Problem 1:

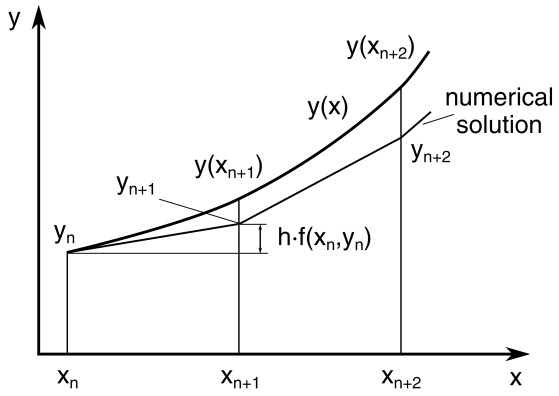
In this problem we want to solve second order differential equation $\boxed{m\frac{d^2x}{dt^2} + kx = a\frac{x^3}{6}}$ using :
1- Iteration Method (Euler's Method).
2- Python built in Method.
3- Map the problem to Schrodinger Equation.

Euler's Method

The Euler's method is a first-order numerical procedure for solving ordinary differential equations (ODE) with a given initial value. It uses the simple formula

$$y(x+h) = y(x) + hf(x,y)$$

to construct the tangent at the point x and obtain the value of $y(x+h)$, whose slope is $f(x,y) = \frac{dy}{dx}$.



In Euler's method, we can approximate the curve of the solution by the tangent in each interval (that is, by a sequence of short line segments), at steps of h.

Pseudo-code

Algorithm 1 Euler's Method

1: define $f(x, y)$
2: input x_0, y_0
3: input h, n
4: **for** j from 0 to (n-1) **do**
5: $\quad y_j + 1 = y_j + hf(x_j, y_j)$
6: $\quad x_{j+1} = x_j + h$
7: **end for**
8: plot result

Result

For $m, k, a = 1$ the result is :

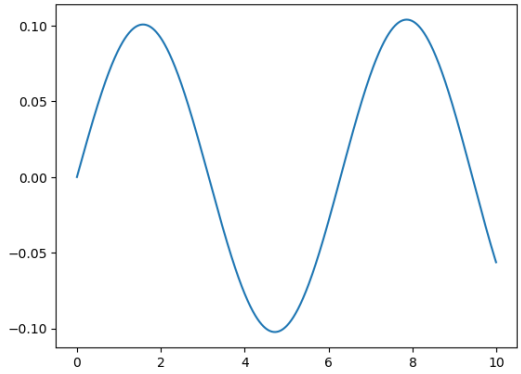


Figure 1: Euler's Method

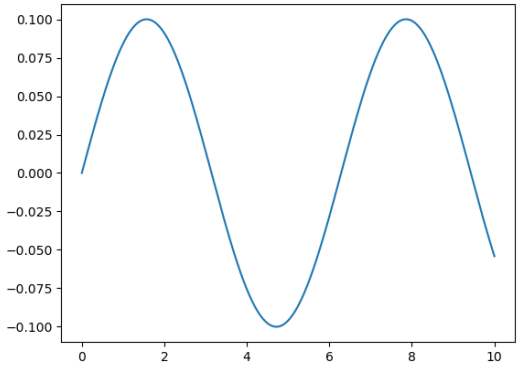


Figure 2: Python built-in

Map to schrodinger equation

We encounter a challenge in solving the equation using linear algebra and vectorization, similar to what we employed in homework 1, due to the presence of the term x^3. However, we can address this by considering a 2-boson problem with a Hamiltonian given by:

$$H = \frac{1}{2m}p^2 + \frac{k}{2}x^2 + \sum_i \alpha.[n_i.n_i]$$

Then, we can define the state $|\psi\rangle = |x\rangle \otimes |x\rangle$, allowing us to find wave functions associated with bosons using permutation operators, as demonstrated in homework 2. The primary issue with this approach lies in discretization. When we consider a small time step dt, the resulting Hamiltonian leads to very large matrices, exceeding the computational capabilities of standard computers.

Result

The result is :

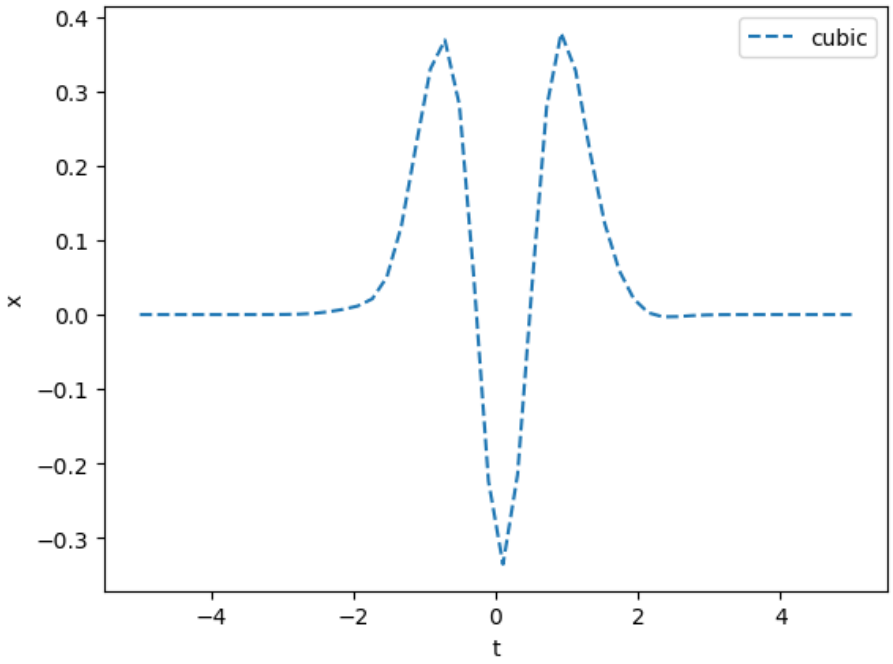


Figure 3: Mapping Method

As we see the result in not accurate.

Problem 2:

In this problem, we tackle the Schrödinger equation for a particle confined within a two-dimensional box. The methodology closely resembles Example 4 from Homework 1, with the primary deviation lying in the potential function. Here, we construct a potential matrix that remains zero across most elements, except for a select few diagonal terms where it takes on significantly large values.

Results

The constructed potential function is depicted in Figure 1, illustrating the imposing wall potential.

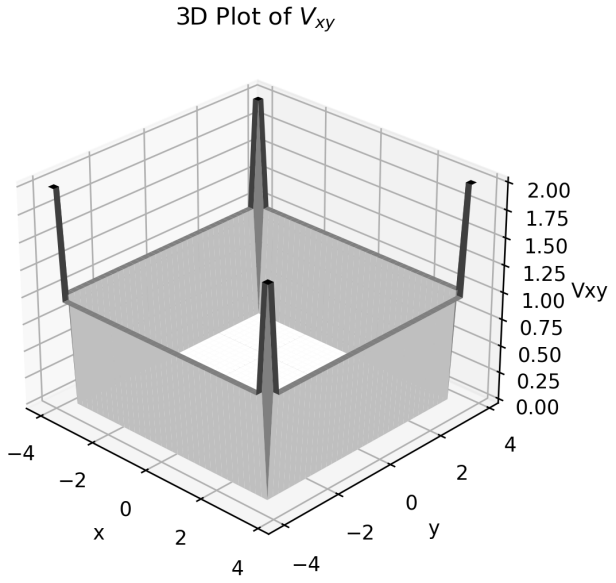


Figure 4: Wall Potential

Subsequently, the wave functions corresponding to the ground state and the first excited state are presented below:

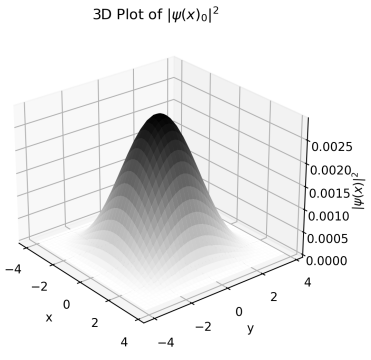


Figure 5: Ground State

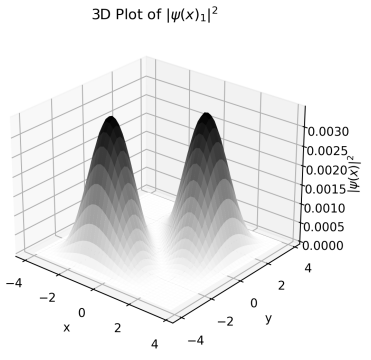


Figure 6: First Excited State

Problem 3:

In this problem, we utilize the Metropolis-Hastings algorithm and Monte Carlo simulation to sample provided distributions and compute integrals through averaging.

Metropolis Hastings algorithm

In statistics and statistical physics, the Metropolis-Hastings algorithm serves as a Markov chain Monte Carlo (MCMC) method designed for generating a sequence of random samples from a probability distribution when direct sampling proves challenging. This sequence aids in approximating the distribution, such as generating a histogram, or computing integrals, like expected values. Metropolis-Hastings and other MCMC algorithms find extensive use in sampling from multi-dimensional distributions, particularly in scenarios with high dimensionality. However, for single-dimensional distributions, alternative methods like adaptive rejection sampling typically offer direct independent samples from the distribution, without encountering the issue of autocorrelated samples inherent in MCMC methods.
The Metropolis–Hastings algorithm generates a sequence of sample values iteratively, gradually refining the distribution of values to closely approximate the desired distribution. Each sample value depends solely on the preceding one, forming a Markov chain. At each iteration, the algorithm proposes a candidate for the next sample value based on the current one. Subsequently, the candidate is either accepted, incorporating its value in the next iteration, or rejected, discarding the candidate value and retaining the current one for the subsequent iteration. The acceptance proba-

bility is determined by comparing the values of the function $f(x)$ evaluated at both the current and candidate sample values relative to the desired distribution.

Pseudo-code

Algorithm 2 The Metropolis-Hastings algorithm

Initialize $x^{(0)}$
for i = 0 to N-1 **do**
    Sample $u -> U_{[0,1]}$
    Sample $x^* -> q(x^*|x^{(i)})$
    If $u < A(x^{(i)}, x^*) = min\{1, \frac{p(x^*)q(x^{(i)}|x^*)}{p(x^{(i)})q(x^*|x^{(i)}}$
    $x^{i+1} = x^*$
    else
    $x^{i+1} = x^(i)$
end for

Result

Initially, we computed the normal distribution to validate the accuracy of our code. The comparison between the calculated distribution and the expected normal distribution is depicted below, affirming the accuracy of our results:

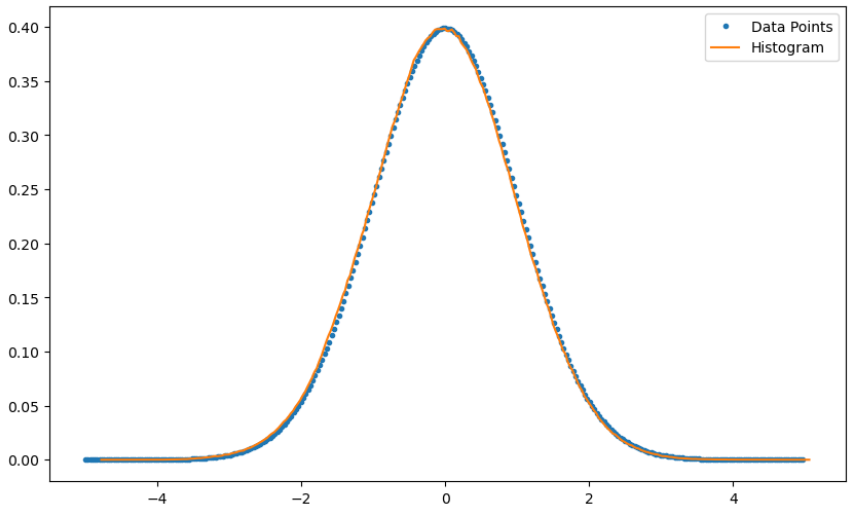


Figure 7: Comparison between calculated and normal distributions

Next we've computed the provided integrals, yet encountered a sign problem in two cases. This issue arises when the function $f(x)$ is not strictly positive.
In the Metropolis-Hastings algorithm, when sampling from a distribution with a non-positive integrand function $f(x)$, the proposed samples may have a negative contribution, leading to a situation where accepted samples do not contribute positively to the integral. Consequently, the Markov chain may struggle to explore the state space effectively, resulting in unreliable estimates and divergence from the expected integral value.
This sign problem underscores the importance of ensuring positivity of the integrand function within the context of the Metropolis-Hastings algorithm. Strategies to mitigate this issue may involve transforming the problem or modifying the proposal distribution to avoid negative contributions during the sampling process. Additionally, careful consideration of the behavior of $f(x)$ and its impact on the sampling procedure can aid in identifying and addressing sign problems effectively.
As we discussed in class if the numbers of sampling be n we have to n^2 times samplings when we have sign problem.
1. For $f(x) = x$ and $w(x) = e^{-|x|}$, we obtain $0.0079 \approx 0$, consistent with other numerical methods.

2. With $f(x) = x^2$ and $w(x) = \frac{1}{1+x^2}$, we observe different values because it is a divergent integral but in [-12,12] its approximately 7.5. This divergence aligns with results from other numerical methods.

3. For $f(x) = x^2$ and $w(x) = \cos(x)$, the integral approximates to zero over the interval $[-6\pi, 6\pi]$. Here, we encounter a sign problem as discussed earlier. Additionally, the function is periodic.

4. In the case of $f(x) = x^2$ and $w(x) = \cos(x)e^{-\frac{x^2}{2}}$, a sign problem arises.

5. Considering $f(x) = \cos(10x)$ and $w(x) = e^{-\frac{x^2}{2}}$, the integral approximates to zero, consistent with outcomes obtained from other numerical methods.

Problem 4:

In this part we are going to implement Lanczos method for diagonalization of a matrix.

Krylov Subspaces and Arnoldi Iteration

Key idea: Matrix-vector multiplication Ab is fast, especially if A is sparse. If we start with A and b, we can quickly compute each of the vectors $b, Ab, ..., A^{n-l}b$. (Never compute $A^2 or A^3$ Only compute vectors.) The combinations of those n vectors make up the nth Krylov subspace. We look inside this subspace K_n for a close approximation to the desired solution x.
$H_k = Q_k^T A Q_k$ is the projection of A onto the Krylov space, using the basis of q's.
The Arnoldi process to find H_k is one of the great algorithms of numerical linear algebra. It is numerically stable and the q's are orthonormal.

Symmetric Matrices : Arnoldi Becomes Lanczos

Suppose our matrix is symmetric:$A = S$. In this important case, two extra facts are true
1. $H_k = Q_K^T S Q_k$ is also symmetric. Its transpose is clearly H_k.
2. H_k is tridiagonal: only one diagonal above because only one diagonal below.
A tridiagonal matrix H gives a major saving in cost-the Arnoldi iteration needs only one orthogonalization step. The other orthogonalities are built in because H is symmetric Hessenberg (so it is tridiagonal)
Here is Lanczos with simpler lettersa_1 to a_k(on the main diagonal) and b_1 to b_{k-1} (on the diagonals above and below). The a's and b's in T replace the h's of Arnoldi's matrix H. Writing T for tridiagonal instead of H for Hessenberg, here are the key

Algorithm 3 The Metropolis-Hastings algorithm

$q_0 = 0, q_1 = b/||b||$
for $K = 1, 2, 3, \ldots$ **do**
    $v = Sq_k$ start with new v
    $a_k = q_k^T v$ Diagonal entry in T is a_k
    $v = v - b_{k-1}q_{k-1} - a_k q_k$ Orthogonal to earlier q's
    $b_k = ||v||$ OFF-diagonal entry in T is b_k
    $q_{k+1} = v/b_k$ Next basis vector
end for

facts for Lanczos. They are simply copied from Arnoldi

$$T_k = Q_k^T S Q_k, \quad and \quad SQ_k = Q_{k+1} T_{k+1,k}$$

"The eigenvalues of Tk (fast to compute) approximate the eigenvalues of S." If only that were exactly and always true ! The problem comes from non-orthogonal q's

when exact Lanczos iterations would guarantee orthogonal q's. Lanczos is valuable. But special care is needed to keep all the q's orthogonal in practice-which was true also of Gram-Schmidt.

A:

In this section, we developed a code utilizing the Lanczos method to find the lowest eigenvalue and eigenvector of $M = A + A^{\dagger}$, where A is a random $n \times n$ matrix. We verified the accuracy of our code for $n = 2000$ by comparing it with Python's diagonalization method. Additionally, we reported and compared their CPU times. To enhance accuracy, we implemented both regular Lanczos and modified Lanczos methods, the latter re-orthogonalizing all vectors every two steps.

Results

The results obtained from **Lanczos**, **Modified Lanczos**, and **numpy.eig()** methods are displayed below:

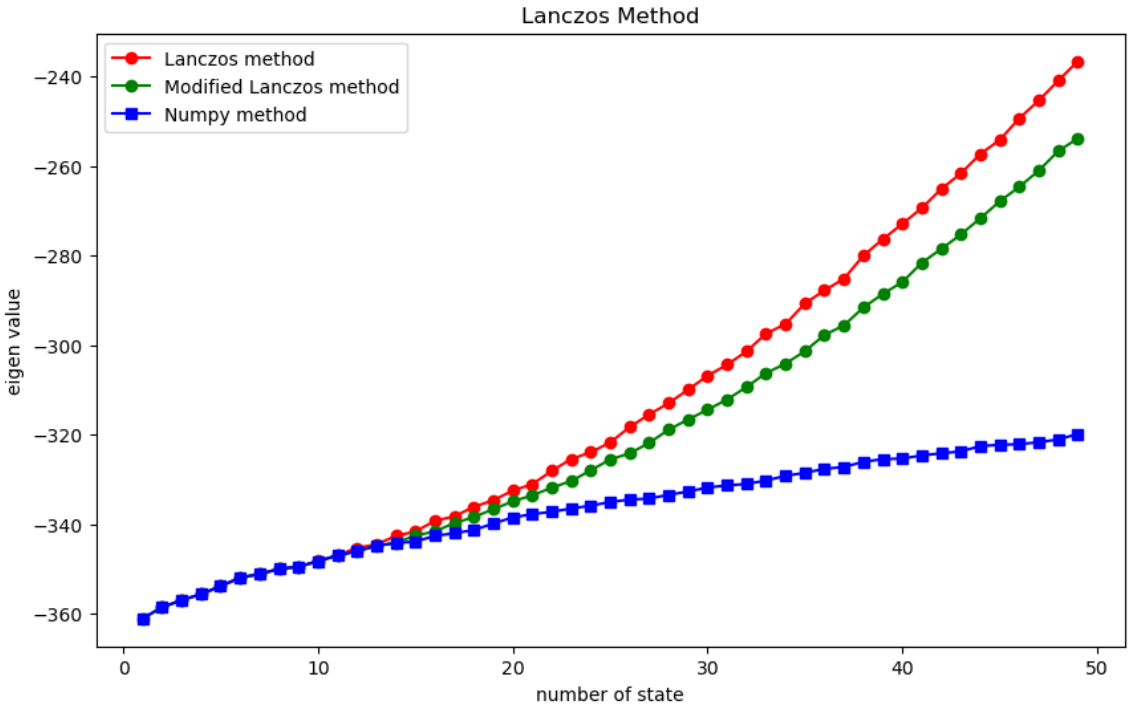


Figure 8: Comparison of results from different methods

As observed, the methods provide fairly accurate results for the first eigenvalues.

For time comparison:

Method	CPU Time (seconds)
Lanczos	2.4
Modified Lanczos	2.7
Numpy.eig()	1

B:

Next, we considered $M = 2A_1 \otimes B_1 + 3A_2 \otimes B_2 + h.c.$, where $\otimes$ denotes the Kronecker tensor product, A_1 and A_2 are $n \times n$ matrices ($n = 50$), and B_1 and B_2 are $m \times m$ matrices ($m = 100$). We constructed M explicitly and utilized the methods from part A to obtain its lowest eigenvalues and eigenvectors.

Results

The results for this section are depicted below:

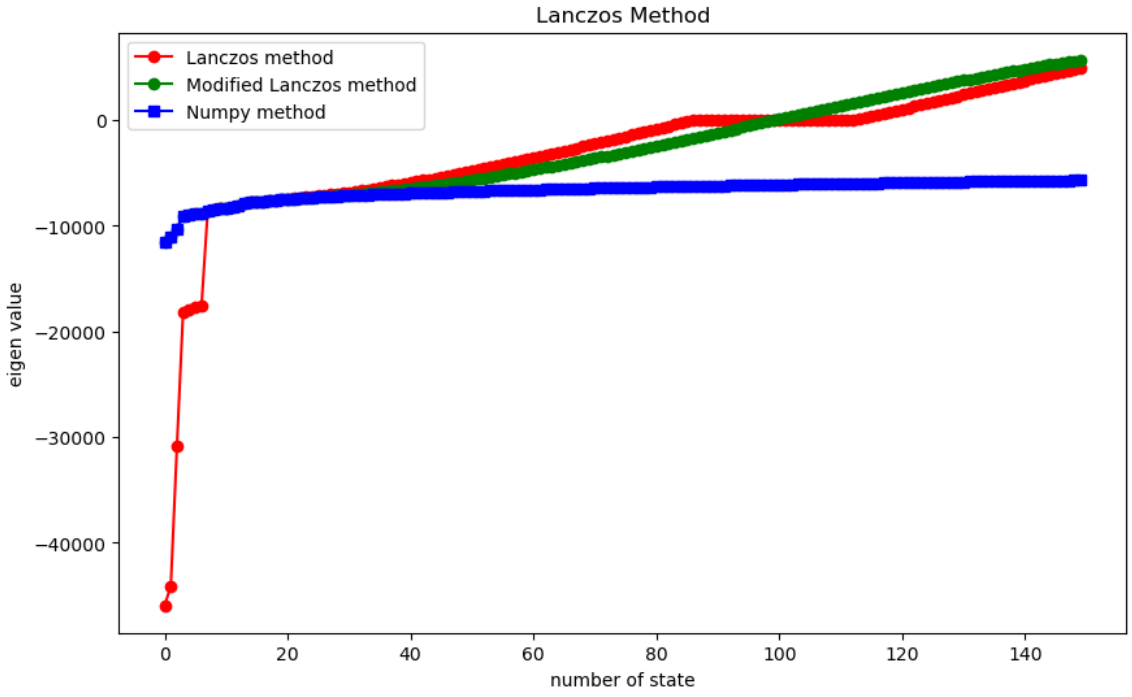


Figure 9: Comparison of results from different methods

Remarkably, the modified Lanczos method demonstrates superior accuracy in this case, highlighting the importance of re-orthogonalization in large matrices.

For time comparison:

Method	**CPU Time (seconds)**
Lanczos	16.7
Modified Lanczos	16.8
Numpy.eig()	13

C:

In this part, we developed a code based on the Lanczos method to obtain the lowest eigenvalue and eigenvectors **without explicitly constructing matrix** M, as explained in class. We then compared the results of parts B and C.

Results

The results show that the eigenvalue obtained using the Lanczos method without explicitly constructing M is consistent with the method used in part B but significantly faster.

Method	**CPU Time (seconds)**
New Lanczos	0.5
Lanczos	16.7
Modified Lanczos	16.8
Numpy.eig()	13