Computational Physics
Set 4

Ali Ashtari
400100038

March 19, 2024

Problem 1: A non-linear differential equation

In this problem, we are dealing with a non-linear differential equation:

$$m\frac{d^2x}{dt^2} + kx = \frac{a}{6}x^3 \tag{1}$$

For numerical solutions, I set m= 5, k=2, a=4. Equation (1) is a second order non-linear differential equation and to be solved, must have two initial values. The initial condition I assume is $x_0 = 1.0$ and $\frac{dx}{dt}(0) = 0.0$. First I use "scipy's odeint method" to solve this equation. Then I use iteration method, and after that I use "Runge Kutta algorithm" to solve this equation numerically. Note that this is a second order differential equation and can be brought into two coupled first order differential equation:

$$\frac{dx_1}{dt} = x_2 \tag{2}$$

$$\frac{dx_2}{dt} = \frac{a}{6m}x_1^3 - \frac{k}{m}x_1 \tag{3}$$

For iteration method, I use the regular matrix method , constructing differential operator $D^{(2)}$ and adding two rows (initial conditions) to make it invertible and computing the inverse. Then I solve the homogeneous equation; then putting the resulting x in the in-homogeneous part (right hand side) and solve it again and repeat this procedure. For Runge-Kutta algorithm, I use the equations in figure 1.

$$y_{n+1} = y_n + h\left(\frac{k_{n1} + 2k_{n2} + 2k_{n3} + k_{n4}}{6}\right),$$

$$\begin{aligned}
k_{n1} &= f(t_n, y_n),\\
k_{n2} &= f(t_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}hk_{n1}),\\
k_{n3} &= f(t_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}hk_{n2}),\\
k_{n4} &= f(t_n + h, y_n + hk_{n3}).
\end{aligned}$$

Figure 1: Runge-Kutta algorithm's equations

The resulting numerical solutions are shown in figure 2.

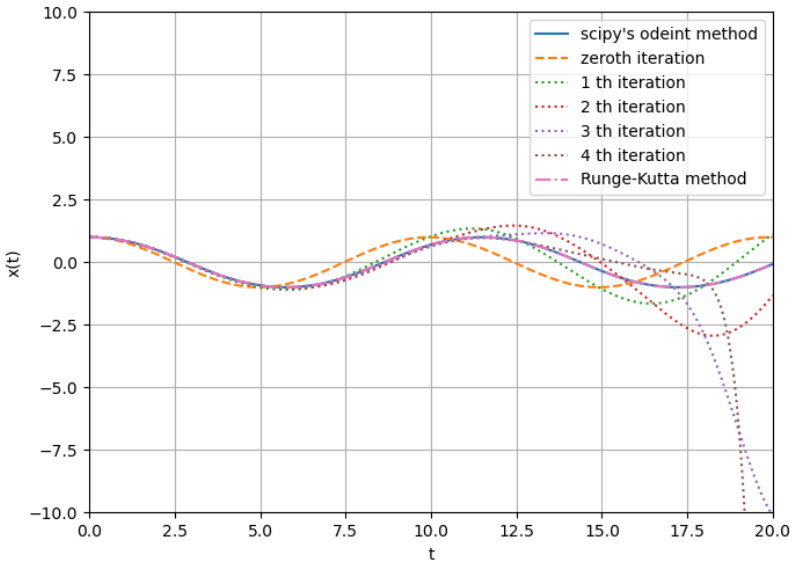

Figure 2: Numerical solutions of equation (1)

Note that Runge-Kutta and scipy methods are compatible with each other. Note the relative instability in iteration method. If I've chosen 'a' smaller, this solution would be similar to the other two, but if the non-linear term has a big contribution to the equation, it cannot be solved peturbatively (with iteration), because the solutions would not converge and this solution suffers from numerical instability.

Problem 2: 2-dimensional infinite potential well

In this problem we have to obtain energy eigenvalues and eigenvectors of a particle in 2-dimensional infinite potential well. The Schrödinger equation takes the form:

$$[-\frac{\hbar^2}{2m}(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2})+V(x,y)]\psi(x,y)=E\psi(x,y) \tag{4}$$

We construct our operators as:

$$\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} = D_x^{(2)} \otimes \mathbb{1}_y + \mathbb{1}_x \otimes D_y^{(2)} \tag{5}$$

Where $D^{(2)}$ s are second derivative operators. Assuming $\hbar = 1, m = 1$, by constructing these operators, and also for the 2D infinite potential well (shown in figure 3):

$$V(x,y) = \begin{cases} 0 & \text{if } -5 \leq x \leq 5 \text{ and } -5 \leq y \leq 5 \\ \infty & \text{otherwise} \end{cases} \tag{6}$$

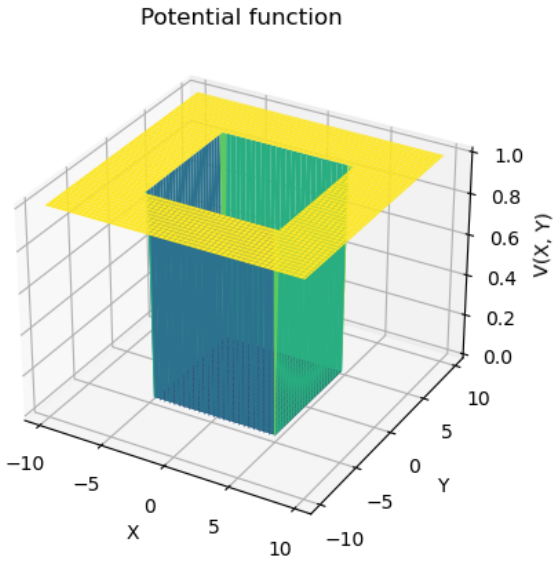

Figure 3: Infinite square well potential function

By diagonalizing the hamiltonian, the spectrum of the energy would be as figure 4.

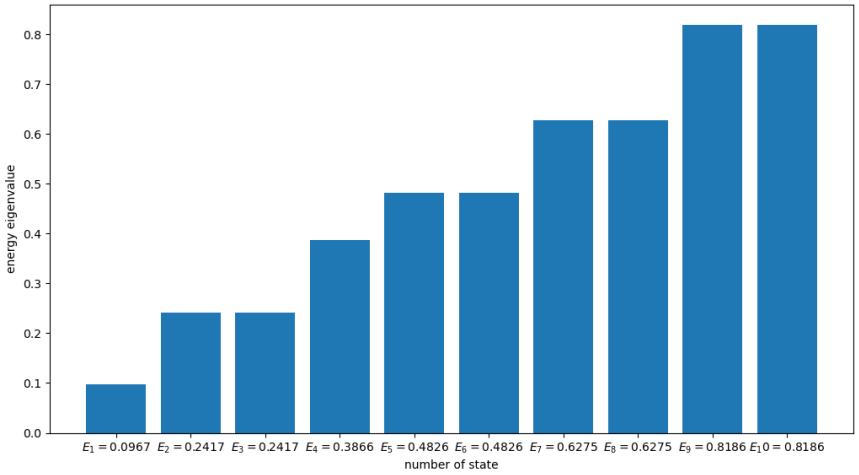

Figure 4: 10 lowest energy eigenvalues

The obtained energy eigenvalues are:
[0.09673972, 0.2416658, 0.24166581, 0.3865919, 0.48259838, 0.48259839, 0.62752447, 0.62752447, 0.81862348, 0.81862349]

The ground state and first excited state wave-functions are shown in figure 5 and 6.

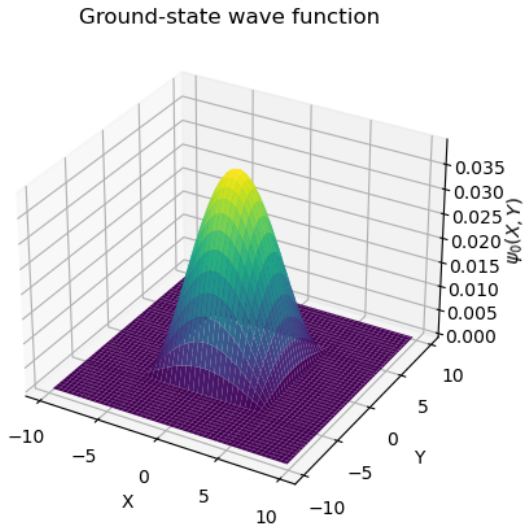

Figure 5: Ground state wave-function of 2D infinite potential well

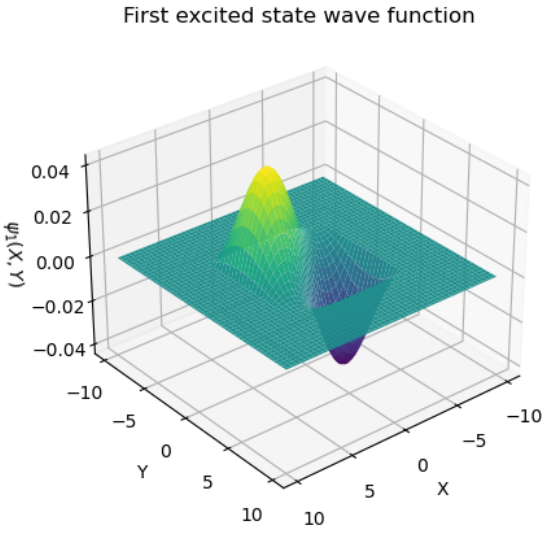

Figure 6: First excited state wave-function of 2D infinite potential well

Problem 3: Monte-Carlo integration

In this problem we have to find the expectation values of f(x) functions, using probability distribution functions (PDF) w(x). In fact w(x) are not PDF because they are not normalized; so we have to do it by using normal distribution function p(x).

$$\langle f(x)\rangle = \frac{\int f(x)w(x)dx}{\int w(x)dx} \tag{7}$$

We can do the integration of denominator by this method:

$$\int w(x)dx = \int \tilde{w}(x)p(x)dx \ \ where\ \tilde{w} = \frac{w(x)}{p(x)} \tag{8}$$

The results obtained from Monte-Carlo integration with 10^7 sample point ($\frac{1}{5}$ of them thrown out as warm up points) and Riemann sum with equal intervals. Note that by running the code a few times, the answer may differ too much (in there is numerical instability) or a little (initial point and length of each step in Metropolis algorithm is random).

A)

$$f(x) = x \ \ with\ w(x) = \exp(-|x|) \tag{9}$$

Monte-Carlo : $< f1(x) >= -1.8820125436432616e-05$
Riemann Sum : $< f1(x) >= -5.997405666119425e-16$
These are close enough. It should be zero and both ways are close to it. For 10^6 samplings the obtained value is -0.005493614439089366. The convergence is relatively fast in this case.

B)

$$f(x) = x^2 \ \ with\ w(x) = \frac{1}{1+x^2} \tag{10}$$

In this case, note that the numerator of equation 7 would diverge! So I restrict the interval of integration to [-10, 10]. Also in this case normalizing w(x) by normal distribution (according to equation (7)) increase the error too much! so I use the Riemann sum to normalize w, and then perform the Monte-Carlo integration. The results are as follows:
Monte-Carlo : $< f2(x) >= 4.644188228339423$
Riemann Sum : $< f2(x) >= 5.829203568906976$

The results are close enough; the relative error is about 20 percents. Not bad! For 10^6 samplings the answer is 9.303543240058215, in this case the convergence is relatively fast.

C)

$$f(x) = x^2 \ with \ w(x) = \cos(x) \tag{11}$$

This case is totally different from the above cases, here we have the famous "sign problem", our probability became negative in some places! First to make this problem more handle-able, I choose to call $|\cos(x)$ the PDF, write the integral as follows:

$$\langle f(x)\rangle = \frac{\int f(x)\cos(x)dx}{\int \cos(x)dx} = \frac{\int f(x)\frac{\cos(x)}{|\cos(x)|}|\cos(x)|dx}{\int \frac{\cos(x)}{|\cos(x)|}|\cos(x)|dx} \tag{12}$$

Dividing numerator and denominator to $\int |\cos(x)|dx$, we would get:

$$\langle f(x)\rangle = \frac{\langle f(x)sgn(\cos(x))\rangle}{\langle sgn(\cos(x)\rangle} \tag{13}$$

Assuming the interval of integration $[-10\pi, 15\pi]$, the results would be:
Monte-Carlo : $< f3(x) >= 1001.5947866710343$
Riemann Sum : $< f3(x) >= 3780.1657231154386$
The answer is too far! But this was expected because of the sign problem. This problem causes numerical instability and the answer cannot be trusted when the sign problem is present. For 10^6 samplings the answer is -793.8880569520712 and for 10^8 samplings the answer is 133.06372907749778. There is no sign of convergence because of the infamous sign problem.

D)

$$f(x) = x^2 \ with \ w(x) = \cos(x)\exp(\frac{-x^2}{2}) \tag{14}$$

Again we have sign problem and we don't expect to obtain the right answer. The general case of equation (13) can be written as:

$$\langle f(x)\rangle = \frac{\langle f(x)sgn(w(x))\rangle}{\langle sgn(w(x))\rangle} \tag{15}$$

Now we proceed as previous part (except that the integration interval is from $-\infty$ to ∞) and use equation (15). The obtained results are:

Monte-Carlo : $< f4(x) >= -2.3988139381904943$
Riemann Sum : $< f4(x) >= 5.260289108843181e-18$
Again the sign problem cause numerical instability and the answer is far from the Riemann sum method. But because of exponential fall of w(x), in this case the answer is less catastrophic than part c.

E)

$$f(x) = \cos(10x) \ with \ w(x) = \exp(\frac{-x^2}{2}) \tag{16}$$

In this case there is no sign problem and the obtained valued must be valid. The obtained values are:
Monte-Carlo : $< f5(x) >= 0.0001545388710736985$
Riemann Sum : $< f5(x) >= -3.543319241585075e-17$
Without the sin problem, the value obtained by Monte-Carlo method is valid and by increasing number of sample points, we get closer to real answer. Here the real answer is 0, and both the answers are close enough to it.

Problem 4: Lanczos algorithms

In this problem, I use Lanczos algorithm to diagonalize matrices and obtain a few of lowest eigenvalues and eigenvectors. Note that I've shown the eigenvalues here, but the eigenvectors are too large for this purpose! The eigenvectors can be seen by ruuning the code, and for easier check, I've calculated the overlap of the states by different method; if the overlap is close to +1 or -1, the eigenvectors are equeivalent (the third one sometimes is less presice, you can increase the number of itterations for more precise results, with the cost of time.). The Lanczos algorithm an be briefly stated by the following formulas:

$$Random \ |v_0\rangle \tag{17}$$

$$a_0 = \langle v_0|H|v_0\rangle \tag{18}$$

$$b_1|v_1\rangle = H|v_0\rangle - a_0|v_0\rangle \tag{19}$$

$$b_1 = ||H|v_0\rangle - a_0|v_0\rangle|| \tag{20}$$

$$a_1 = \langle v_1|H|v_1\rangle \tag{21}$$

$$...$$

$$|w_j\rangle = b_j|v_j\rangle = H|v_{j-1}\rangle - |v_{j-1}\rangle\langle v_{j-1}|H|v_{j-1}\rangle - |v_{j-2}\rangle\langle v_{j-2}|H|v_{j-1}\rangle \tag{22}$$

$$b_j = ||w_j|| \tag{23}$$

$$|v_j\rangle = \frac{|w_j\rangle}{b_j} \tag{24}$$

A)

In this part, I've used two algorithms to obtain 3 lowest eigenvalues and corresponding eigenvectors. In first one, I've used the numpy's eigh method, and for one run, These values obtained (we are dealing with random matrices and in each run, values would change):
[-62.99041674, -62.65035174, -62.22502963]
Run time = 0.5168230533599854 s

Then, I use Lanczos algorithm with m=60 iterations (more iteration, more precision, more time elapsed) and 3 lowest eigenvalues are as follows:
[-62.99032595, -62.64384571, -62.10408386]
Run time = 0.04687857627868652 s

It can be observed that the run time for the Lanczos algorithm is much lesser, and the eigenvalues are very close the numpy's method (the smallest eigenvalues has the least error, and by moving to larger eigenvalues, the error would be larger too). I could increase the number of iterations and obtain more precision, but with cost of more time.

B) In this part, we are dealing with tensor product structured matrices.

$$M = 2A_1 \otimes B_1 + 3A_2 \otimes B_2 + h.c. \tag{25}$$

In which "h.c." means hermitian conjugate of the terms must be added too. A_1, A_2 are 50×50 random matrices and B_1, B_2 are 100×100 random matrices. M is a hermitian random matrix.

By explicitly constructing M and using numpy's eigh method, the 5 lowest eigenvalues are obtained as:
[-972.11594919, -953.60160812, -939.41714083, -933.40198831, -915.00015101]
Run time = 11.299247026443481 s

Using Lanczos method with m=100 iterations to diagonalize it, the 5 lowest eigenvalues are:
[-972.11594919, -953.60160812, -939.41714083, -933.40198831, -915.00014543]
Run time = 1.236741065979004 s

The eigenvalues in both methods are very close to each other (in the smallest eigenvalues, the value obtained by two methods are equal to the shown precision), but Lanczos algorithm do it in much smaller time!

C)

In this part, we have to diagonalize the M matrix from the last part, but without explicitly constructing M! This method uses much less RAM and is much faster than the standard Lanczos method (and of course incredibly faster than numpy's eigh).

Without doing tensor products explicitly, we would have:

$$|g\rangle = \sum_{i,j} C_{ij} |u_i\rangle \otimes |v_j\rangle \tag{26}$$

$$H|g\rangle = \sum_{\alpha} g_\alpha B_\alpha C A_\alpha^T \tag{27}$$

We can regard this C same as the $|v\rangle$ in the standard Lanczos method and after reshaping, doing the same operations as before.

The eigenvalues obtained by this algorithm (with 100 iterations):
[-972.11594919, -953.60160812, -939.41714083, -933.40198831, -915.00014418]
Run time = 0.031247377395629883 s

Eigenvalues are very close (mostly identical!) to numpy's method and standard Lanczos method. But the run time is much less!! Its about 360 times faster than numpy's method! And about 40 times faster than standard Lanczos method!!! It's by far more efficient algorithm to obtain a few of eigenvalue and eigenvectors of a hermitian matrix.