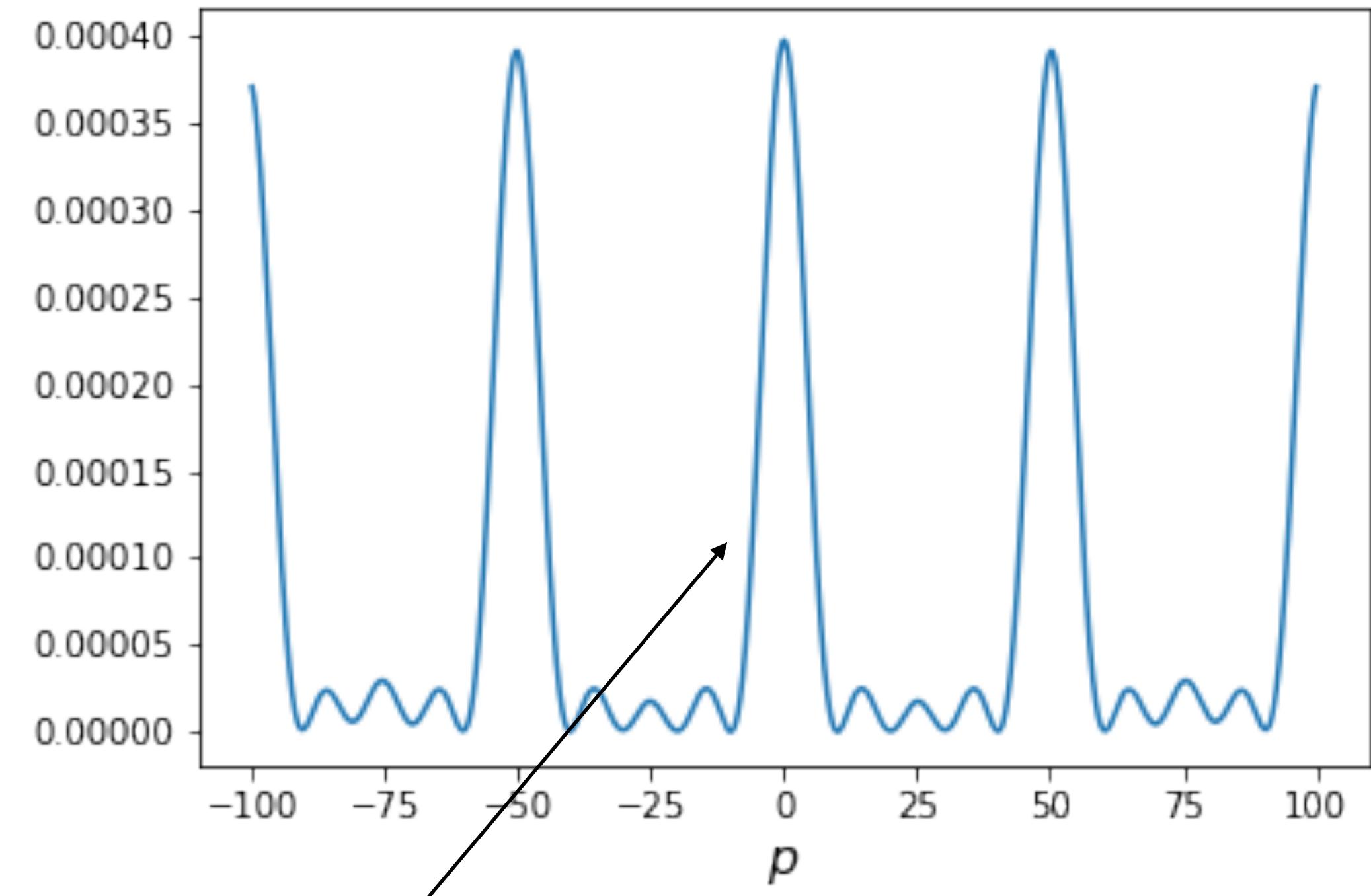
# Sum of three position eigenstates



# Sum of five position eigenstates

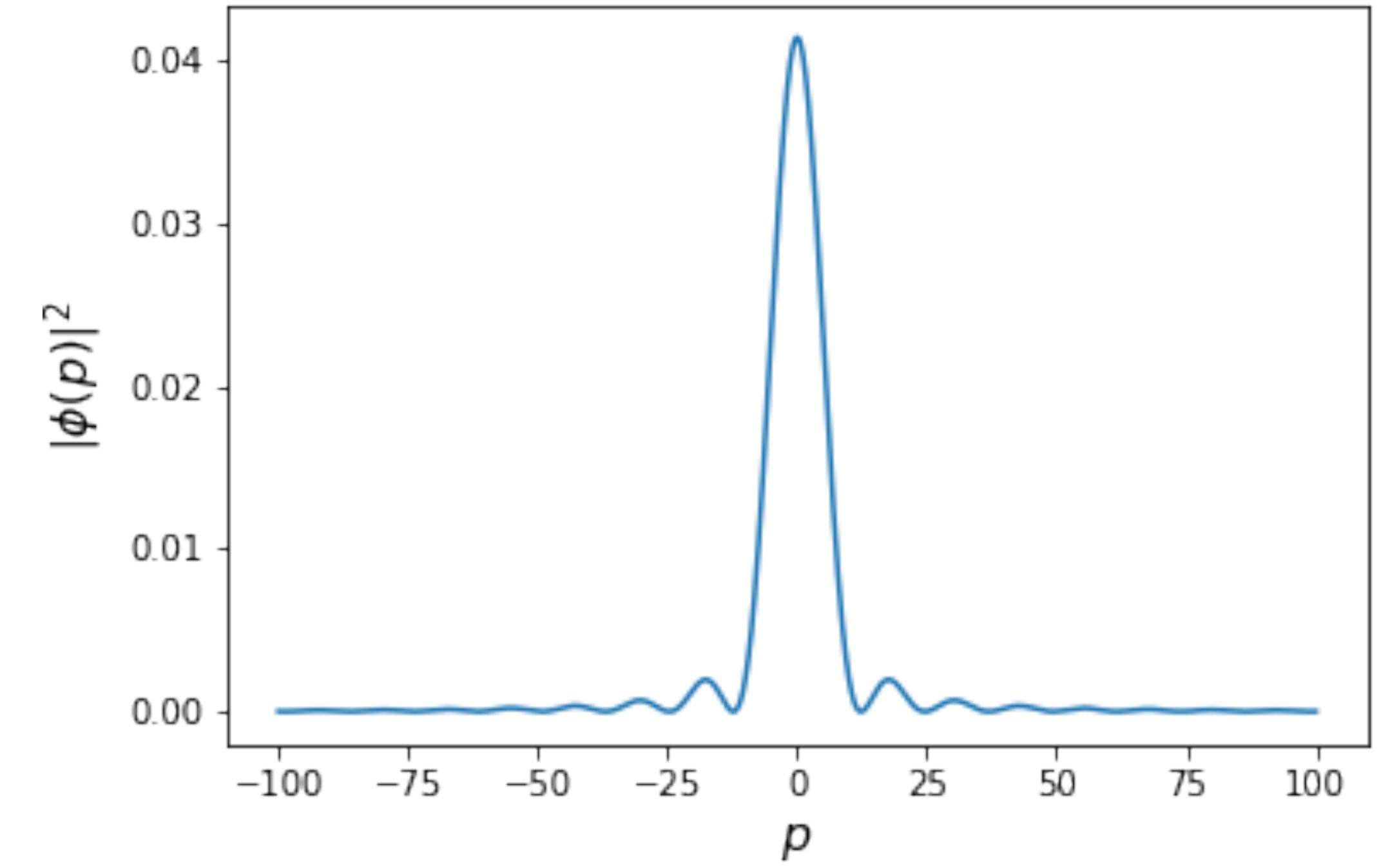
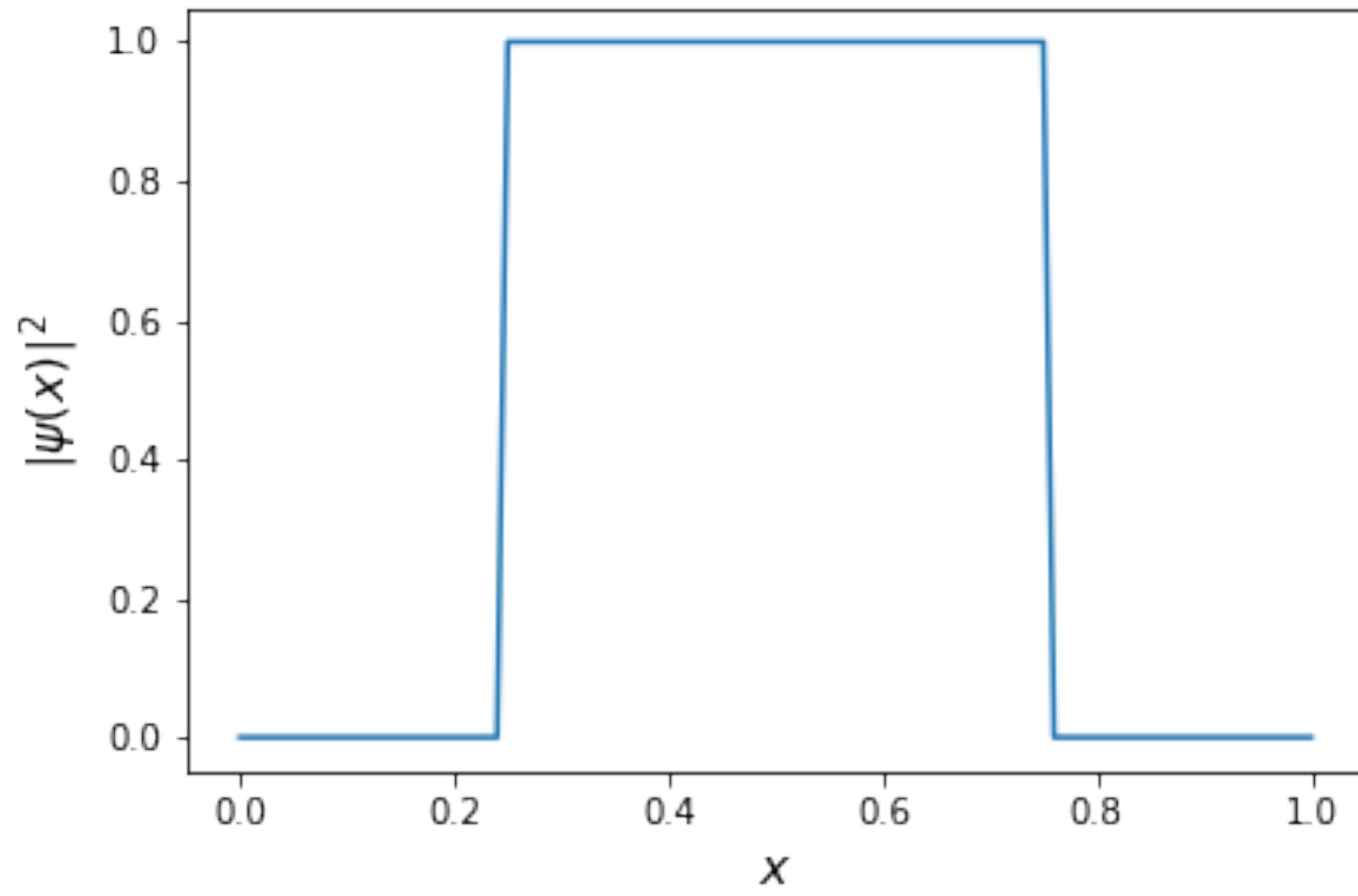


beginning to spread in the  $x$ -representation



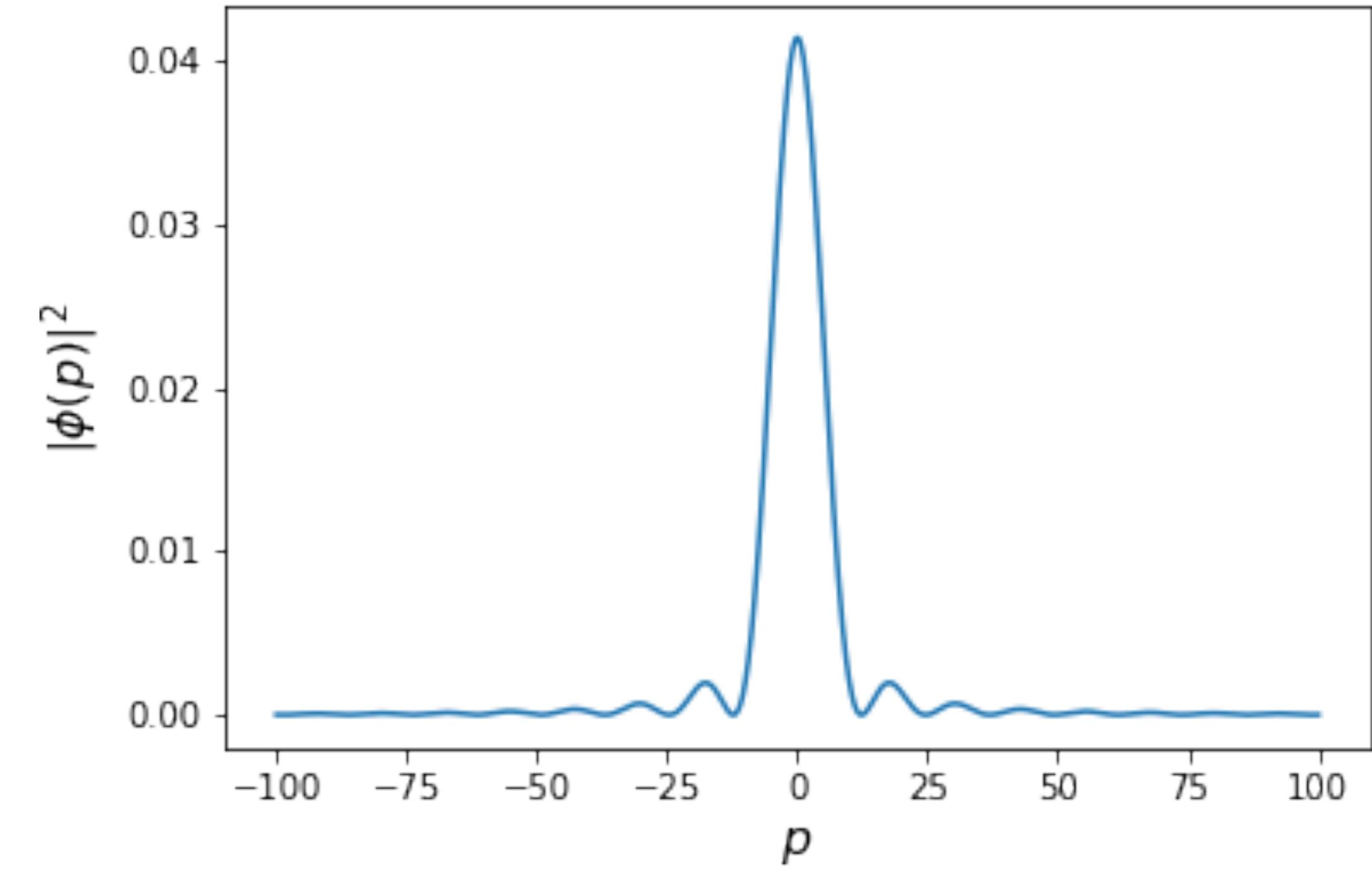
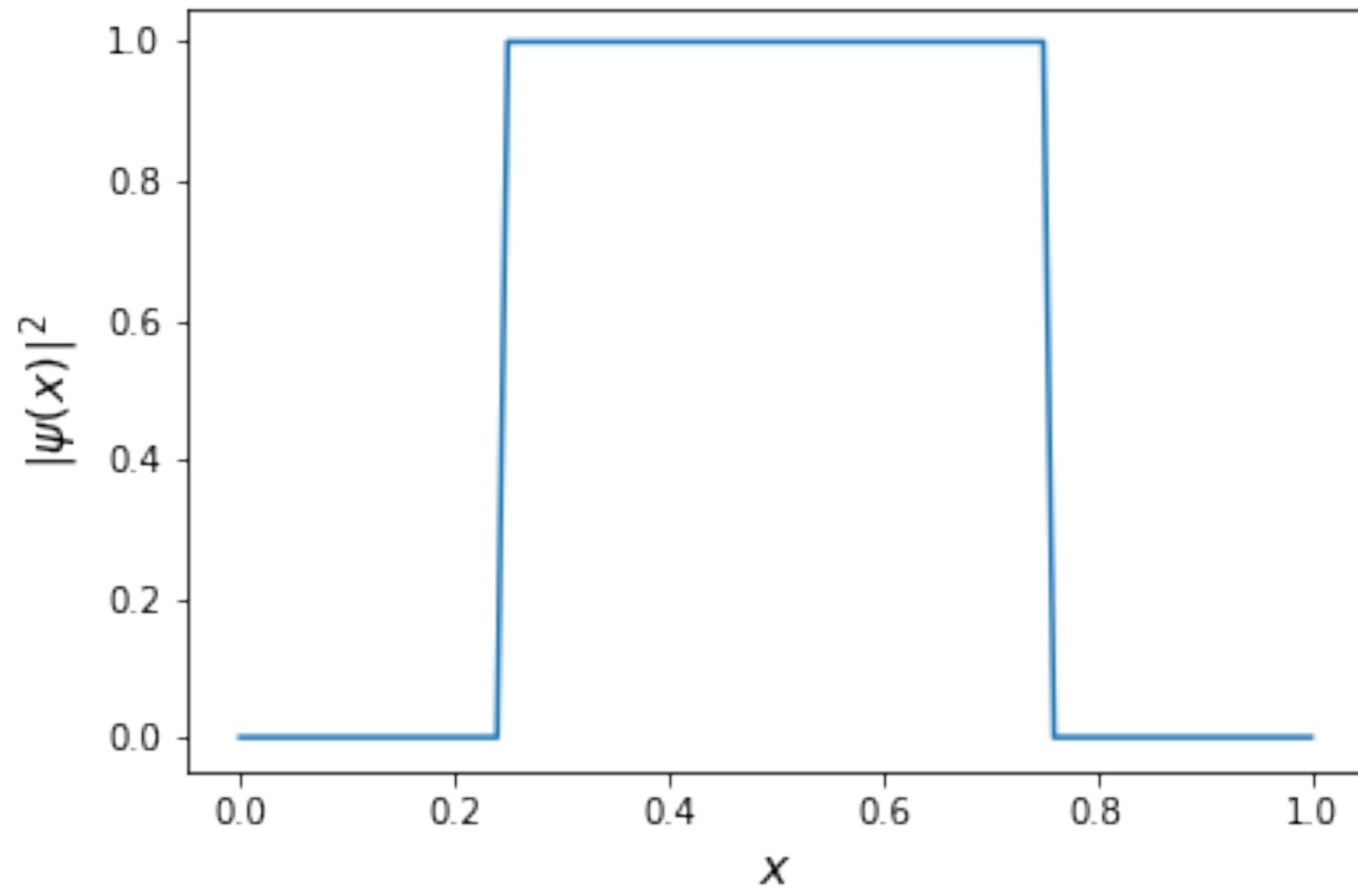
quasi-localisation in the  $p$ -representation emerges

# Integration over position eigenstates



full-localisation in the  $p$ -representation

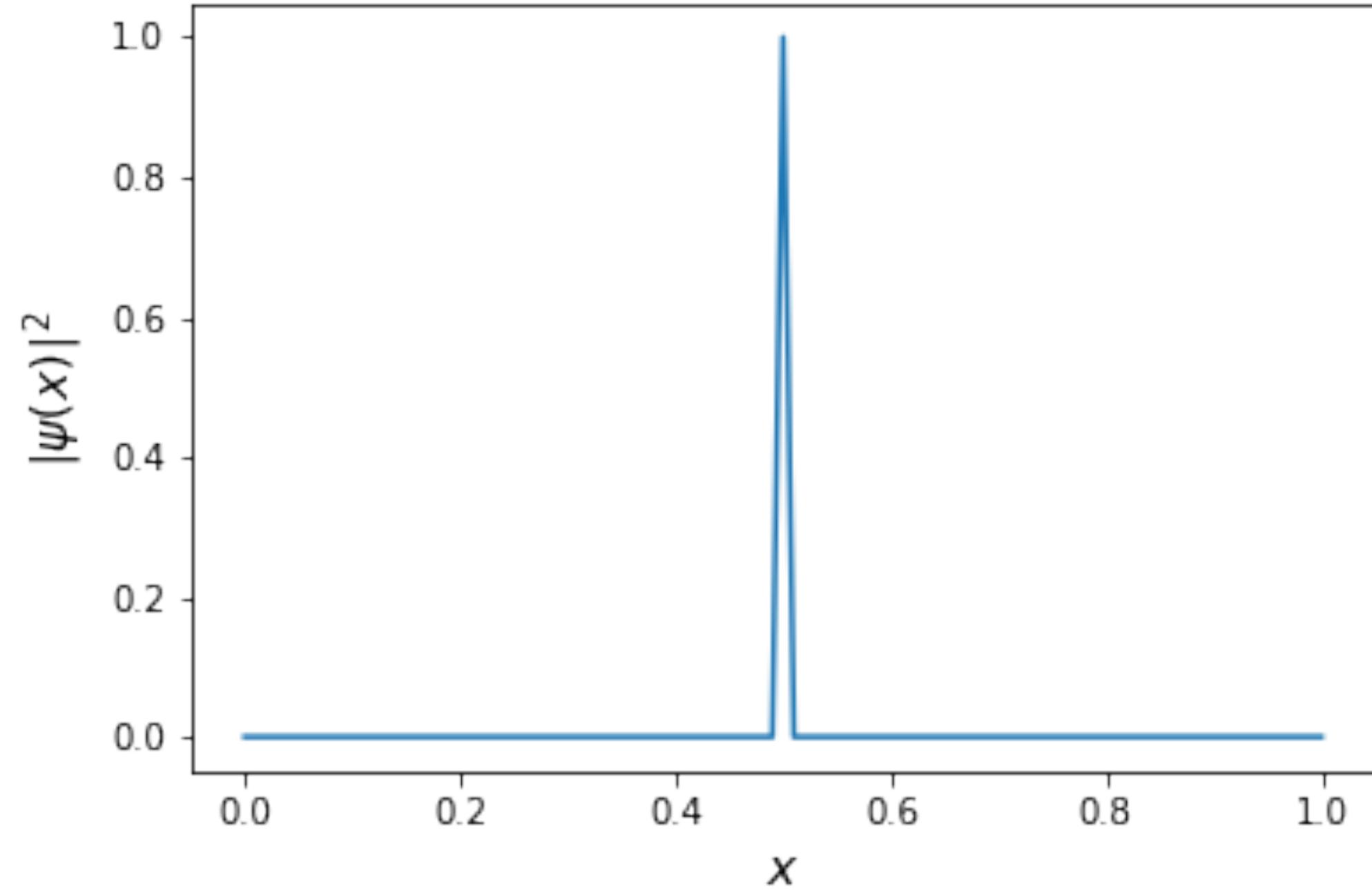
# Integration over position eigenstates



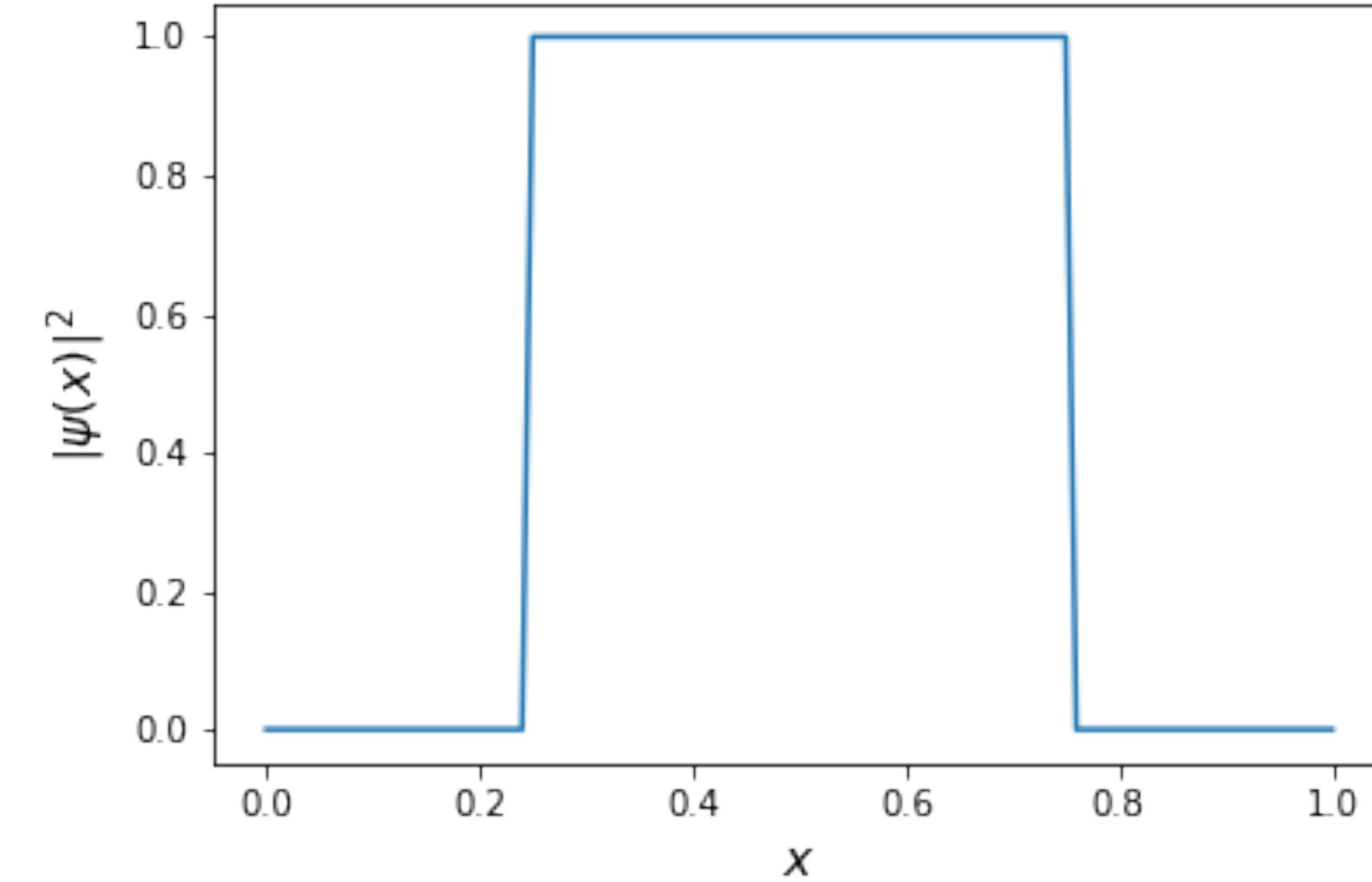
Have you seen this function before? Can you guess the mathematical expression for this function?



# A connection to uncertainty principle



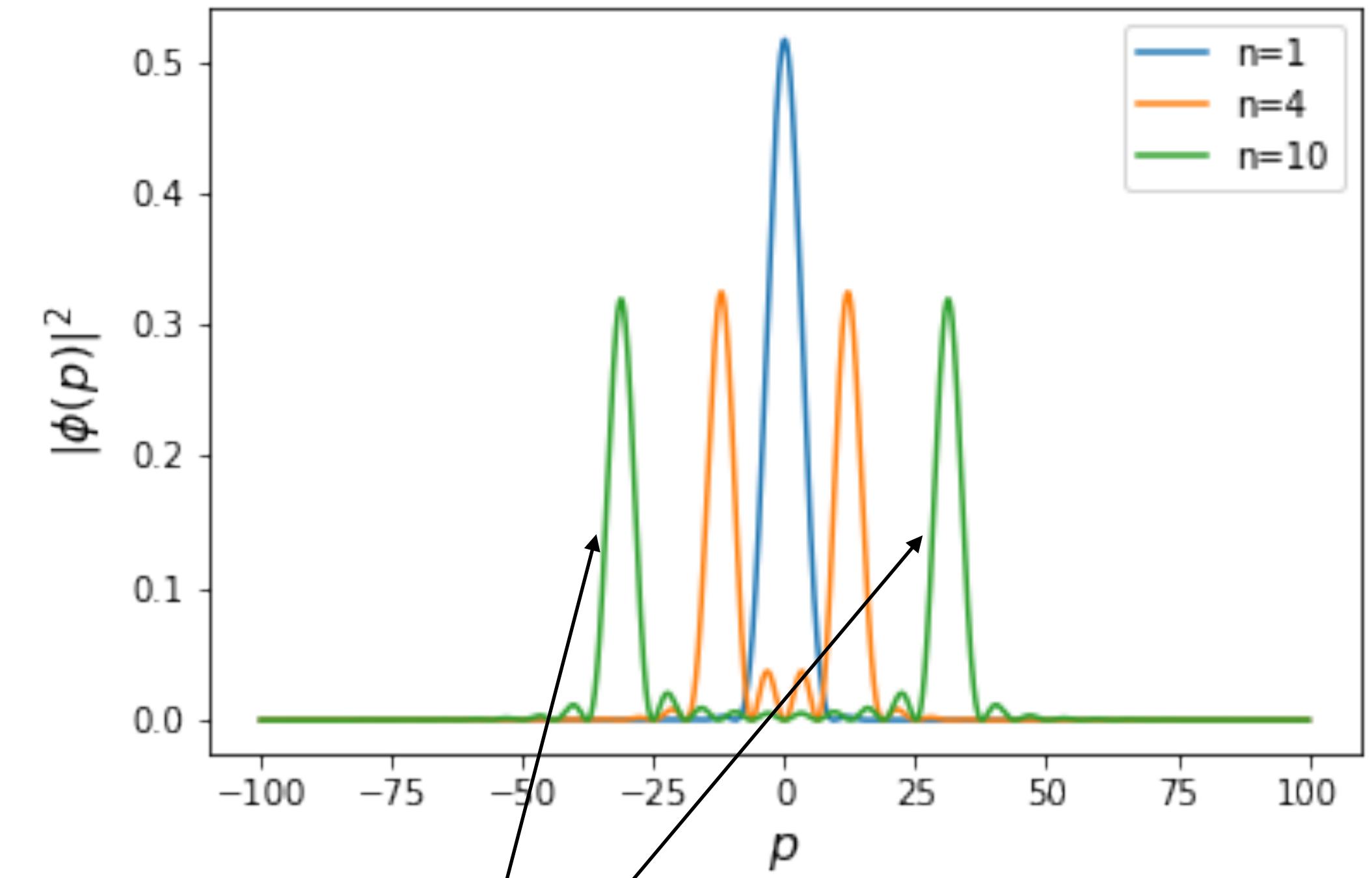
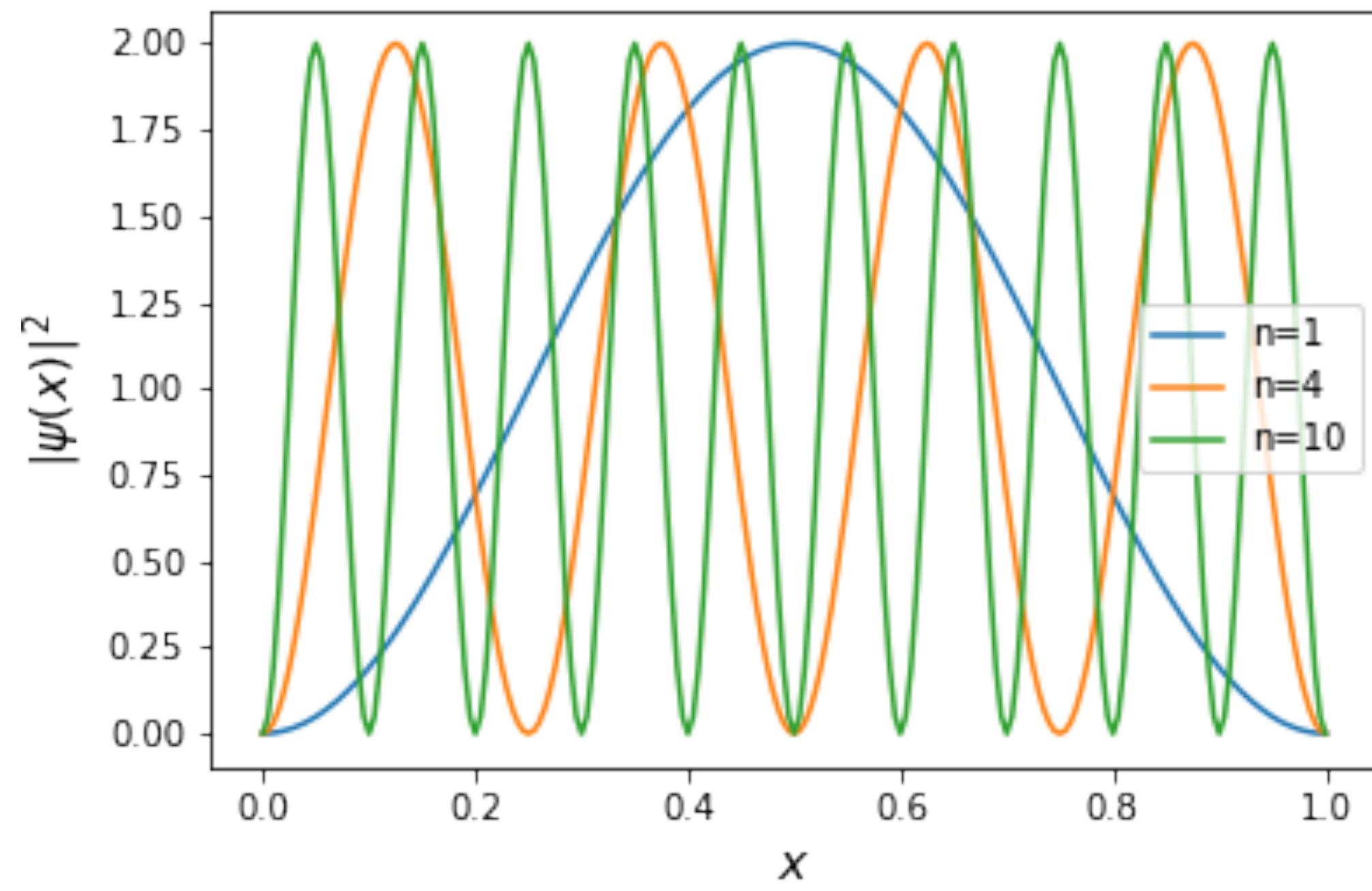
Experimentally, it is impossible to prepare a system in a position eigenstate,  $|x_0\rangle$ , where the wave function is precisely localised at one point.



When the position of a particle is determined experimentally, it is always found in a state integrated over the position eigenstates within the uncertainty,  $\Delta x$ , defined by the experimental resolution.

$$|\widetilde{x_0}\rangle = \int_{x_0-\Delta x/2}^{x_0+\Delta x/2} |x\rangle \langle x| x_0\rangle$$

# Particle-in-a-box in the $p$ -representation



high-energy states correspond to particles traveling towards left and right with definite momentum

# Homework

Exercise 2: Plot the probability distributions for a classical particle-in-a-box in position and momentum representations. Can you interpret these plots using the ideas discussed in the previous slides?



# A linear variational problem

An approximation technique in QM is to expand the eigenfunctions of a new problem as linear combinations of basis functions that are eigenfunctions of a known problem.

$$\psi_n^{\text{new}}(x) = \sum_k c_{k,n} \psi_k^{\text{known}}(x) \text{ or simply } \psi_n(x) = \sum_k c_{k,n} \chi_k(x)$$

The best wavefunctions for the new problem are obtained by varying the coefficients  $\{c_{k,n}\}$  that minimise the energy eigenvalues of the new problem.

The procedure for finding these coefficients is by representing the Hamiltonian operator of the new problem using the eigenfunctions of the known problem. The eigenvectors of the Hamiltonian matrix provide  $\{c_{k,n}\}$ .

Let's apply this method to find the eigenvalues and eigenfunctions of the quantum harmonic oscillator (new problem) in terms of particle-in-a-box (known problem) eigenfunctions.

# Numerical solution to the harmonic oscillator

```
In [4]: def fn_V(x):
    psi_i=np.sqrt(1/L)*np.sin((i+1)*(x-L)*np.pi/(2*L))
    psi_j=np.sqrt(1/L)*np.sin((j+1)*(x-L)*np.pi/(2*L))
    fn_V=psi_i * k * x**2/2 * psi_j
    return fn_V

for iN in range(0,6):

    N=2**iN      # No. of basis functions

    V=np.zeros([N,N])
    T=np.zeros([N,N])
    H=np.zeros([N,N])

    for i in range(N):
        for j in range(N):
            Int_V=integrate.quadrature(fn_V, -L, L,maxiter=1000)
            V[i][j]=Int_V[0]
            #kinetic energy part is same as in the particle-in-a-box
            T[i][i]=(i+1)**2 * hbar**2 * np.pi**2 / (8 * m * L**2)

    H=T+V

    E,V=eigen(H)

    print("Number of basis: ", N, ", ground state energy is:", E[0])
```

```
Number of basis:  1 , ground state energy is: 6.546885307943112
Number of basis:  2 , ground state energy is: 6.546885307943112
Number of basis:  4 , ground state energy is: 2.2409822186260655
Number of basis:  8 , ground state energy is: 0.7712925817042399
Number of basis: 16 , ground state energy is: 0.5026588346751345
Number of basis: 32 , ground state energy is: 0.499999999991960487
```

# Higher states converge slowly

```
In [5]: for i in range(N):
    print(i,E[i])
```

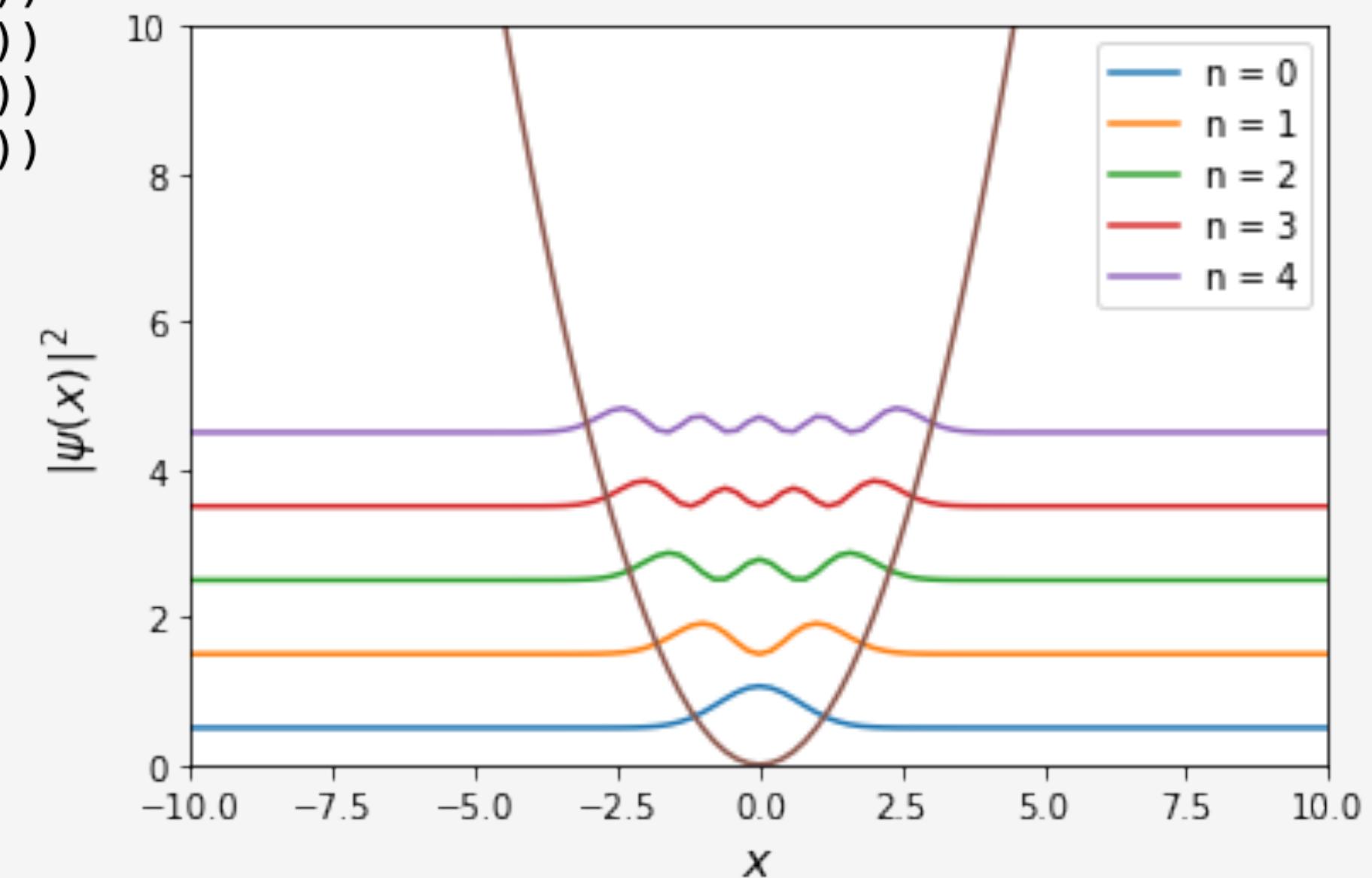
0 0.49999999991960487  
1 1.50000003533623  
2 2.500000246525915  
3 3.500000749476574  
4 4.50003826380895  
5 5.50009108730652  
6 6.500213561983046  
7 7.500380483848162  
8 8.50489994173305  
9 9.507109605274334  
10 10.550503775703861  
11 11.563037889597  
12 12.757188589561512  
13 13.790161054294524  
14 15.274948394336956  
15 16.323820558552228  
16 18.175821285164076  
17 19.22309685760712  
18 21.47078218078809  
19 22.495304266567338  
20 25.155539320689343  
21 26.135710969665308  
22 29.225350687266882  
23 30.13958386030048  
24 33.67696348822975  
25 34.50365514827044  
26 38.50946635546445  
27 39.22763463005119  
28 43.745025808894155  
29 44.34804530869218  
30 50.588924425384036  
31 51.22056108474399

# Eigenfunctions as linear combination of basis

```
In [6]: x=np.linspace(-L, L, 101)
Vharm=k*x**2/2
```

```
psi0=np.zeros(101)
psi1=np.zeros(101)
psi2=np.zeros(101)
psi3=np.zeros(101)
psi4=np.zeros(101)
for i in range(N):
    psi0=psi0+V[i][0]*np.sqrt(1/L)*np.sin((i+1)*(x-L)*np.pi/(2*L))
    psi1=psi1+V[i][1]*np.sqrt(1/L)*np.sin((i+1)*(x-L)*np.pi/(2*L))
    psi2=psi2+V[i][2]*np.sqrt(1/L)*np.sin((i+1)*(x-L)*np.pi/(2*L))
    psi3=psi3+V[i][3]*np.sqrt(1/L)*np.sin((i+1)*(x-L)*np.pi/(2*L))
    psi4=psi4+V[i][4]*np.sqrt(1/L)*np.sin((i+1)*(x-L)*np.pi/(2*L))

plt.plot(x,np.abs(psi0)**2+E[0])
plt.plot(x,np.abs(psi1)**2+E[1])
plt.plot(x,np.abs(psi2)**2+E[2])
plt.plot(x,np.abs(psi3)**2+E[3])
plt.plot(x,np.abs(psi4)**2+E[4])
plt.plot(x,Vharm)
plt.xlabel("$x$", fontsize=14)
plt.ylabel("$|\psi(x)|^2$", fontsize=14)
plt.savefig('psi_harm.png')
plt.legend(['n = 0', 'n = 1', 'n = 2', 'n = 3', 'n = 4'])
plt.ylim(0,10)
plt.show()
```



# One-dimensional molecule

The method can be extended easily to any arbitrary 1D problem. Let's apply to 1D diatomic molecule

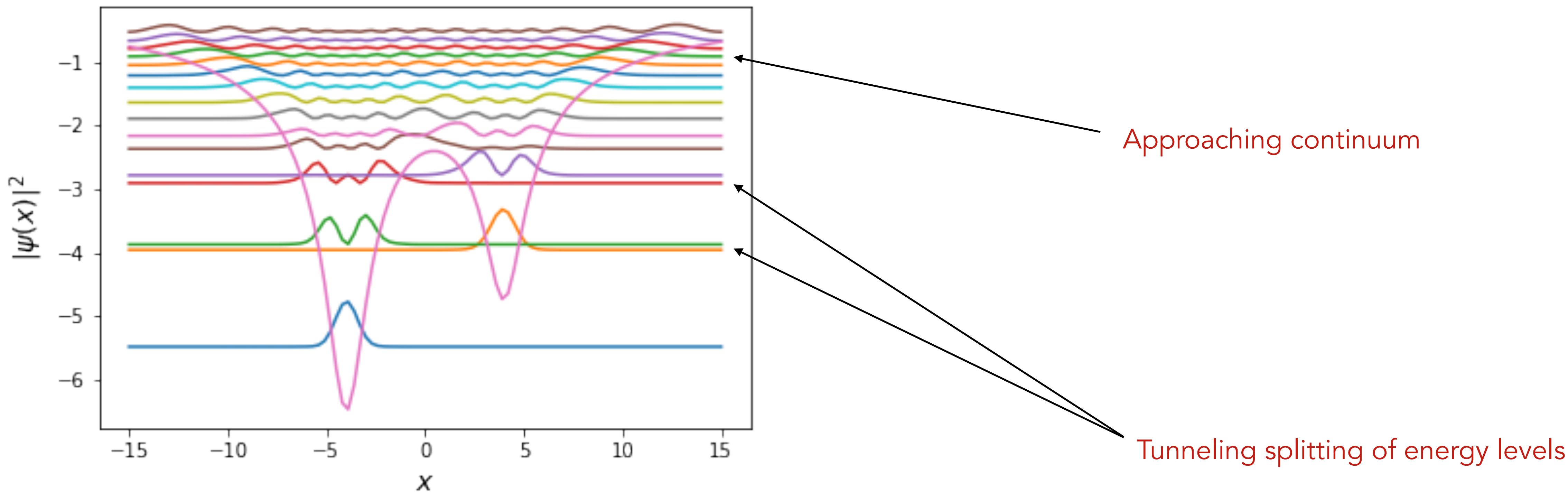
$$V(x) = - \left[ \frac{Z_A}{\sqrt{|x - x_A|^2 + \alpha}} + \frac{Z_B}{\sqrt{|x - x_B|^2 + \alpha}} \right]$$

The constant  $\alpha \geq 0$  'softens' the Coulomb potential (let's set it to 1.0). Can you think of some reasons for why should we soften the potential?

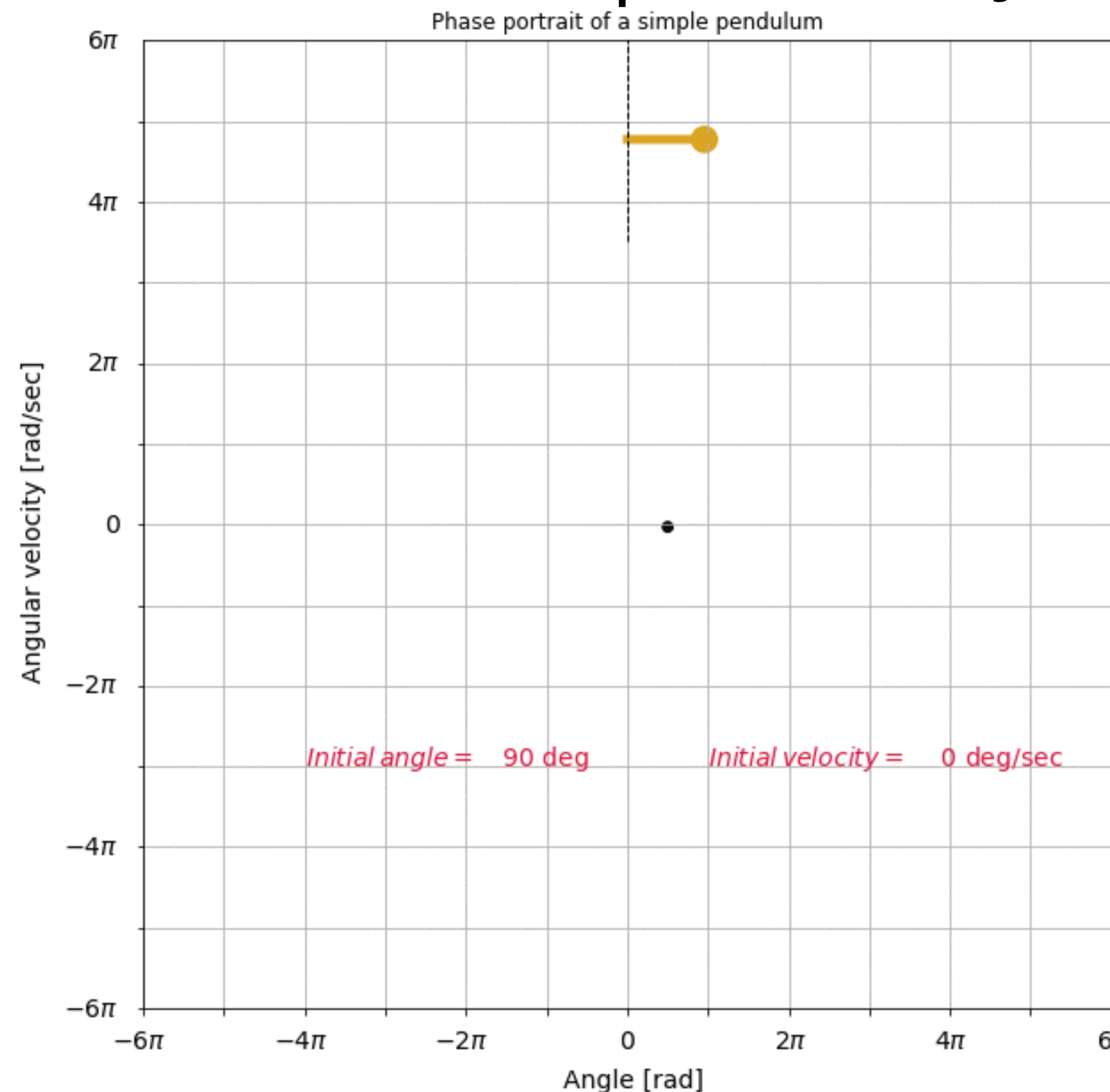


$$V(x) = - \left[ \frac{6}{\sqrt{|x+4|^2 + 1}} + \frac{4}{\sqrt{|x-4|^2 + 1}} \right]$$

A heteronuclear diatomic molecule



# Classical Phase Space Trajectories



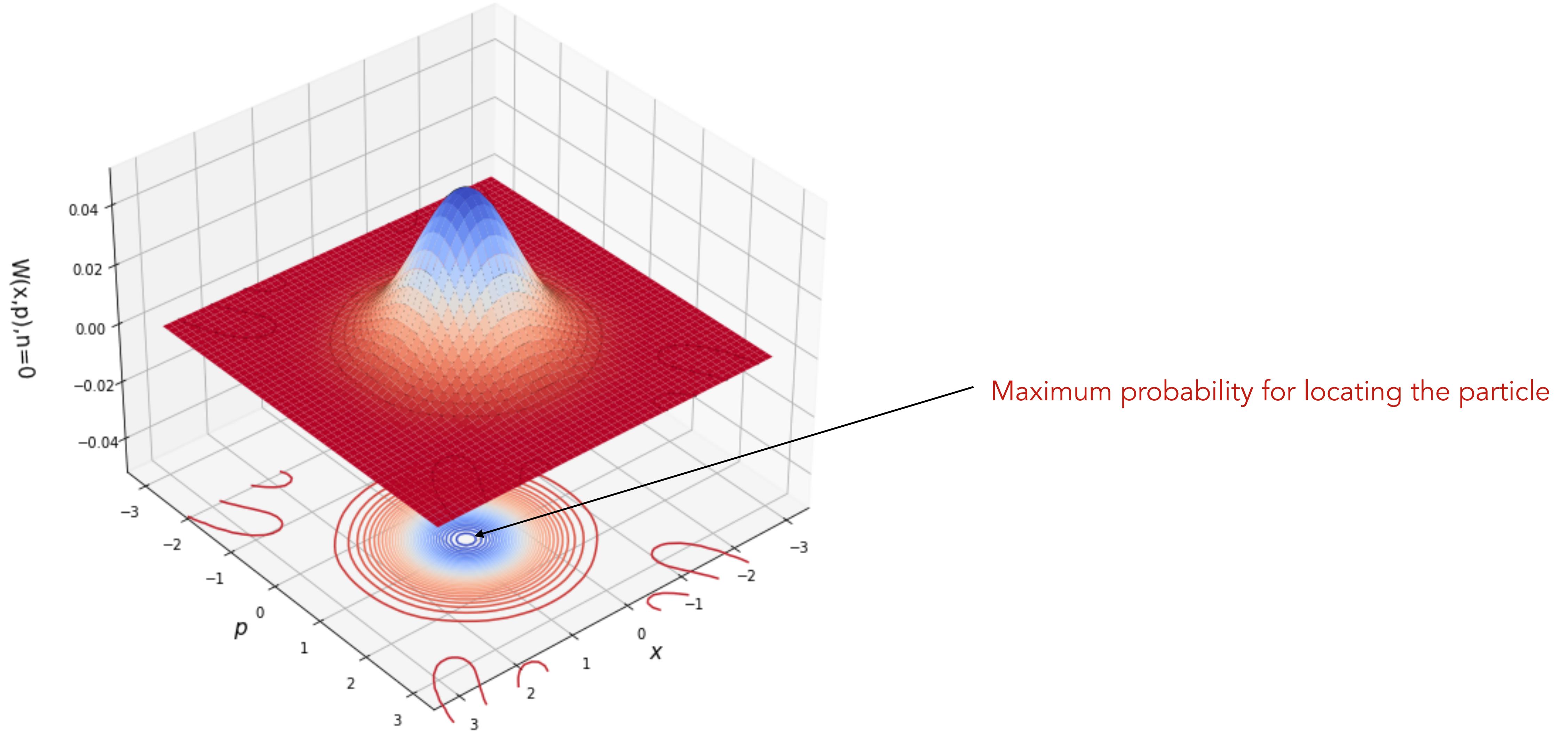
In QM, we have Wigner distribution in  $x$  and  $p$

$$W(x, p) = \frac{1}{4\pi^2} \int_{-\infty}^{+\infty} \psi^* \left( x + \frac{\hbar y}{2} \right) \psi \left( x - \frac{\hbar y}{2} \right) e^{ipx}$$

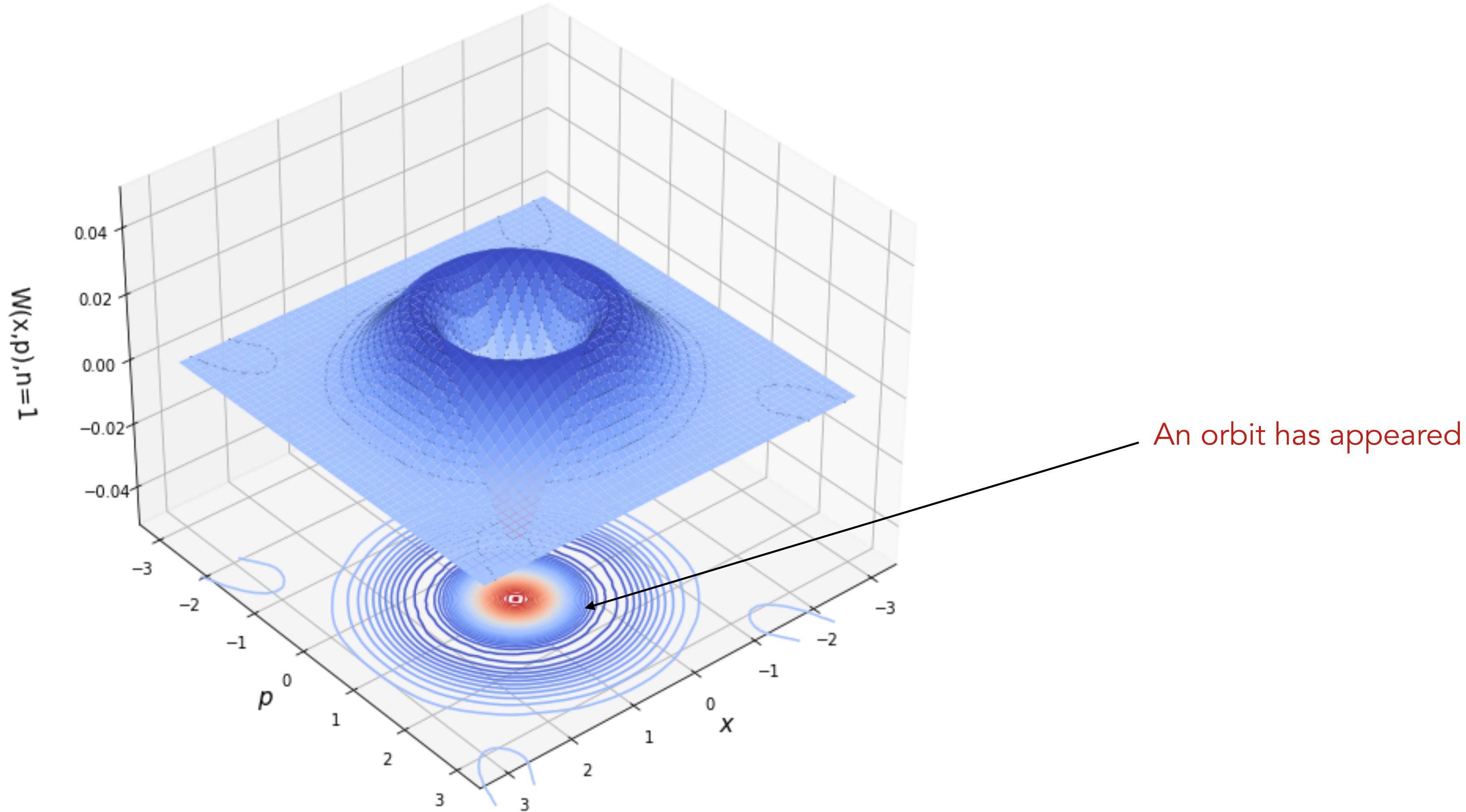
The Wigner distribution function is a quasi-probability distribution function. It can take negative values in small regions of phase space. Can you think of some ways to post-process the Wigner function to eliminate the negative values?



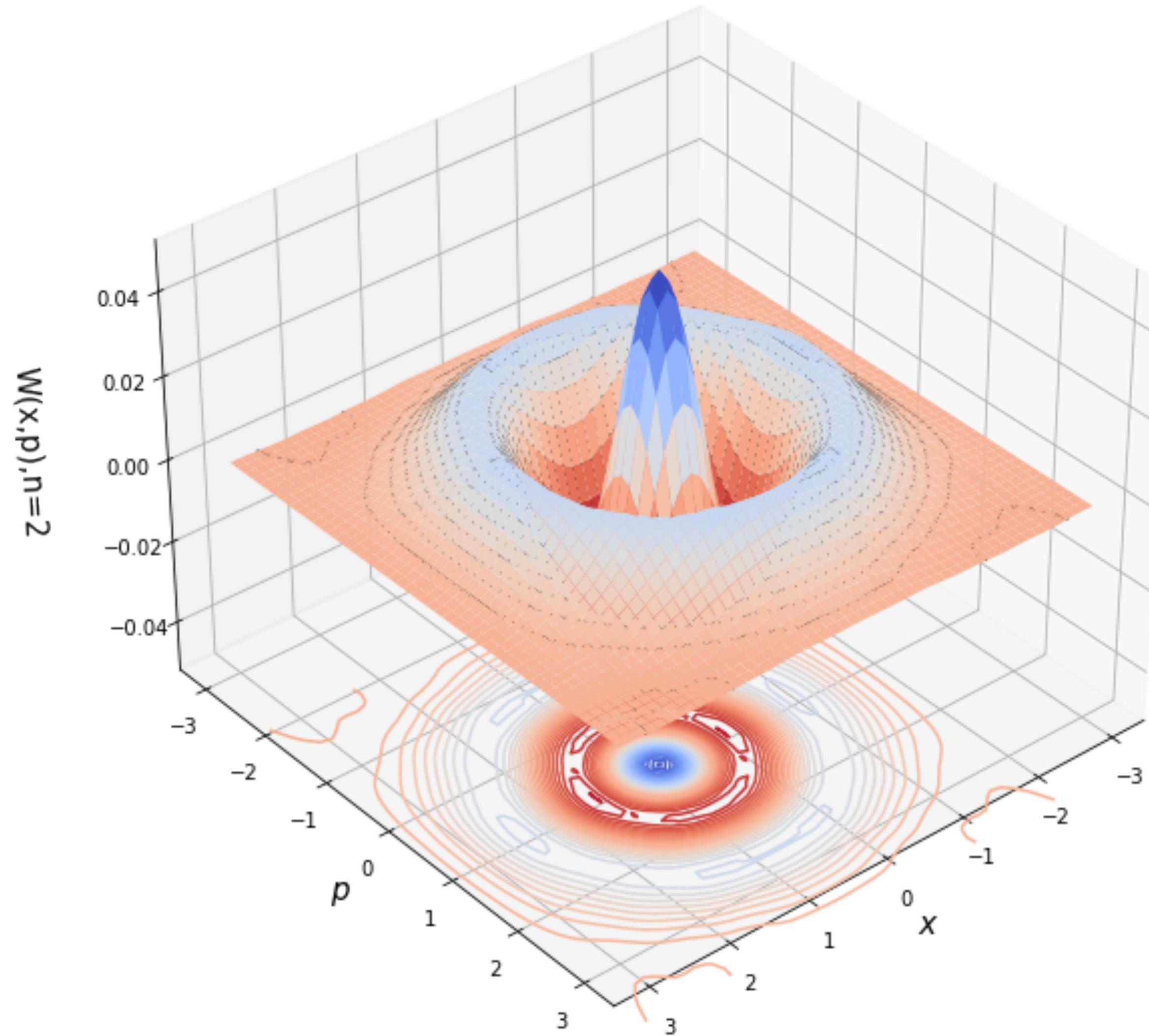
# $W(x, p)$ for the ground state of harmonic oscillator



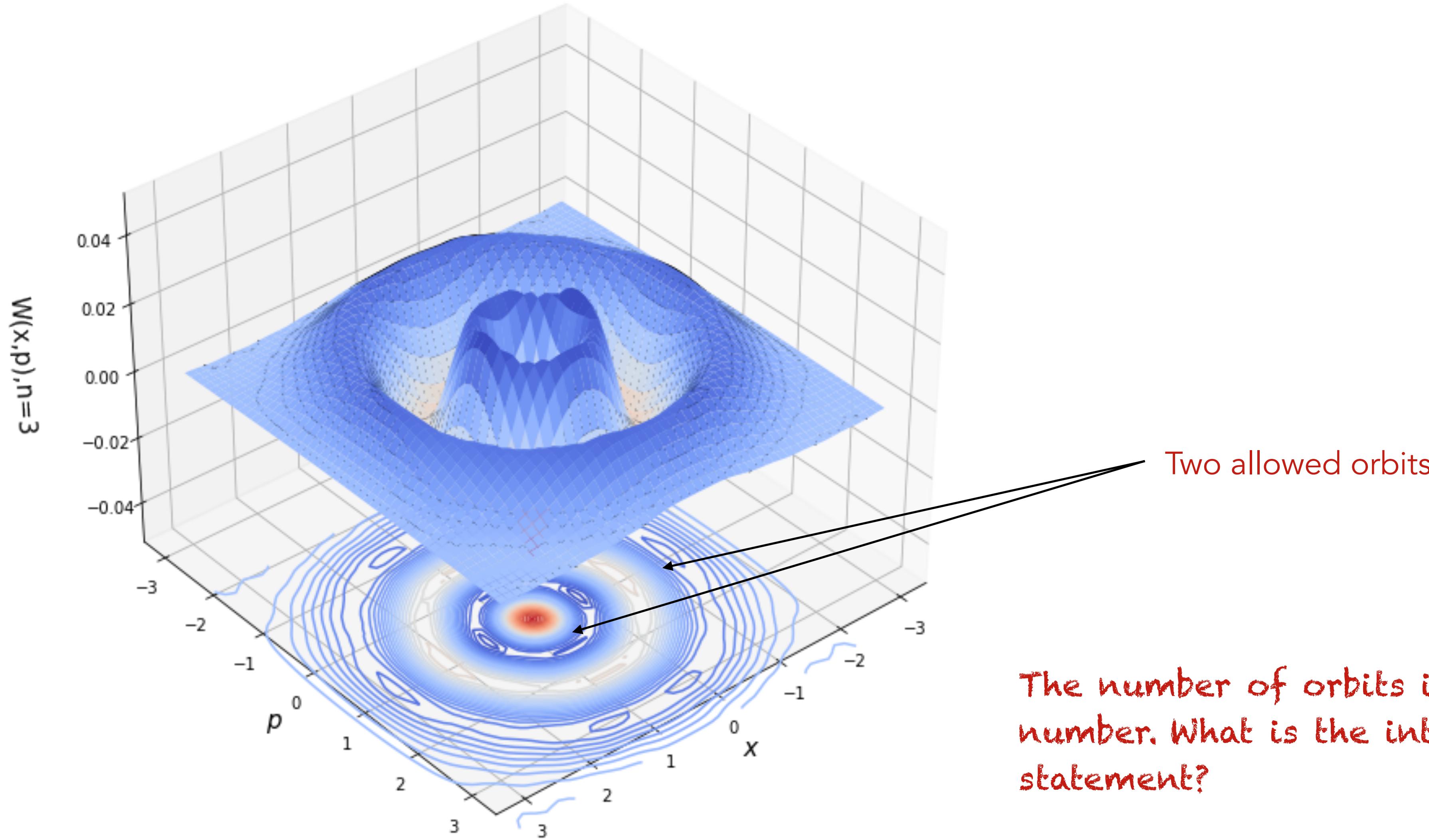
# $W(x, p)$ for the first excited state, $n = 1$



$W(x, p)$  for the first excited state,  $n = 2$



# $W(x, p)$ for the first excited state, $n = 3$



The number of orbits increase with the quantum number. What is the interpretation of this statement?



Thank you for your attention!  
[ramakrishnan@tifrh.res.in](mailto:ramakrishnan@tifrh.res.in)