Problem Set 1

Submit your solution via NTULearn, with one Python file per problem. Problems labeled like this—(5* marks)—are optional for undergraduates, and can be attempted for a bonus of 1/5 the indicated marks; for graduate students, these problems are compulsory and receive full marks. Your code should follow good programming style, with clear comments. Plots should be labeled clearly.

0. GAUSSIAN ELIMINATION

In this problem, you will write and test an implementation of the *Gaussian elimination* algorithm, which solves linear systems of equations of the form

$$A\vec{x} = \vec{b}, \tag{0}$$

where A is a square matrix, and $\vec{x}$ and $\vec{b}$ are vectors. The goal is to find $\vec{x}$ given A and $\vec{b}$. See ⟨`http://spike.spms.ntu.edu.sg/wiki/Gaussian_elimination`⟩ for a description of the algorithm. We will split the implementation into several pieces.

(a) (2 marks) Write a function to perform the row reduction phase of the algorithm:

`def row_reduce(A, b, n):`	
Inputs	
`A`	A 2D array specifying a square matrix A.
`b`	A 1D array specifying a vector $\vec{b}$.
`n`	The pivot row index (an integer).

This function has no return values, and directly modifies the contents of the `A` and `b` arrays, in order to perform "*row reduction*". This means modifying `A` and `b` so that all the elements underneath `A[n,n]` become zero. The procedure is as follows:

$$\text{For each } m > n, \begin{cases} \text{(i)} & b_m \to b_m - \left(\frac{A_{mn}}{A_{nn}}\right) b_n \\ \text{(ii)} & A_{mk} \to A_{mk} - \left(\frac{A_{mn}}{A_{nn}}\right) A_{nk}, \end{cases}$$

where k runs over all the columns of A.

You should test this code. For example, try the example given in the course notes:

```
A = array([[1., 2., 3.],
           [3., 2., 2.],
           [2., 6., 2.]])
b = array([3., 4., 4.])

row_reduce(A, b, 0)
print(A)
print(b)
```

This should print:

```
[[ 1.  2.  3.]
 [ 0. -4. -7.]
 [ 0.  2. -4.]]
[ 3. -5. -2.]
```

(b) (2 marks) Write a function to perform pivoting:

`def pivot(A, b, n):`	
Inputs	
`A`	A 2D array specifying a square matrix A.
`b`	A 1D array specifying a vector $\vec{b}$.
`n`	The pivot row index (an integer).

This function has no return values, and directly modifies the contents of the `A` and `b` arrays. It performs "*pivoting*", which consists of the following steps:

(i) Look through rows $m \geq n$, and find the row with the largest value of $|A_{mn}|$.

(ii) If $m \neq n$ was the row round in (i), then swap rows m and n in the matrix A; also, swap elements m and n in the vector $\vec{b}$.

Again, be sure to test your code for correctness.

(c) (3 marks) Write a function to perform Gaussian elimination:

`def gauss_eliminate(A, b):`	
Inputs	
`A`	A 2D array specifying a square matrix A. Will not be altered.
`b`	A 1D array specifying a vector $\vec{b}$. Will not be altered.
Return value	
`x`	A 1D array containing the solution to $A\vec{x} = \vec{b}$.

This should use the functions written in (a) and (b) as subroutines, as well as code for performing the back-substitution part of the Gaussian elimination algorithm.

Note that the `gauss_eliminate` function is *not* supposed to alter the contents of the input arrays `A` and `b`, unlike the functions written in (a) and (b). Use `copy` appropriately to accomplish this.

(d) (3 marks) Write a function, `gauss_eliminate_profile()`, to measure and report the performance of the Gaussian elimination algorithm. This function should measure the time taken to run Gaussian elimination, t_N, versus the problem size N. You are free to choose an approprate range for N, such as $10 \lesssim N \lesssim 500$. Performance times can be measured using Python's `time.perf_counter` function.

The function should then show a "log-log" plot of t_N versus N (e.g., using the `plt.loglog` function). Within the same figure, plot the performance graphs for two different solvers: (i) the `gauss_elimination` solver you wrote in part (c), and (ii) the `scipy.linalg.solve` function, which is Scipy's own implementation of Gaussian elimination.

Furthermore, for each graph, compute and show the trend-line for the least-squares fit

$$\log(t) \approx p \log(N) + q.$$

(To get cleaner results, you might want to fit to a subset of the data points; if so, indicate this choice clearly in code comments.) Make separate fits for the `gauss_eliminate` and `scipy.linalg.solve` results. You can use `polyfit` for the linear fitting. Label the two graphs clearly, and show the fitted value of p in the trend-line labels.
Discuss in code comments: What is the significance of the values of p and q found?

(e) (2 marks) Modify your Gaussian elimination code to detect the case where A is non-invertible, and generate an error using Python's `raise` statement. Hint: you should *not* do this by calculating $\det(A)$; why?

(f) (4* marks) Modify your Gaussian elimination code so that it handles the case where x is a 2D array, representing an $N \times M$ matrix. The code should still work as usual if x is a 1D array representing a vector.

Then, write a function `gauss_eliminate_profile2()`, to profile the performance of matrix inversion, comparing your Gaussian elimination code to `scipy.linalg.inv`.

1. THE HARPER MODEL

The *Harper model* is a theoretical quantum system consisting of a particle existing along a discrete one-dimensional "chain". The chain consists of a set of discrete points labeled by $n = 0, 1, \ldots, N-1$. A quantum state is described by a complex vector,

$$\psi = \begin{bmatrix} \psi_0 \\ \vdots \\ \psi_{N-1} \end{bmatrix}, \quad \text{where} \quad \sum_{n=0}^{N-1} |\psi_n|^2 = 1.$$

For each n, the complex number ψ_n gives the "quantum wavefunction" at position n; the probability to observe the particle at that position is $|\psi_n|^2$.

An "energy state" is a quantum state that satisfies the eigenvalue problem

$$H\psi = E\psi,$$

where $E \in \mathbb{R}$ is the energy and H is an $N \times N$ matrix called the "Hamiltonian". The Hamiltonian has the form

$$H = \begin{bmatrix} V_0 & 1 & & \\ 1 & V_1 & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & V_{N-1} \end{bmatrix}, \quad \text{where} \quad V_n = W\cos(2\pi\alpha n + \phi).$$

The diagonal entries V_n represent the potential along the 1D chain. This potential varies sinusoidally, with amplitude W and phase parameters α and ϕ. If α is irrational, the potential function is "aperiodic" (i.e., it does not repeat with n).

(a) (2 marks) Write the following function:

`def harper_hamiltonian(N, W, phi, alpha):`	
Inputs	
`N`	The Hamiltonian size (an integer).
`W`	The amplitude of the potential function (a number).
`phi`	The parameter ϕ in the potential function (a number).
`alpha`	The parameter α in the potential function (a number).
Return value	
`H`	The Harper model's Hamiltonian matrix (a 2D array).

(b) (2 marks) Write the following function:

`def harper_levels_plot(N=199, W=2., phi=0.):`	
Inputs	
`N`	The Hamiltonian size (an integer).
`W`	The amplitude of the potential function (a number).
`phi`	The parameter ϕ in the potential function (a number).

This function should plot the "energy-level diagram" of E versus α: i.e., for each value of α on the horizontal axis, plot the discrete energies E (the eigenvalues of H) on the vertical.

(c) (2 marks) Write `harper_wavefunction_demo(N=199, alpha=1.618034, phi=0.)`, which plots the probability density $|\psi_n|^2$ for the *ground state* (the energy state with lowest energy), versus position n. Do this for several choices of W, in the range $1.2 \lesssim W \lesssim 2.5$. Label all plots clearly; you may use subplots for clarity.

Discuss in code comments: How does the behavior of $|\psi_n|^2$ change with W?

(d) (2 marks) The "inverse participation ratio" (IPR) of a wavefunction is the quantity

$$\mathrm{IPR}[\psi] = \sum_n |\psi_n|^4.$$

It is a crude measure of whether a wavefunction is "localized" (i.e., concentrated near a few points) or "extended" (i.e., spread out over the whole chain). Write a function `harper_ipr_demo(N=199, alpha=1.618034, phi=0.)`, which plots the mean IPR of the Harper model's energy states, versus W.

Discuss in code comments: Is the IPR large or small for localized states? Estimate the critical W where localization begins to occur.

(e) (6* marks) We now consider the ϕ parameter. Write a function `harper_phi_demo()`, which plots the energy levels versus $\phi \in [-\pi, \pi]$. Choose appropriate values for the other parameters, but let α be irrational. You should find that most of the energies cluster into "bands". However, there exist some states lying outside the bands. In a separate subplot or figure, plot the probability density for one or more of these "out-of-band" states. In the energy level plot, use markers to indicate the states you are plotting.

Discuss in code comments: How do the in-band and out-of-band states differ? How does this phenomenon vary with α?