CS 369 Assignment 1 2017

Due: Friday, March 24 2017, 6:00 p.m.

Marked out of 24 points. 6% of the final grade.

Before you make a start, read over the whole assignment including the requirements section at the end.

Problem 0: Working with Jupyter notebooks [*5 points*].

All your assignments in this course will be handed in as Jupyter notebooks. Jupyter notebooks (formerly known as iPython notebooks) allow you to gather you code, output, discussion and analysis all in one place. They are a valuable tool for creating reproducible science.
The basic organising principle of a notebook is that it is a series of cells which contain either code (we'll use Python) or formatted text (markdown). The output of a code cell is written directly below that cell.
The purpose of this section is just to familiarise yourself with Jupyter notebooks. Jupyter notebooks can be accessed on the lab computers via Anaconda (search in the Start menu) or install Anaconda on your own machine (install instructions at `http://jupyter.org/install.html`).
There is many basic intros to Jupyter you can refer to, for example `http://jupyter-notebook.readthedocs.io/en/latest/#community-documentation`.
Demonstrate that you have some mastery of notebooks by starting your submission with the following:

a. A markdown cell containing some formatted to text with two sizes of headings, a list and the following two sentences including mathematical notation:
It is easy to type maths in markdown, it is like Latex. We can write inline maths like $f(x) = x^2$ or displayed maths such as

$$\mathbf{a} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}.$$

b. An image embedded so that the image is included at distribution (see how at `http://stackoverflow.com/questions/26068316/embedding-image-in-ipython-notebook-for-distribution`)

c. A plot of the function $f(x) = x^2 - 5$ for $-10 < x < 10$ using matplotlib

Problem 1: Root finding [*7 points*].

a. Write down the formula used get the approximation x_i given x_{i-1} to the roots of $f(x)$ and $g(x)$ (given below) using Newton's root finding algorithm and implement the algorithm.

$$\begin{aligned} f(x) &= 20x^3 - 50x^2 + 3x + 20 \\ g(x) &= \exp(0.5x) - \exp(0.6x) + 4 \end{aligned}$$

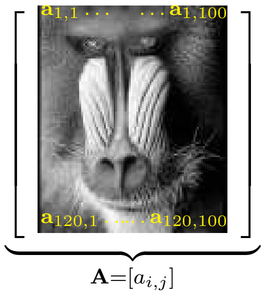
For both functions, start Newton's algorithm at $x = 0$ and terminate after it converges to a stationary point such that the candidate root points x found at two successive steps do not differ up to the fourth significant fractional digit, i.e. $|x_i - x_{i-1}| \leq 0.0001$.

b. For each iteration of the algorithm, output the iteration number, the current estimate of the root and the value of the function there. For example, for $f(x) = \log(x) - 1 + \exp(-x)$, you would show:

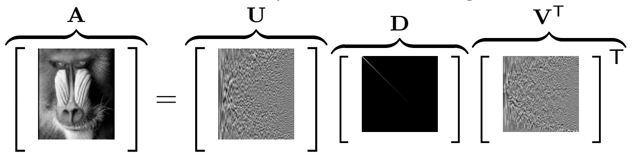
Iteration	x_i	$f(x_i)$
0	1.00000	-0.63212
1	2.00000	-0.17152
2	2.47034	-0.01109
3	2.50496	-0.00005
4	2.50511	-0.00000

c. Write a function to find the root of the function $f(x)$ in part a. using the bisection method. Run the function starting at $a = 0$ and $b = 1$. Output the value of a, b and $f(x)$ evaluated at the midpoint of a and b at each iteration. Stop when $|a - b| < 0.0001$. Comment of the speed of convergence compared to Newton's method for the same problem.

Problem 2: SVD based data compression [*12 points*]. For this problem, you will a need 120×100 pixel grayscale image of your own face (similar to the baboon picture below). You can use any digital camera and image processing software to capture a colour image of own face, crop and scale it down to 120 pixels (height) $\times$ 100 pixels (width), and convert it to a grayscale image in the pgm format. This will be used as a 120×100 rectangular matrix $\mathbf{A}$.

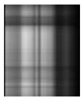

$\mathbf{a}_{1,1} \cdots$
$\cdots \mathbf{a}_{1,100}$
$\mathbf{a}_{120,1}$
$\cdots \mathbf{a}_{120,100}$
$\mathbf{A}=[a_{i,j}]$

a. Perform the SVD $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^{\mathsf{T}}$ of your facial image and illustrate the obtained results in your report: e.g.



For visualisation, values of the matrix elements U_{ij}, D_{ii}, and V_{ij} should be mapped linearly to the range $[0, 255]$, e.g. $255\frac{U_{ij}-U_{\min}}{U_{\max}-U_{\min}}$, where $U_{\max} = \max\limits_{i,j} U_{ij}$ and $U_{\min} = \min\limits_{i,j} U_{ij}$.

b. Approximate $\mathbf{A}$ with ρ singular values and columns of $\mathbf{U}$ and $\mathbf{V}$ for $1 \leq \rho \leq 100$ and illustrate the obtained results in your report: e.g.

ρ	1	2	3	4	5	...10	...20	...30	...40	...90
$\widehat{\mathbf{A}}_\rho$										

To visualise every approximate matrix $\widehat{\mathbf{A}}_\rho$, values that are less than 0 should be set to 0 and the values larger than 255 should be set to 255.

c. Find the absolute approximation errors and the level of compression (see below) with respect to the original image for different values of ρ. Tabulate these in your notebook, e.g.:

ρ	1	2	3	4	5	…10	…20	…30	…40	…90
Max error	130	139	119	115	120	107	80	38	31	0.5
Mean error	31	23	20	16	14	10	6.5	4.5	2.9	0.04
Compression,%	98.2	96.3	94.5	92.6	90.8	81.6	63.2	44.8	26.3	n/a

The compression achieved by an approximation is measured by the ratio of the number of real numbers used in the approximation to the number of real numbers used in the original. The number of real numbers used in the original is 120×100, so if k_ρ is the number used in the ρth approximation, the compression achieved is $1 - \frac{k_\rho}{12000}$. Since the compression should always be positive, there is no compression for large ρ.

Describe what negative values for compression correspond to and find the smallest value of ρ for which compression is negative.

d. Determine the most compressed approximation that is acceptable (to you) in terms of visual quality, explain why you chose this one and state the corresponding compression and mean error.

Requirements: Use Python to write your code and present your results in a Jupyter notebook. For matrix algebra, use numpy.linalg.
A method for reading and displaying pgm (grayscale) is given in a notebook on the course resources page.

Submission: Submit your notebook containing your code, output, analysis and discussion as a .ipynb file and as an html file (with all output showing). Make it clear which question you are answering. Also include the pgm picture you used. Submit your files on or before Friday, March 24, 2017, 6:00 pm via the ADB `https://adb.auckland.ac.nz/`.
Please complete your work on time as extensions will be granted only in special circumstances.

Brief marking scheme for Problems 1–2: You will be marked on the both the accuracy and clarity of your answers.

- Your code must run and produce correct answers. It should also be sensibly and clearly commented so the marker can understand your logic.
- Your notebook should describe your implementations of algorithms for solving Problems 1 and 2, present numerical and pictorial outputs produced by your algorithms, provide your evaluations of these algorithms and discussion of the results.
- Your notebook should be coherent and address the problems in full sentences. Do not just dump data to the report.