Part IA Paper 4: Mathematical Methods

Examples Paper 2 – Computing

*Straightforward exercises are marked †, challenging problems are marked *, optional problems beyond the expected level are marked **.*
Answers can be found at the back of the paper.

– You may need to revise material in the Michaelmas Term Activity Notebooks to complete this examples paper.

– For questions that involve computation you should implement your algorithms in Python. Under examination conditions you would be asked to express algorithms in Python/Python-like pseudocode. The most important requirement of pseudocode is that it clearly describes an algorithm.

1. † You are responsible for the implementation of a new software system for vehicle congestion zone charging. Each day, the license plate numbers of vehicles entering the charging zone are recorded. You receive each day a new, unsorted list of all registered motor vehicles in the UK (vehicles are identified by license plate number) from the Driver and Vehicle Licensing Agency. The list of all vehicles in the UK has approximately 26 million entries (n). Your task is to perform look-ups to find the record of each vehicle that enters the charging zone (daily). The number of vehicles entering the charging zone each day (m) varies.

   Experiments on the charging system computer have measured the average time cost of the following operations on lists of length n[1]:

   Linear search: $10^{-8}n$ (applicable for any list)

   Binary search: $10^{-6}\log n$ (requires that list is sorted)

   Sorting a list: $2 \times 10^{-5} n \log n$

   By assessing the time cost of algorithms that use the above operations:

   (a) What look-up algorithm would you suggest for the cases $m = 10^3$, $m = 10^4$, $m = 3 \times 10^5$ and $m = 5 \times 10^5$?

   (b) Determine the value of m at which you would suggest switching look-up algorithm.

2. † The following function computes the eigenvalues of the the real, symmetric matrix $A + B$, where A and B are both matrices (2D NumPy arrays):

```
import numpy as np
    def evals(A, B):
    return np.linalg.eigvalsh(A + B)
```

   (a) To ensure that the function is not incorrectly used, describe in words what checks you would you add. Divide checks into 'fast' and 'slow' based on the algorithmic complexity[2]. Bear in mind that exact comparison of floats is not robust.

   (b) Propose automated (unit) tests for this function that test for correctness.

[1] The presented time cost expressions use the natural logarithm, which in mathematics and many programming languages is denoted by 'log'.

[2] In practice, it would be common to always perform fast checks, whereas slow checks would only be performed in 'debug' mode.

3. In a particular program, we can evaluate a mathematical function $f(x)$ for values of x. However, we do not know the derivative of the function so we estimate it using difference equations. Determine the order of accuracy (order of the error), using a Taylor series, for the cases:

   (a) One-sided difference:
   $$\frac{\mathrm{d}f}{\mathrm{d}x}(x) \approx \frac{f(x+h) - f(x)}{h}$$

   (b) $*$ Symmetric difference:
   $$\frac{\mathrm{d}f}{\mathrm{d}x}(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

   when h is small. Develop a Python program to experimentally test the order of accuracy of the two methods.

   Suggest how you would test an implementation of numerical differentiation for correctness.

4. (a) Describe in Python/Python-like pseudocode an algorithm that computes a three-point moving average for an array of numbers. Implement your algorithm as a function and apply it to an array generated by evaluating $f(x) = \sin(x) + \cos(10x)/5$ at 50 equally spaced points in the interval $[0, 2\pi]$. Plot the original and smoothed data.

   (b) Grey-scale images can be represented as matrix, where each matrix entry represents a pixel and the value of the entry is the pixel intensity. A method of processing images is 'matrix convolution'. Given an image matrix A and a 'kernel' matrix G (kernel is assumed to be $n \times n$, with n being odd), the ij component of a convolved image matrix B is given by:
   $$B_{ij} = \sum_{k=0}^{n-1} \sum_{l=0}^{n-1} G_{kl} A_{(i-d+k)(j-d+l)},$$
   where $d = n/2$ (integer division).

      i. For a $n \times n$ kernel G, describe in Python/Python-like pseudocode an algorithm that computes a matrix convolution. Take care at the edges of the matrix. Implement your algorithm in a Jupyter notebook and test it on the below image of the Baker Building South Wing[3]:

      using the 'edge detection' kernel ($n = 3$):
      $$G := \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}.$$
      A notebook that loads the Baker Building image as a matrix and displays it can be found at `https://github.com/CambridgeEngineering/PartIA-Computing-Examples-Papers`.

      ii. † Give the complexity of the algorithm for an image with $p \times q$ pixels.

[3] `http://www-g.eng.cam.ac.uk/125/1950-1975/images/southwing.jpg`

5. It is 1969 and Apollo 11 has landed on the moon. But, the recording of Neil Armstrong's words as he makes the first moon walk[4] has background noise. Fortunately, four years earlier Cooley and Tukey re-discovered the 'fast Fourier transform'. Use a discrete fast Fourier transform[5] to eliminate specific frequencies from the recording to reduce the background noise without overly affecting Armstrong's speech. Which frequencies would you suggest removing?

   A notebook on which you can model your implementation is available at `https://github.com/CambridgeEngineering/PartIA-Computing-Examples-Papers`.

Optional extension 1: Perform a discrete Fourier transform of your own voice, and by experimentation estimate at which frequencies common letter sounds are 'lost'.

Optional extension 2: Implement an equaliser that manipulates the amplitudes of different frequencies of a sound track. Many interesting sound tracks can be found on the Internet.

6. Consider a *cost* function $f(\boldsymbol{x})$ that depends on design variables $\boldsymbol{x} := (x_0, x_1, \ldots, x_{n-1})$. The objective is to find $\boldsymbol{x}$ that minimises the cost f. This is known as an *optimisation* problem.

   When the cost is minimised, $\partial f / \partial \boldsymbol{x} = 0$ and Newton's method can be used to solve this problem. Newtons's method for a problem with a single variable x involves:

$$
\begin{aligned}
x_{n+1} &= x_n - \left(\frac{\mathrm{d}^2 f(x_n)}{\mathrm{d}x^2} \right)^{-1} \frac{\mathrm{d} f(x_n)}{\mathrm{d}x} \\
x_n &\leftarrow x_{n+1}.
\end{aligned}
\tag{1}
$$

   Iteration stops when $|\,\mathrm{d}f(x_n)/\,\mathrm{d}x| / |\,\mathrm{d}f(x_0)/\,\mathrm{d}x| < \texttt{tol}$. For a cost function of more than one variable, the cost is minimised when $\boldsymbol{g} := [\partial f/\partial x_0, \partial f/\partial x_1, \ldots, \partial f/\partial x_{n-1}]^T = \boldsymbol{0}$[6]. Newton's method in this case involves:

$$
\begin{aligned}
\boldsymbol{x}_{n+1} &= \boldsymbol{x}_n - \left[\boldsymbol{J}(\boldsymbol{x}_n)\right]^{-1} \boldsymbol{g}(\boldsymbol{x}_n) \\
\boldsymbol{x}_n &\leftarrow \boldsymbol{x}_{n+1},
\end{aligned}
\tag{2}
$$

   where $[\boldsymbol{J}]_{ij} := \partial^2 f / \partial x_i \partial x_j$ is a matrix. Iteration stops when $\|\boldsymbol{g}_n\| / \|\boldsymbol{g}_0\| < \texttt{tol}$. Newton's method requires an initial guess, $\boldsymbol{x}_0$, to start.

   (a) Implement Newton's method for a cost function with one variable (eq. (1)) in Python. Test your implementation for a quadratic cost function and for $f(x) = x^3 + x^2$.

   (b) * Implement Newton's method for a cost function with more than one variable (eq. (2)) in Python. Use your implementation to compute the minimum value of the *Rosenbrock function* $f(x_0, x_1) = (a - x_0)^2 + b(x_1 - x_0^2)^2$, with $a = 2$ and $b = 100$. Use starting values of $x_0 = 1.1$ and $x_1 = 1.1$, and $\texttt{tol} = 10^{-9}$. Report also the values of x_0 and x_1 at which f is minimum.

   (c) When would you expect Newton's method to converge in one iteration?

   (d) † Suggest a correctness test for your implementation.

   Hint: To evaluate $\left[\boldsymbol{J}(\boldsymbol{x}_n)\right]^{-1} \boldsymbol{g}(\boldsymbol{x}_n)$, do not compute the inverse of $\boldsymbol{J}$ but use the NumPy function `numpy.linalg.solve(J, g)`.

7. Newton's method in Question 6 for minimising a function $f(\boldsymbol{x})$ requires first ($\boldsymbol{g} := [\partial f / \partial x_i]$) and second ($\boldsymbol{J} := [\partial^2 f / \partial x_i \partial x_j]$) derivatives of f.

[4] `https://www.nasa.gov/62284main_onesmall2.wav`

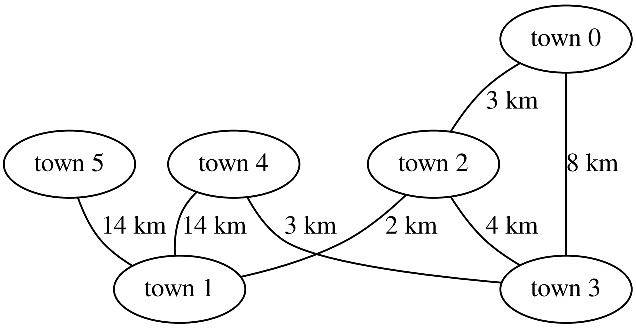
[5] Discrete Fourier transform are covered in Part IB. It is not necessary to understand how they work for this question.

[6] If you are unfamiliar with partial derivatives, look at the 'Difference equations and partial differentiation' handout from the Part IA Michaelmas Term (standard pace) mathematics course.

(a) Design a Python class for the Rosenbrock function that would be suitable for use in Newton's method. Use Python/Python-like pseudocode to present the implementation of your class.

(b) Test the implementation of your class by modifying your implementation of Newton's method to use your new class.

8. The below *weighted graph*[7] represents a road network between towns. The towns are the *nodes* of the graph, and road connections between towns are the *edges* of the graph. Each edge has a weight which, in this case, is the distance between the nodes (towns) connected by the edge (road).



(a) † Select a suitable data structure to store the graph, and build a graph for this network in Python.

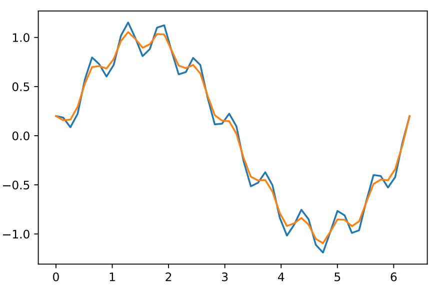
(b) † By inspection, what is the shortest route between town 0 and and town 5?

(c) ** Develop a recursive algorithm that computes the shortest route between two towns. Test for routes between different towns.

One the most famous algorithms is Dijkstra's algorithm (`https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm`) – it finds the shortest path between two nodes of a weighted graph. It is much more efficient than a simple recursive algorithm.

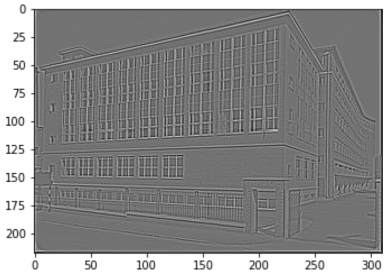
Answers

1. (a) - (b) Approximately 34000
2. (a) - (b) -
3. (a) $O(h)$ (b) $O(h^2)$
4. (a) .



[7]A graph is a very powerful concept. Graphs are used to represent road, communication and social networks, and many other problems that involve connections. There are numerous classic algorithms for operating on graphs, especially for finding the shortest path between nodes.

(b) i. .



ii. $O(pq)$

5. -

6. (a) - (b) $f_{\min} = 0.0$, $(x_0, x_1) = (a, a^2)$ (c) - (d) -

7. -

8. (a) A `dictionary` (map) with the town name as the key, and a `list` of (town, distance) `tuples` as the value, e.g.:

```
nodes = {}
graph["town 0"] = [("town 2", 3), ("town 3", 8)]
graph["town 1"] = . . . .
```

(b) town 0 – town 2 – town 1 – town 5 (19 km)

(c) -