PHYS 142/242

Lecture 10: Eigenvalue Problem

Javier Duarte — January 29, 2025

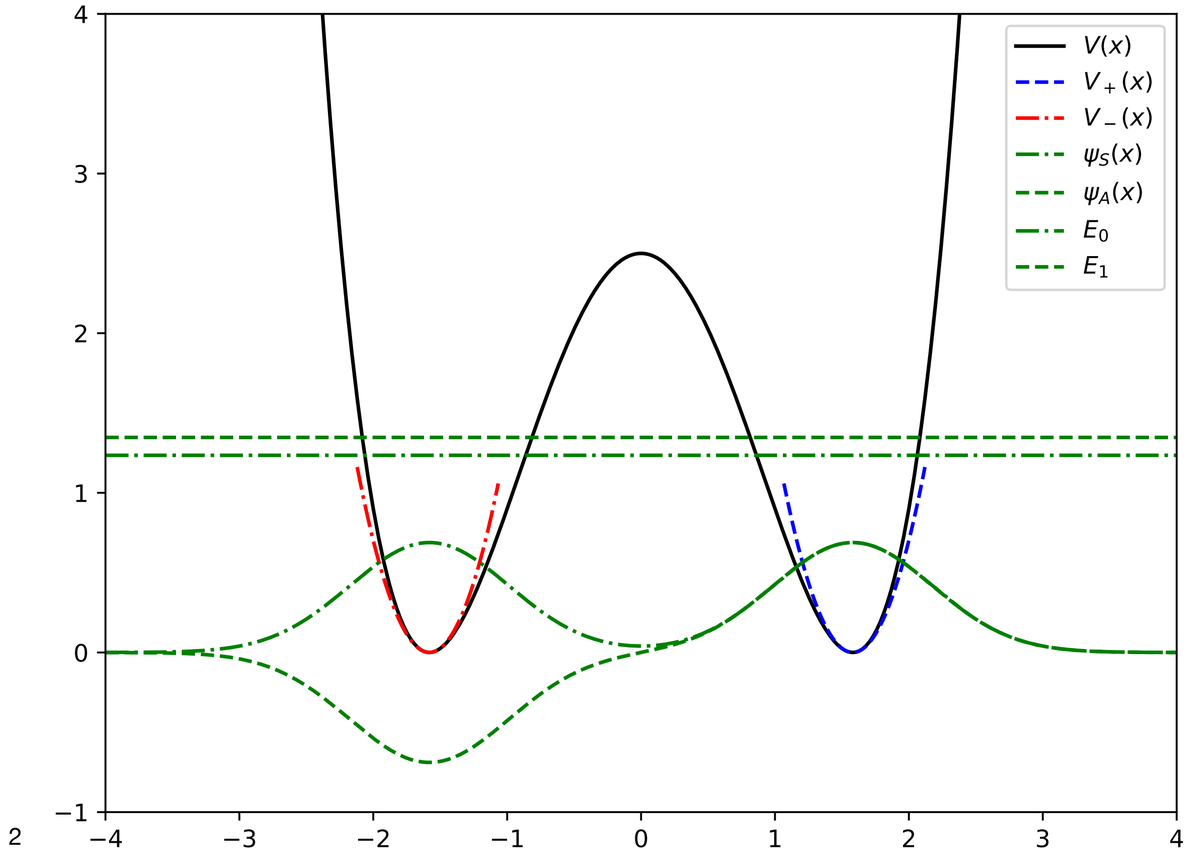
Midterm preview

- In the midterm, we will numerically solve for the two lowest energy eigenvalues of the double well
- We will sketch the method today with the HO

TABLE I. Eigenvalues of the two-well oscillator in the small-λ regime. $\epsilon_n(\lambda)$ are the computed exact eigenvalues of the energy-shifted operator $H(1,\lambda) + (1/4\lambda)$, which is positive definite.

λ	ϵ_0 ϵ_1	ϵ_2 ϵ_3
0.01	1.404 048 605 297 7[a]	4.170 193 605 999 3
	1.404 048 605 297 7	4.170 193 605 999 3
0.02	1.393 527 585 044 2	4.092 028 112 820 5
	1.393 527 587 151 0	4.092 028 608 428 7
0.03	1.382 601 444 053 8	4.006 049 199 465 7
	1.382 605 783 831 4	4.006 655 466 749 5
0.04	1.371 122 236 557 5	3.901 359 951 813 1
	1.371 308 461 612 9	3.918 263 337 997 1
0.05	1.358 422 103 747 8	3.746 917 080 727 9
	1.360 133 597 773 3	3.848 838 300 057 4
0.07	1.323 374 074 208 5	3.342 216 720 258 7
	1.343 365 616 287 4	3.833 129 937 607 9
0.10	1.234 507 162 786 0	3.009 488 545 436 2
	1.346 940 868 922 5	4.043 546 039 767 6
0.15	1.062 499 247 956 5	3.033 667 276 570 6
	1.421 086 890 539 3	4.589 838 495 543 4
0.17	1.007 165 158 778 7	3.118 337 642 119 7
	1.464 225 132 421 2	4.816 923 221 196 9
0.20	0.941 750 342 076 9	3.270 377 801 715 3
	1.535 530 204 085 8	5.148 274 740 096 0

[a] Since near the minima the potentail function $\sim 2x^2 + O(\lambda^{1/2}x^3)$, $\epsilon_0 \to \sqrt{2}$ (ground-state energy in a potential $2x^2$) as $\lambda \to 0$. We find $\epsilon_0(\lambda = 0.001) = 1.413\,211\,965\,792$.



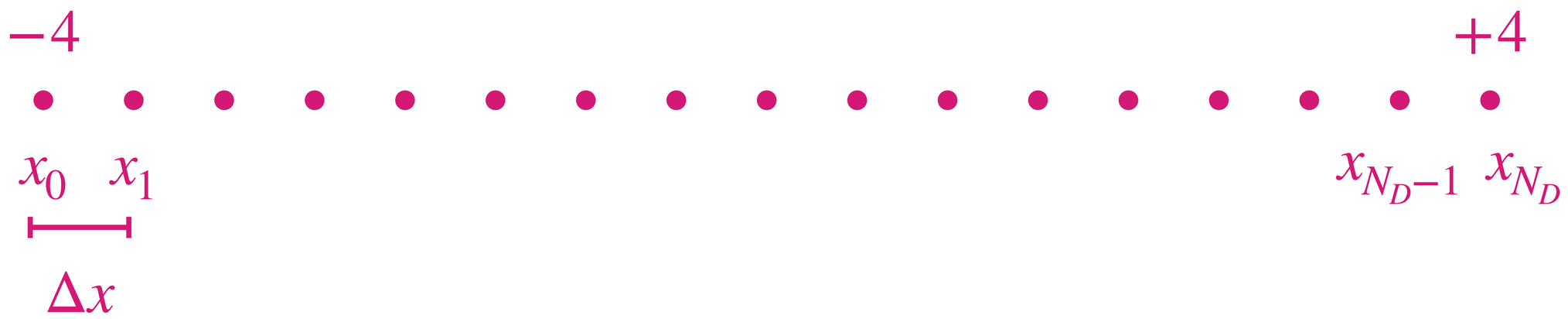
Numerical solution

We want to solve the following eigenvalue equation

$$\hat{H}\phi_n(x) = -\frac{1}{2}\frac{\partial^2\phi_n(x)}{\partial x^2} + V(x)\phi_n(x) = E_n\phi_n(x)$$

where $V(x) = \frac{1}{2}x^2$

We will discretize the problem as usual



Discrete Hamiltonian

In the discrete case, the wave function ϕ_n is a vector $\phi_n^i = \phi_n(x_i)$ of length $N_D + 1$ and the Hamiltonian is an $(N_D + 1) \times (N_D + 1)$ matrix

$$\sum_j H_{ij}\phi_n^j = E_n\phi_n^i$$

What does H_{ij} look like?

The potential part is diagonal $\frac{1}{2}x_i^2\phi_n^i = \frac{1}{2}x_i^2\delta_{ij}\phi_n^j \subset H_{ij}\phi_n^j$

What about the kinetic part?

Numerical solution

The second derivative (or Laplacian operator) on a grid can be approximated as

$$\frac{\partial^2 \phi_n}{\partial x^2} \longrightarrow \frac{\left(\frac{\phi^{i+1}-\phi^i}{\Delta x}\right) - \left(\frac{\phi^i - \phi^{i-1}}{\Delta x}\right)}{\Delta x} = \frac{\phi^{i+1} - 2\phi^i + \phi^{i-1}}{\Delta x^2}$$

So $H_{ij} = -\frac{1}{2\Delta x^2}\left(\delta_{i+1,j} - 2\delta_{ij} + \delta_{i-1,j}\right) + \frac{1}{2}x_i^2\delta_{ij}$

Numerical solution

What if $N_D = 4$? Then $\Delta x = 2$. Let's calculate the first entry

$$H_{00} = -\frac{1}{2(2^2)}(-2) + \frac{1}{2}(-4)^2 = \frac{1}{4} + 8 = 8.25$$

Filling out the rest, we have

$$H = \begin{pmatrix} 8.25 & -0.125 & 0 & 0 & 0 \\ -0.125 & 2.25 & -0.125 & 0 & 0 \\ 0 & -0.125 & 0.25 & -0.125 & 0 \\ 0 & 0 & -0.125 & 2.25 & 0.125 \\ 0 & 0 & 0 & -0.125 & 8.25 \end{pmatrix}$$

Numerical solution

What if $N_D = 8$?

The Hamiltonian matrix is sparse with a band structure!

$$H = \begin{pmatrix} 9 & -0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.5 & 5.5 & -0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -0.5 & 3 & -0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.5 & 1.5 & -0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -0.5 & 1 & -0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.5 & 1.5 & -0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -0.5 & 3 & -0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -0.5 & 5.5 & -0.5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.5 & 9 \end{pmatrix}$$

ARPACK

See: https://docs.scipy.org/doc/scipy/tutorial/arpack.html

ARPACK [1] is a Fortran package which provides routines for quickly finding a few eigenvalues/eigenvectors of large sparse matrices.

It requires only left-multiplication by the matrix in question performed through a **reverse-communication interface**. ARPACK is able to find eigenvalues and eigenvectors of any linear function mapping a vector to a vector (i.e. “matrix-free”)

> Rather than storing the matrix directly, when a matrix operation is required it returns control to the calling program with a flag indicating what operation is required

Two high-level interfaces: **scipy.sparse.linalg.eigs** (real or complex nonsymmetric square matrices) and **scipy.sparse.linalg.eigsh** (real-symmetric or complex-hermitian matrices)

[1] https://github.com/opencollab/arpack-ng

Functionality

ARPACK can solve eigenvalue problems of the form

$$A\boldsymbol{v} = \lambda\boldsymbol{v}$$

where A is an $n \times n$ matrix that has n linearly independent eigenvalues $\lambda_1, \ldots, \lambda_n$, rank them by their magnitude $|\lambda_1| > |\lambda_2| > \ldots > |\lambda_n|$, and the corresponding eigenvectors $v_1, \ldots, v_n$

The power of ARPACK is that it only computes a specified subset of eigenvalue/eigenvector pairs through the keyword which

which = 'LM' : Eigenvalues with largest magnitude $|\lambda|$

which = 'SM' : Eigenvalues with smallest magnitude $|\lambda|$

Power method: iteration 1

Let's see how we can solve just the largest magnitude eigenvalues with the *power method*

The eigenvectors are a basis, so we can write any vector $\boldsymbol{u}_0$ as a linear combination of them

$$\boldsymbol{u}_0 = c_1\boldsymbol{v}_1 + c_2\boldsymbol{v}_2 + \cdots + c_n\boldsymbol{v}_n,\ c_1 \neq 0$$

Multiply both sides by A

$$\begin{aligned} A\boldsymbol{u}_0 &= c_1 A\boldsymbol{v}_1 + c_2 A\boldsymbol{v}_2 + \cdots + c_n A\boldsymbol{v}_n \\ &= c_1\lambda_1\boldsymbol{v}_1 + c_2\lambda_2\boldsymbol{v}_2 + \cdots + c_n\lambda_n\boldsymbol{v}_n \\ &= c_1\lambda_1\left(\boldsymbol{v}_1 + \frac{c_2\lambda_2}{c_1\lambda_1}\boldsymbol{v}_2 + \cdots + \frac{c_n\lambda_n}{c_1\lambda_1}\boldsymbol{v}_n\right) \equiv c_1\lambda_1\boldsymbol{u}_1 \end{aligned}$$

where we have defined $\boldsymbol{u}_1 \equiv \boldsymbol{v}_1 + \frac{c_2\lambda_2}{c_1\lambda_1}\boldsymbol{v}_2 + \cdots + \frac{c_n\lambda_n}{c_1\lambda_1}\boldsymbol{v}_n$

Power method: iteration 2

Multiply $\boldsymbol{u}_1$ by A

$$A\boldsymbol{u}_1 = A\boldsymbol{v}_1 + \frac{c_2\lambda_2}{c_1\lambda_1}A\boldsymbol{v}_2 + \cdots + \frac{c_n\lambda_n}{c_1\lambda_1}A\boldsymbol{v}_n$$

$$= \lambda_1\boldsymbol{v}_1 + \frac{c_2\lambda_2^2}{c_1\lambda_1}\boldsymbol{v}_2 + \cdots + \frac{c_n\lambda_n^2}{c_1\lambda_1}\boldsymbol{v}_n$$

$$= \lambda_1\left(\boldsymbol{v}_1 + \frac{c_2\lambda_2^2}{c_1\lambda_1^2}\boldsymbol{v}_2 + \cdots + \frac{c_n\lambda_n^2}{c_1\lambda_1^2}\boldsymbol{v}_n\right) \equiv \lambda_1\boldsymbol{u}_2$$

where we have defined $\boldsymbol{u}_2 \equiv \boldsymbol{v}_1 + \frac{c_2\lambda_2^2}{c_1\lambda_1^2}\boldsymbol{v}_2 + \cdots + \frac{c_n\lambda_n^2}{c_1\lambda_1^2}\boldsymbol{v}_n$

Power method: iteration k

Multiply $\boldsymbol{u}_{k-1}$ by A

$$A\boldsymbol{u}_{k-1} = \lambda_1 \left(\boldsymbol{v}_1 + \frac{c_2\lambda_2^k}{c_1\lambda_1^k}\boldsymbol{v}_2 + \cdots + \frac{c_n\lambda_n^k}{c_1\lambda_1^k}\boldsymbol{v}_n \right) \equiv \lambda_1 \boldsymbol{u}_k$$

Eventually, as we increase k, the ratio $\left(\frac{\lambda_n}{\lambda_1}\right)^k \ll 1$ is sufficiently small so that $\boldsymbol{u}_k \approx \boldsymbol{u}_{k-1} \approx \boldsymbol{v}_1$

Shift-invert mode

ARPACK is better at finding eigenvalues with large magnitudes. So, using which = 'SM' may lead to slow execution time and/or anomalous results. A better approach is to use ***shift-invert mode***.

We can rewrite the eigenvalue problem by subtracting σv from both sides

$$(A - \sigma I)\boldsymbol{v} = (\lambda - \sigma)\boldsymbol{v}$$

Multiplying both sides the inverse matrix $(A - \sigma I)^{-1}$ and dividing both sides by the constant $\lambda - \sigma$, we find

$$(A - \sigma I)^{-1}\boldsymbol{v} = \left(\frac{1}{\lambda - \sigma}\right)\boldsymbol{v}$$

If we set $\sigma = 0$, we can solve for the largest eigenvalues of A^{-1}, i.e. $1/\lambda$, which correspond to the smallest eigenvalues of A, i.e. λ

Demo

https://jduarte.physics.ucsd.edu/phys142/lectures/10_EigenvalueProblemDemo.html