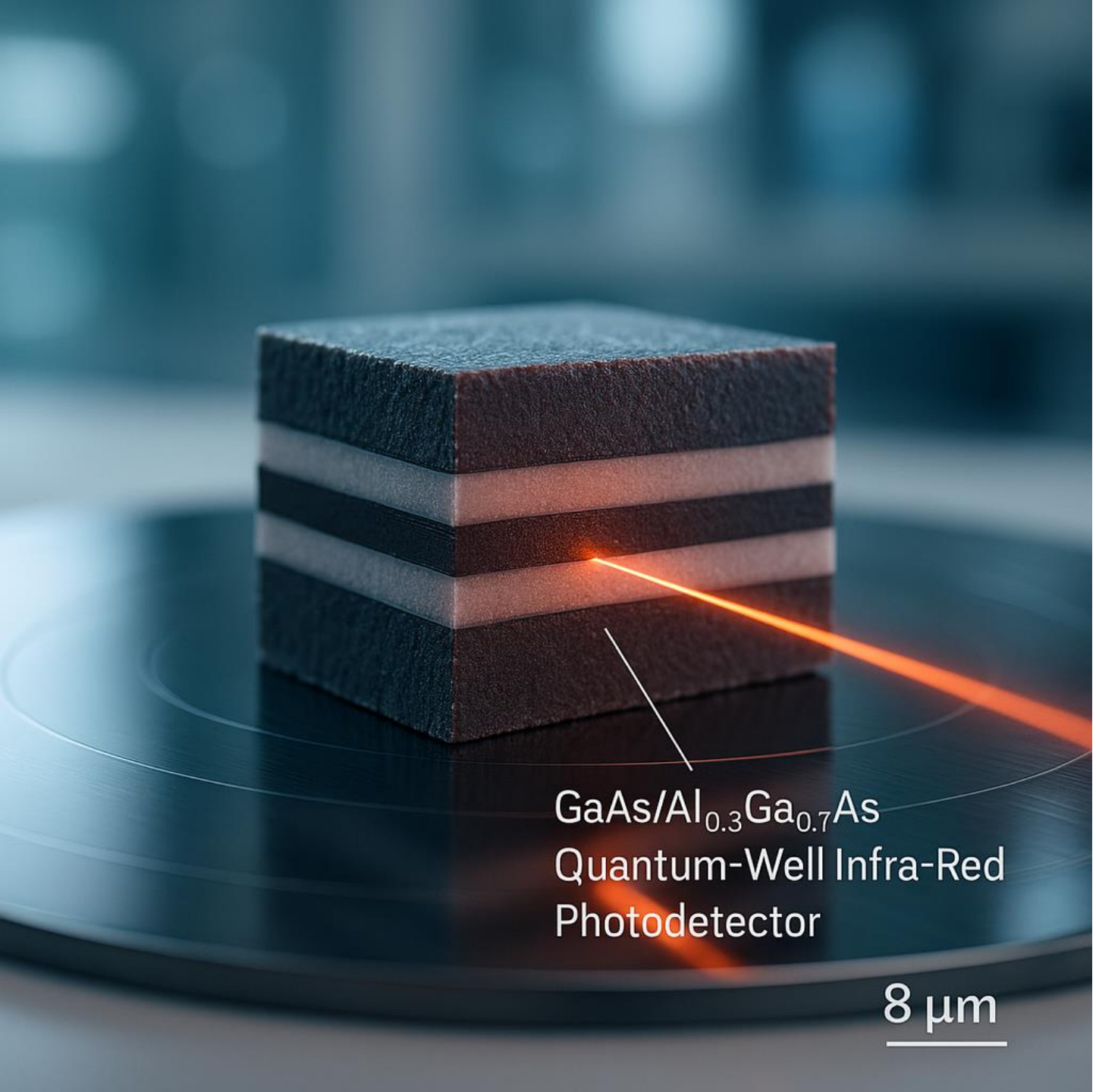


Designing a Quantum-Well Infra-Red Photodetector for 8 µm

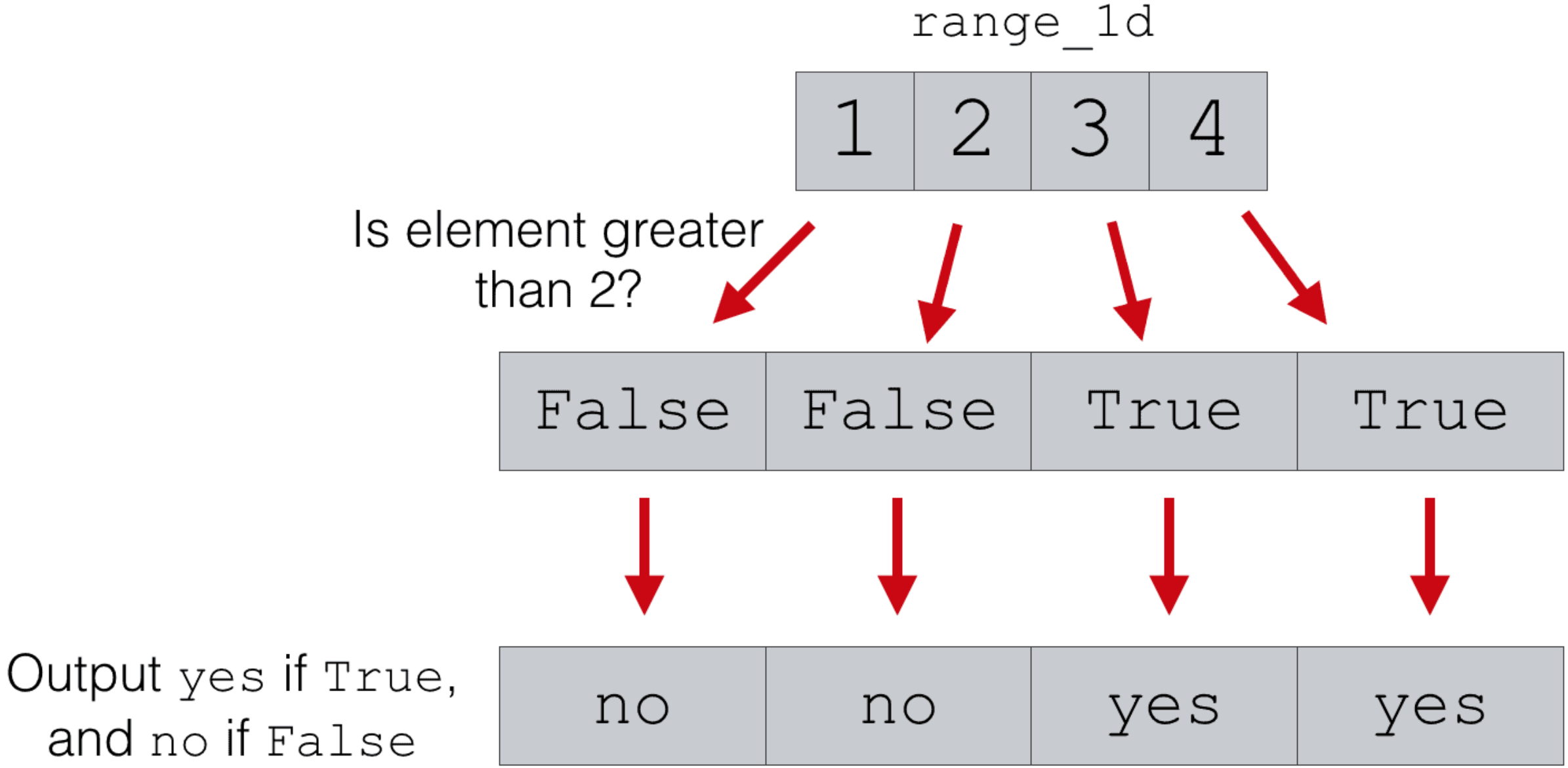
By AP Kai Lin Woon

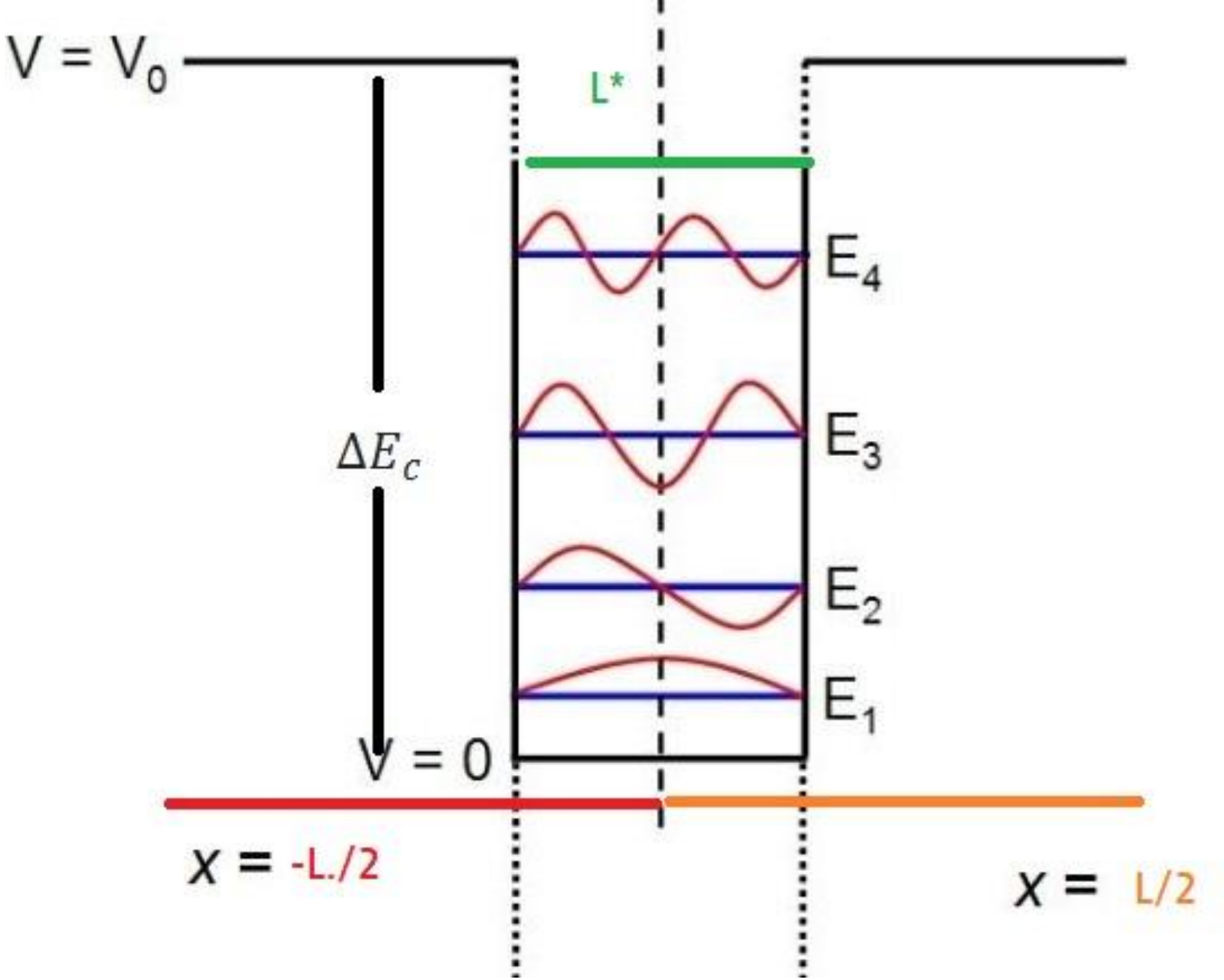
Test every element, and evaluate if element is greater than 2

Output this value if the condition evaluates as `True`

```
np.where(range_1d > 2, True, False)
```

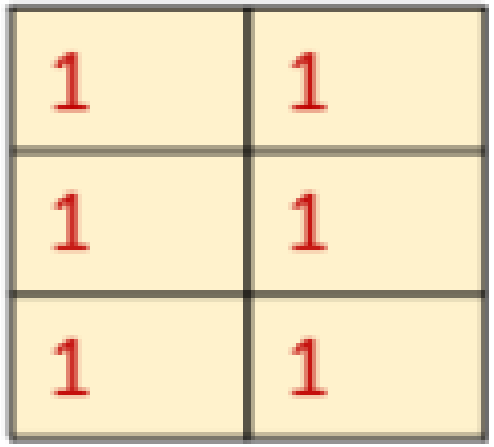
Output this value if the condition evaluates as `False`


range_1d
1
2
3
4
Is element greater than 2?
False
False
True
True
Output yes if True, and no if False
no
no
yes
yes




```
V = np.where(np.abs(z_nm) <= L_nm/2, 0.0, Delta_Ec)  # eV
```

a1=np.ones([3,2])



np.ones(N)

```
In [3]: np.ones(5)
Out[3]: array([1., 1., 1., 1., 1.])
```

```
w, v = eigh_tridiagonal(d, e)
```

$$T = \begin{pmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix}$$

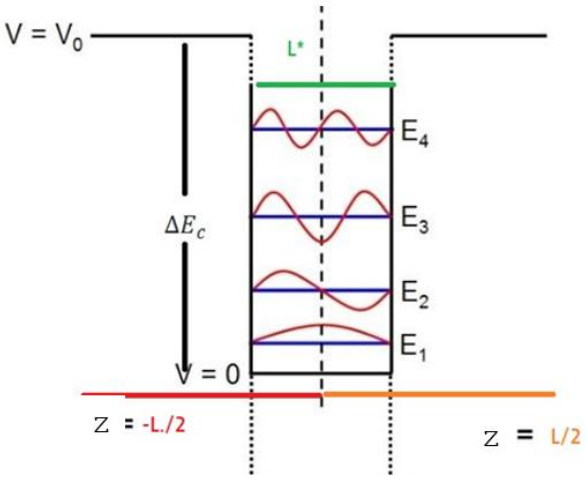
That matrix is completely specified by two 1-D arrays

- **d** – the main diagonal $[2, 2, 2, 2, 2]$
- **e** – the off-diagonal just above (and, because the matrix is symmetric, just below) the main diagonal $[-1, -1, -1, -1]$

Effective-Mass Schrödinger Equation

We solve the 1D time-independent effective-mass Schrödinger equation:

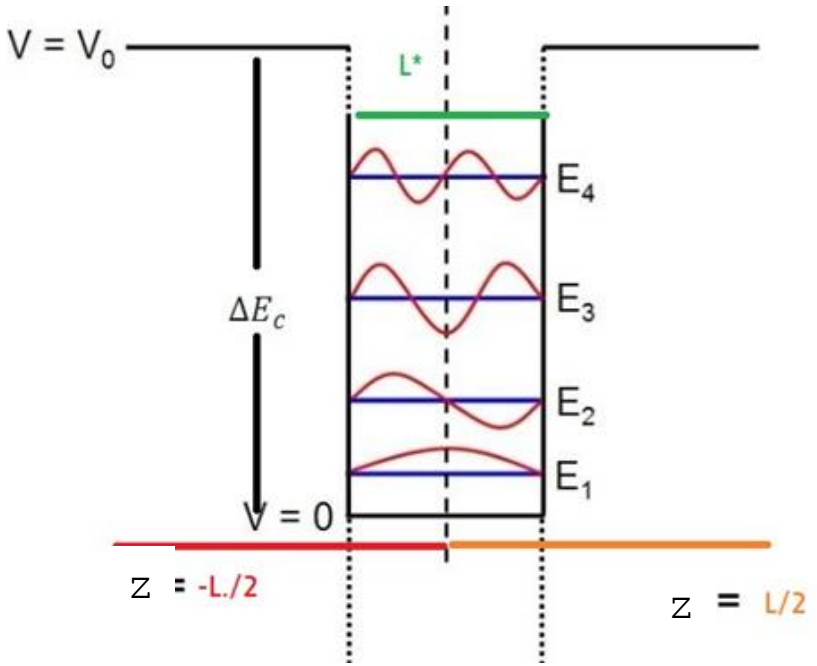
$$\frac{\hbar^2}{2m^*}\frac{d^2\psi(z)}{dz^2} + V(z)\psi(z) = E\psi(z)$$



A quantum well of width L centered at z=0 with barrier height ΔEc:

$$V(z) = \begin{cases} 0, & |z| \le \frac{L}{2} \\ \Delta E_c, & |z| > \frac{L}{2} \end{cases}$$

Discretization

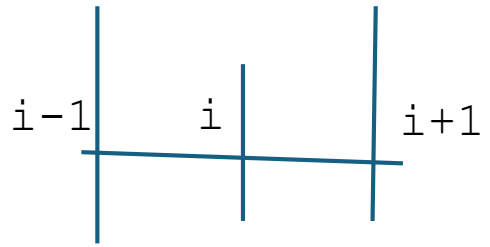


Define grid points:

$$z_i = z_0 + i\Delta z, \ldots, L/2$$

Second derivative using central differences:

$$\left.\frac{d^2\psi}{dz^2}\right|_{z_i} \approx \frac{\psi_{i-1} - 2\psi_i + \psi_{i+1}}{(\Delta z)^2}$$



Hamiltonian Matrix

The Hamiltonian matrix H is tridiagonal with elements:

$$H_{ii} = \frac{\hbar^2}{m^* (\Delta z)^2} + V(z_i)$$

$$H_{i,i\pm1} = -\frac{\hbar^2}{2\, m^* (\Delta z)^2}.$$

Eigenvalue Problem

Solve the matrix equation:

$H\Psi=E\Psi$

and extract the lowest eigenvalue E_1 as the ground-state energy.

Finite-Difference Hamiltonian Matrix

- For a grid of N points $z_1, \ldots z_N$ with spacing Δx, the finite-difference Hamiltonian in the effective-mass approximation is the N×N tridiagonal matrix:

$$H = \begin{pmatrix} d_1 & -t & 0 & \cdots & 0 \\ -t & d_2 & -t & \ddots & \vdots \\ 0 & -t & d_3 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -t \\ 0 & \cdots & 0 & -t & d_N \end{pmatrix}$$

Setting the code

```
from scipy.linalg import eigh_tridiagonal


# select='i', select_range=(0,0) asks only for
the lowest eigenpair
   eigvals, eigvecs = eigh_tridiagonal(main_diag,
   off_diag,select='i', select_range=(0,0))
```

Tells SciPy to select eigenvalues by index rather than computing all of them

Return only the eigenvalue whose index is 0

Setting the code

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.constants import hbar, m_e, e
from scipy.linalg import eigh_tridiagonal


# Physical constants
m_star = 0.066 * m_e    # effective mass inside well
eV_to_J = e  # convert eV to Joule
J_to_eV = 1 / e # convert Joule to eV

Delta_Ec_nominal = 0.30  # eV  Bandgap
hv = 0.155               # eV   emission wavelength  in eV given 8um
```

Setting the code

```
def ground_state_energy(L_nm, Delta_Ec=Delta_Ec_nominal, dz_nm=0.05):
    """
    Ground state energy (eV) for a finite square well of width L_nm
(nm).
    """
    pad_nm = 10.0
    z_nm = np.arange(-pad_nm - L_nm/2, pad_nm + L_nm/2 + dz_nm,
dz_nm) # length of quantum well
    N = len(z_nm)
    V = np.where(np.abs(z_nm) <= L_nm/2, 0.0, Delta_Ec)  # eV

    dz = dz_nm * 1e-9  # meters
    prefactor = (hbar**2) / (2 * m_star * dz**2) * J_to_eV  # eV

    main_diag = 2 * prefactor + V
    off_diag = -prefactor * np.ones(N-1)
```

$$H_{ii} = \frac{\hbar^2}{m^*\,(\Delta z)^2} + V(z_i)$$

$$H_{i,i\pm1} = -\frac{\hbar^2}{2\,m^*\,(\Delta z)^2}.$$

why not np.ones(N)

a tridiagonal $N \times N$ matrix has only $N-1$ super- (and sub-) diagonal entries, not N.

For any square matrix

$$\begin{pmatrix} d_0 & e_0 & & & \\ e_0 & d_1 & e_1 & & \\ & e_1 & d_2 & \ddots & \\ & & \ddots & \ddots & e_{N-2} \\ & & & e_{N-2} & d_{N-1} \end{pmatrix}$$

the **main diagonal** contains $d_0, \ldots, d_{N-1} \Rightarrow N$ **numbers**;

the **off-diagonal** contains $e_0, \ldots, e_{N-2} \Rightarrow N - 1$ **numbers**

Continue from def ground_state_energy function

```
    eigvals, eigvecs = eigh_tridiagonal(main_diag, off_diag, select='i', select_range=(0,0))
    E1 = eigvals[0]          # the ground-state energy
    psi1 = eigvecs[:,0]  # the corresponding eigenvector (wavefunction on the grid)
    # normalize
    norm = np.trapz(np.abs(psi1)**2, z_nm*1e-9)
    psi1 /= np.sqrt(norm)
    return E1, z_nm, V, psi1
```

Normalisation condition

In continuum quantum mechanics a bound-state wave-function must satisfy the normalisation condition

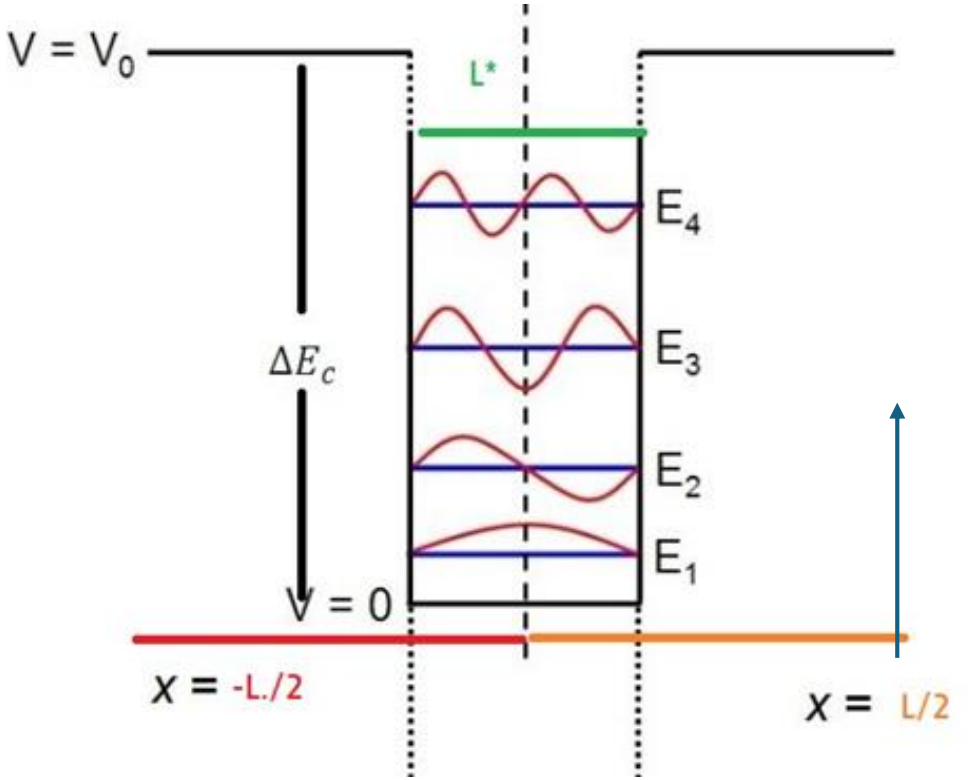
$$\int_{-\infty}^{\infty} |\psi(z)|^2 \, \mathrm{d}z \; = \; 1$$

The eigenvector returned by a linear-algebra routine has an arbitrary scale so its norm is almost never 1.

$$\mathbf{norm} \; = \; \sum_i |\psi_1(z_i)|^2 \, \Delta z$$

```
norm = np.trapz(np.abs(psi1)**2, z_nm*1e-9)
```

Mechanism of emission



E_1 to which state
as a result of absorption of
0.155 eV (8um) photon?

```
target = Delta_Ec_nominal - hv
```

Setting the code

```
# Sweep L values
L_values = np.linspace(0.0, 10.0, 1000)
E1_values = np.array([ground_state_energy(L)[0] for L in
L_values])


target = Delta_Ec_nominal - hv
tolerance = 0.001  # eV


indices = np.where(np.abs(E1_values - target) <=
tolerance)[0]
L_star = L_values[indices[0]] if len(indices)>0 else
None
```

Among all the well widths L we swept, which ones give a ground-state energy E1(L) that matches our design target to within 1 meV?

np.where(condition)

```
import numpy as np


a = np.array([10, 11, 12, 13, 14])
mask_1d = a % 2 == 0
idx_tuple_1d = np.where(mask_1d)
idx_array_1d = np.where(mask_1d)[0]
```

```
1-D mask: [ True False  True False  True]

In [15]: mask_1d[0]
Out[15]: True

In [16]: idx_tuple_1d
Out[16]: (array([0, 2, 4], dtype=int64),)

In [17]: idx_array_1d
Out[17]: array([0, 2, 4], dtype=int64)
```

Always a **tuple**, where each element is an array of indices along one dimension of the input.

- For a 1-D condition → tuple length 1: (array_of_indices,)
- For a 2-D condition → tuple length 2: (row_indices, column_indices)

```
L_star = L_values[indices[0]] if len(indices)>0 else None
```

```
In [19]: indices
Out[19]: array([315, 316, 317, 318, 319], dtype=int64)
```

indices[0] to pull the corresponding width and energy out of the sweep arrays.

L_values[indices[0]] is therefore the *actual physical width* in nanometres of the first well that works.

How sensitive the design is to a ±5% uncertainty in ΔE_c (re-optimise for $\Delta E_c = 0.285\text{eV}$ and 0.315eV)

```python
def optimise_L_for_DeltaEc(Delta_Ec_new):
    E1_vals = [ground_state_energy(L, Delta_Ec=Delta_Ec_new)[0] for L in L_values]
    target_new = Delta_Ec_new - hv
    idx = [i for i, E in enumerate(E1_vals) if abs(E - target_new) <= tolerance]
    return (L_values[idx[0]], E1_vals[idx[0]]) if idx else (None, None)
```

List comprehension repeats the Schrödinger solve for every trial width in L_values *using the new barrier height*.

Builds a list of indices whose computed $E1$ values fall within the ±1 meV tolerance band around the new target. Enumerate lets us keep track of the index i while looping through E1_vals

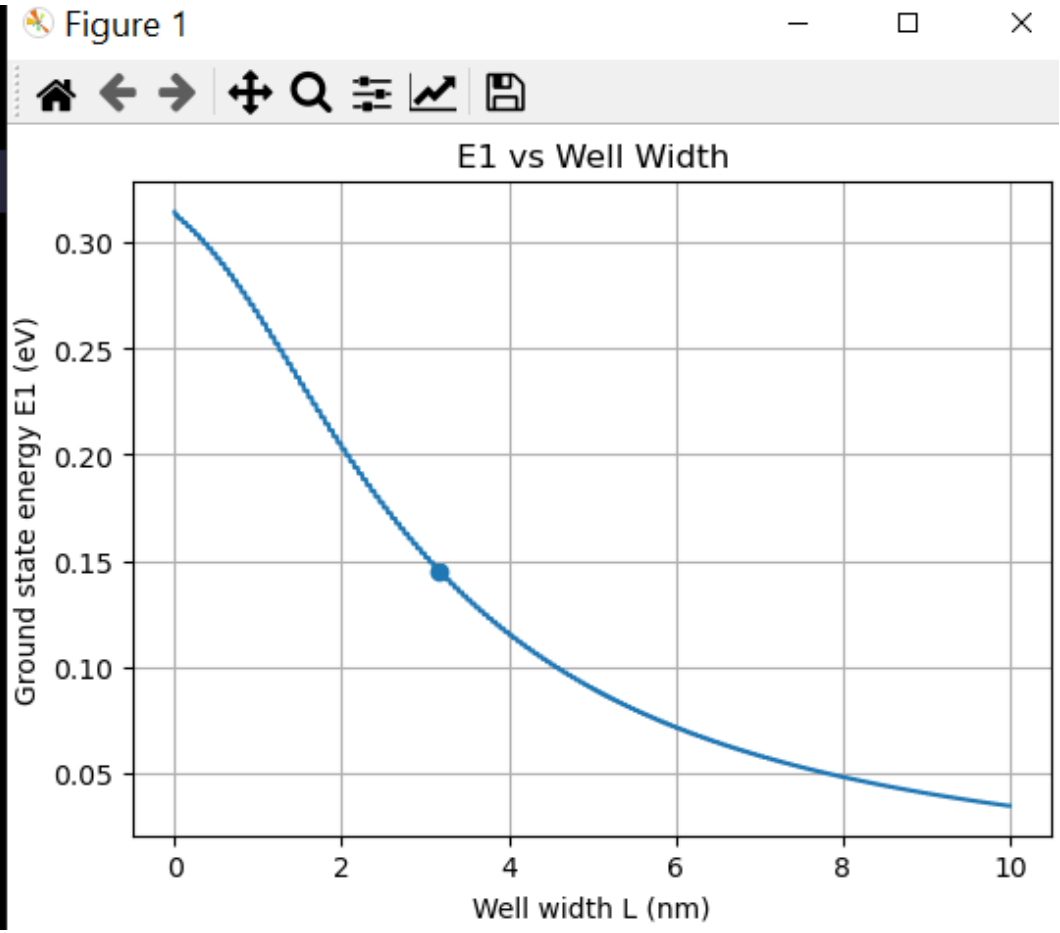
```
 idx = [i for i, E in
enumerate(E1_vals)
              if abs(E - target_new)
<= tolerance]
```

i	L_values[i] (nm)	E1_vals[i] (eV)	abs(E - target) (eV)	Condition met?
0	2.00	0.280	0.135	No
…	…	…	…	…
75	5.75	0.145	0.0008	**Yes**
76	5.80	0.144	0.0013	No
…	…	…	…	…

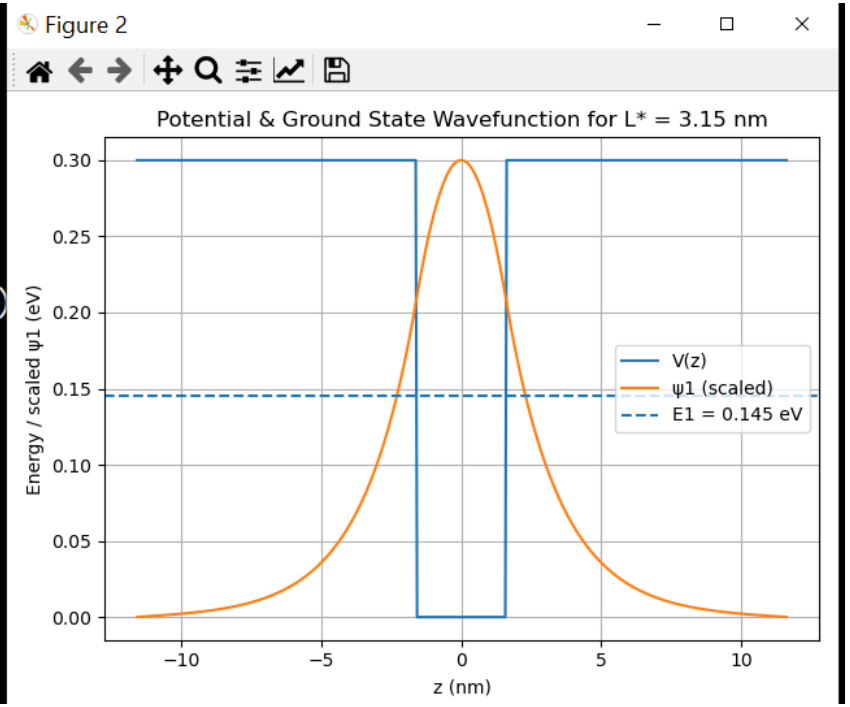
After the comprehension runs,
idx = [75]

```python
L_minus, E1_minus = optimise_L_for_DeltaEc(0.285)
L_plus,  E1_plus  = optimise_L_for_DeltaEc(0.315)

# Plot E1(L)
plt.figure()
plt.plot(L_values, E1_values)
if L_star is not None:
    plt.scatter([L_star], [E1_values[indices[0]]])
plt.xlabel('Well width L (nm)')
plt.ylabel('Ground state energy E1 (eV)')
plt.title('E1 vs Well Width')
plt.grid(True)
plt.tight_layout()
plt.show()
```



```python
# Potential and wavefunction for L*
if L_star is not None:
    E1_star, z_nm_star, V_star, psi1_star = ground_state_energy(L_star)
    plt.figure()
    plt.plot(z_nm_star, V_star, label='V(z)')
    scale = Delta_Ec_nominal
    plt.plot(z_nm_star, psi1_star/np.max(np.abs(psi1_star))*scale, label='ψ1 (scaled)')
    plt.axhline(E1_star, linestyle='--', label=f'E1 = {E1_star:.3f} eV')
    plt.xlabel('z (nm)')
    plt.ylabel('Energy / scaled ψ1 (eV)')
    plt.title(f'Potential & Ground State Wavefunction for L* = {L_star:.2f} nm')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```



```
results = {
    "L_star (nm)": L_star,
    "E1(L_star) (eV)": E1_values[indices[0]] if len(indices)>0 else None,
    "L_star (DeltaEc -5%) (nm)": L_minus,
    "L_star (DeltaEc +5%) (nm)": L_plus
}
```

```
In [24]: results
Out[24]:
{'L_star (nm)': 3.153153153153153,
 'E1(L_star) (eV)': 0.1450174331349965,
 'L_star (DeltaEc -5%) (nm)': 3.4534534534534536,
 'L_star (DeltaEc +5%) (nm)': 2.9029029029029028}
```