SOLUTION: DO NOT DISTRIBUTE

Thursday 13 May
Expected Duration: 1 hour 30 minutes
Start Time: Available from 9:30 BST
Time Allowed: 3 hours
Timed exam in 24 hours

DEGREES of MSci, MEng, BEng, BSc, MA and MA (Social Sciences)

DATA FUNDAMENTALS (H)

COMPSCI 4073

(Answer 2 from 3 questions) Answers involving code do NOT need working code to get full marks. Do not spend excessive time getting code to run.

This examination paper is an open book, online assessment and is worth a total of 60 marks.

1. You are building a system to help optimise the engine performance for racing cars. There is a pre-built simulator that can simulate engine performance for a given driver control signal and a set of engine tuning settings.

(a) The lead engineer instructs you to build software to work with an objective function $L(\mathbf{x};\theta)$. Identify the parameters θ and the input $\mathbf{x}$ in the description above.

[2]

Solution:

- parameters are the engine tuning settings **[1]**
- input data are the driver control signals **[1]**

(b) There are many possible objective functions that are important, such as peak velocity, engine wear, fuel efficiency, vibration level. An engineering team has come up with over 300 possible objective functions, indexed $L_0, L_1, \ldots$.

You are using an automatic differentiation library. This library computes the *entire* Jacobian matrix J at a given parameter vector θ and *all* of the objective functions L_i. That means that the Jacobian has size $n \times K$, where n is the dimension of θ and K is the number of sub-objective functions.

(i) Write NumPy code to compute the Euclidean norm of the gradient vector for the first sub-objective function L_0 only, $l_0 = ||\nabla L_0(x;\theta)||_2$ from the matrix `J`.

[3]

Solution:

```
l_0 = np.linalg.norm(J[0,:])
```

- **[1]** for norm
- **[1]** for slicing somehow
- **[1]** for correct slice of matrix

(ii) Explain how this quantity l_0 might be used in a gradient descent algorithm.

[3]

Solution:
Termination of the gradient descent **[1]** would typically occur when l_0 **[1]** drops below some set threshold **[1]**

(c) Below are three graphs showing the convergence of gradient descent applied with respect to one objective function with various settings for the hyperparameter δ.

(i) The three graphs correspond to $\delta = 0.000001$, $\delta = 0.05$ and $\delta = 0.5$. Identify which graph goes with which setting of δ and give a short justification of why you believe this to be the case.

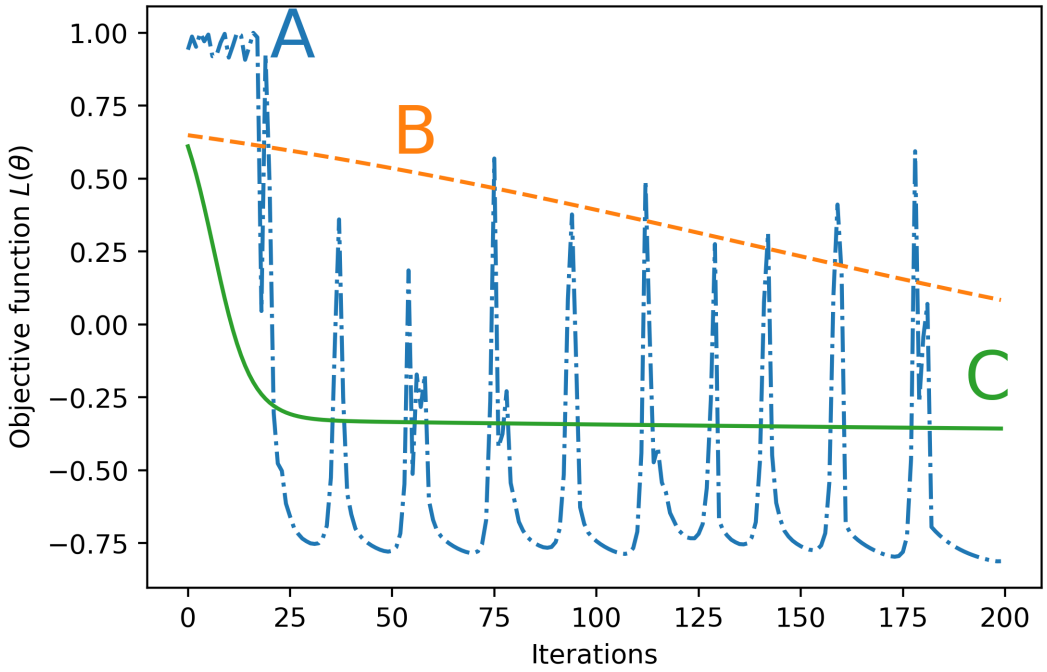

Figure 1: Three gradient descent convergence graphs.

[6]

Solution:

(a) is 0.5 as it has very noisy behaviour and does not converge

(b) is 0.000001 and makes very slow progress

(c) is 0.05 and as it converges smoothly and relatively quickly

(ii) It has been suggested that the best objective function to use for this problem is the sum of all of the previously identified objective functions over all the data

$$L(\theta) = \sum_i L_i(\mathbf{x}; \theta).$$

To cope with the memory constraints, it has been suggested to modify **gradient descent** to reduce the memory load. However, the input data $\mathbf{x}$ cannot be split into batches because it represents a continuous time series. Give a detailed outline of an approach to reduce memory.

[6]

Solution:
Use stochastic gradient descent **[1]** but split by objective function **[1]** Compute the gradient for some (random) subset of sub-objective functions **[1]** sum it **[1]** and apply to the parameters **[1]** and then repeat **[1]** with different random batches of sub-objective functions.

(d) When optimising the objective function $L(\theta)$, a problem often arises where the values of θ are useless, as the values are not physically realisable in the engine. The engineers have come up with a list of minimum and maximum values for the parameters, θ_{min} and θ_{max} Suggest how to incorporate these limits, while still using the same optimisation algorithm.

[4]

Solution:
Add a penalty function **[1]** which is a function added to $L'(\theta)$, **[1]** which increases as the limits are violated **[1]** For example $L(\theta) = L'(\theta) + \lambda(\theta)$, where $\lambda(\theta)$ smoothly increases as θ exceeds the limits **[1]**

(e) The racing team has observed there is an apparently random bug in the optimisation code that causes code to fail 0.5% of the time without explanation. A new automatic testing package is being proposed, which will detect 90% of all random bugs present, and will only falsely alert that a bug is present when there is not 5% of the time. Ignoring any bugs, each optimisation has an average return in increased race winnings of of £1,000. If the team runs an optimisation and:

- there is no bug, they have no additional cost.
- there is a bug, they don't know about it, they lose £5,000 from crash recovery costs.
- there is a bug, but it is detected, they lose £100 to re-run the simulation.
- there is *no* bug, but it falsely detected that there is one, they lose £1000 wasting time.

Is the automatic testing package worthwhile *on average*? Justify your reasoning with a calculation.

[6]

Solution:

- **[1]** for using expectation in any form
- **[1]** for computing probability in no test case
- **[1]** for computing correct expectation in no test case
- **[1]** for enumerating cases in test case
- **[1]** for computing probabilities in test case

- **[1]** for computing expectation correctly
- Case no software; Can never detect a bug, so expected return of an optimisation run is:
  - EX = +1000 - 5000 * 0.005 = £975
- Case software:
  - P(¬B ^ T) = 0.995 * 0.05 = 0.04975
  - P(¬B ^ ¬T) = 0.995 * 0.95 = 0.94525
  - P(B ^ T) = 0.005 * 0.9 = 0.0045
  - P(B ^ ¬T) = 0.005 * 0.1 = 0.0005
  - EX = 1000 - P(¬B ^ T) * 1000 - P(¬B ^ ¬T) * 0 - P(B ^ T) * 100 - P(B ^ ¬T) * 5000
  - EX = +1000 - 0.04975 * 1000 - 0.94525 * 0 - 0.0045 * 100 - 0.0005 * 5000
  - = £947.30

Not worth using the software.

```
std_return = 1000
fail_rate = 0.005
detect_rate = 0.9
false_rate = 0.05

u_no_bug_no_detect = 0
u_bug_no_detect = -5000
u_bug_detect = -100
u_no_bug_detect = -1000

case_a = (std_return +
            u_bug_no_detect * fail_rate +
            u_no_bug_no_detect * (1-fail_rate))

case_b = (std_return +
            u_bug_no_detect * fail_rate * (1-detect_rate) +
            u_bug_detect * fail_rate * detect_rate +
            u_no_bug_detect * false_rate * (1-fail_rate) +
            u_no_bug_no_detect * (1-fail_rate) * (1-false_rate))

print(case_a, case_b)
```

Note: students do not have write any code – this is just by way of explanation

2. You are employed to analyse data from a set of vibration sensors mounted on an autonomous vehicle. There are 30 sensor packs deployed across the chassis, each pack recording vibration from 8 sensor channels mounted around the edge of the unit. The sensors are sampled at 500Hz.

(a) One minute of data is stored in a standard (30000, 30, 8) NumPy tensor of `float64`. It is known that the sensor 18 failed after 2.5 seconds (exactly) and recorded a value so negative that it overflowed on all 8 channels.

Using your knowledge of array layouts and IEEE754, at which byte offset from the start of the array data in memory would you **first** be guaranteed to find a **byte** with value 255 (`1111 1111` in binary)?

Show your working clearly and state any assumptions made.

[12]

Solution:

- **Note: because of ambiguity about 0 versus 1 indexing, any number in the working can be off +/-1 without losing marks.**
- 2.5 seconds @ 500Hz = 1250 samples in **[1]** for number **[1]** for working
- so we know that we are looking at sensor 18 **[1]** and the first channel will be the first one we **[1]** array element [1250, 18, 0] **[1]** for index
- `float64` is 8 bytes long **[1]**
- We assume C/row major ordering (as NumPy standard) **[1]**
- Therefore the byte strides are [1920=`8*8*30`, 64=`8*8`, 8] **[1]**
- We know that an overflow negative value will become -inf in IEEE754 **[1]**
- And the bit pattern for -inf is all ones **[1]**
- So if we look at the byte $8 \times 0 + 64 \times 18 + 1920 \times 1250 = 2401152$ **[1]** for computation
- We will be guaranteed to find a byte of value 255 **[1]**
- [Note: endianness does not affect this answer]

(b) The sensor units are poorly shielded and are subject to significant electrical disturbances, which introduces high-frequency noise. Additionally, the wiring is flaky and occasionally the connection momentarily breaks, creating occasional spike noise.

Suggest a simple signal processing strategy to reduce these problems, and indicate any parameters that could be controlled by engineers to control the results.

[6]

Solution:

A median filter **[1]** (or order filter, or non-linear filter) could eliminate spike noise **[1]**

A simple low-pass filter like moving averages **[1]** (or any other filter suggestion) would remove the high-frequency noise **[1]** The median filter should be applied first, to avoid spreading the spikes out **[1]**

The window length **[1]** of the moving average and the median filter can be used to vary the "strength" of the filtering; longer windows will more aggressively smooth the signal.

(c) The vibration units are used as inputs to a machine learning system that classifies road state into four categories of smoothness (0,1,2,3). These road states are computed once a second. You are asked to comment on an approach for modelling an entire road using a probabilistic model of these smoothness states.

Three example sequences from different roads on a test track are shown below:

```
A [3 3 3 3 3 3 3 3 3 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 3 3]
B [3 0 2 2 1 3 2 3 1 0 2 1 3 2 3 2 2 1 3 1 1 0 1 2 0 1 2 3 0 1 1 2]
C [0 1 0 1 1 1 0 0 1 0 0 1 0 1 0 0 0 0 0 0 2 0 0 2 1 0 1 0 1 1 0 0]
```

(i) Without doing a calculation of any kind, give a suggested ranking of these sequences in order of entropy of the probability distribution that generated them, smallest entropy first. Give a brief justification.

[3]

Solution:

- A, C, B **[1]** for correct order
- A is hardly ever changing and so probability is concentrated on one outcome **[1]**
- B is very variable, where as C sticks mainly to two possible outputs **[1]**

(ii) How would you compute the empirical PMF for each sequence, assuming each symbol is generated independently from all the others? Include the appropriate equation in your answer. Do not perform any calculation.

[2]

Solution:
Compute the sum of the count of each symbol, divided by the total number of symbols **[1]**

$$P(s) = \frac{N_s}{N}$$

[1] or any similar equation

(iii) If you were asked to compute the likelihood of one entire sequence (e.g. for one whole road) based on this model, how could you compute this?

[3]

Solution:
The likelihood of the entire sequence would be the product **[1]** of the PMF **[1]** for each symbol, due to the independence assumption above **[1]**

(iv) Computing the likelihood directly can have interactions with floating point representa-

tions. State the typical IEEE754 exception that would occur during such a computation, the effect of this, and describe an alternative procedure that would minimise the occurrence of such floating point behaviour.

[4]

Solution:
This computation would be likely to cause an underflow **[1]** rounding the result down to zero **[1]** An alternative process would be to compute the log-likelihood **[1]** instead, and sum the log of the PMF of each symbol **[1]**

3. You are asked to analyse data from a system that analyses voices and infers a vector of "emotional" attributes for an input waveform from someone speaking. Each input waveform is converted to a 18 element vector representing this emotional state, as real numbers in the range [-1, 1]. You do not know the interpretation of any of the elements of an emotion vector.

The company creating the system has used 38 actors who have recorded 1,000 phrases each, and each has been transformed into an emotion vector.

(a) The data is to be stored in an ndarray `voice_data`. What would an appropriate ndarray data structure be to represent this dataset of emotion vectors? Use the concepts of shape and dtype.

[2]

Solution:

`shape = [38, 1000, 18]` (any permutation of this is acceptable) **[1]**

`dtype = float64 (or float32)` **[1]**

(b) You are asked to write a function that:

- given the indices of two actors (e.g. 5 and 13) and an integer k
- will compute the *average* emotion vector for each actor (i.e. an 18 element vector per actor)
- generate a sequence of k point evenly spaced in the emotion space between the two *specified* actors
- and return the list of indices of the (average) actors with emotion vectors closest (in the L_2 norm sense) to each of the k evenly spaced points between the actors.

This function is to be called with a signature like `actor_sequence(voice_data, from_actor, to_actor, k)`. Write code for this function. Note: your code does **not** need to run, but should illustrate the correct sequence of NumPy operations.

[11]

Solution:

```
# [+1] for def
def actor_sequence(voice_data, from_actor, to_actor, k):
    #[+1] for mean, [+1] for axis
    actor_averages = np.mean(voice_data, axis=1)
    indices = []
    # [+1] for linspace or for computing manually
    for a in np.linspace(0, 1, k):
        # [+1] for linear interpolation formula;
        # [+1] for applying correctly
        lerped = (1-a) * actor_averages[from_actor] + a * actor_averages[to_
```

```
        # [+1] for norm; [+1] for axis; [+1] for ord
        ds = np.linalg.norm(lerped-actor_averages, axis=1, ord=2)
        # [+1] for using argmin
        indices.append(np.argmin(ds))
    # [+1] for constructing and returning a list
    return indices
```

(c) An analyst suggests using the distance of datapoints to the extreme possible values (i.e. proximity to the bounding box edges at -1 and 1 on each axis) to detect outliers. Is this a sensible approach? Describe why or why not for this dataset.

[3]

Solution:
This is unlikely to work well **[1]** as the high-dimensional data **[1]** will typically exhibit similar distances **[1]** to the bounding boxes edges for all points.

(d) To analyse the data in a more human friendly format, it is suggested to project the dataset onto one dimension that best reveals the differences in emotion; that is the direction that spreads out the data furthest.

You have to compute this projection, without a library like NumPy, but a more basic library that supports only elementary vector and matrix operations (norm, multiplication, addition, dot product, transpose, inversion).

(i) Suggest a viable approach and the steps required to do this. **Do *not* write code – outline each step in words that a junior programmer who has not taken a course like Data Fundamentals could use to approach this problem.**

[8]

Solution:

1. Taking `voice_data`, compute the covariance matrix **[1]** as the mean squared difference **[1]** of the each column from the mean value of each column **[1]** Alternatively, the equation could be given here.
2. Find the first eigenvector **[1]** $\mathbf{x_1}$, using power iteration **[1]** starting from a random vector $\mathbf{x}$ **[1]** computing a $\Sigma\mathbf{x}$, normalising **[1]** and repeating until it ceases to change **[1]** Alternatively the equation could be stated.
3. Project onto this vector **[1]** by computing the dot product **[1]** of each row of `voice_data` with $\mathbf{x_0}$ and collecting the results into a vector.

(ii) The covariance matrix Σ is found to be low rank. What can you infer about the eigenspectrum of the covariance matrix?

[2]

Solution:
Many (or most) of the entries **[1]** in the covariance matrix are 0 **[1]**

(e) The analyst produces the 2D representation of the data, including a scatterplot and the approximate density of points. Describe, *briefly*, the form of the following elements of the Layered Grammar of Graphics that appear in this plot:

- layers
- guides
- geoms
- stats

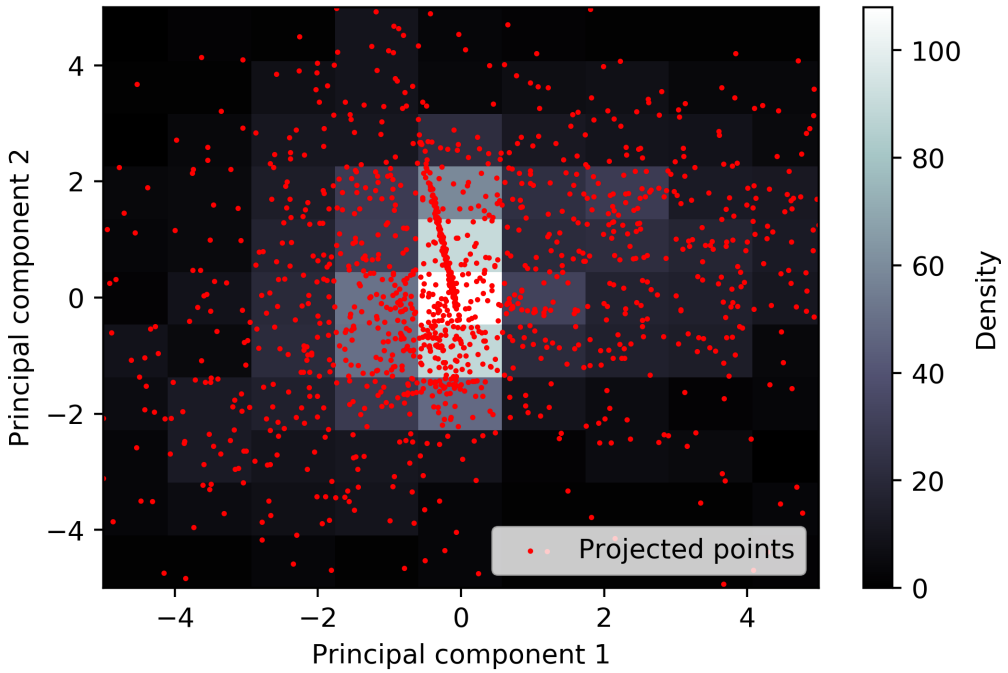

Figure 2: A plot of the emotion vectors in 2D, showing the projected vectors (red) and the approximate density of vectors (gray squares).

[4]

Solution:

- layers: there are two *layers*; the histogram layer and the points **[1]**
- guides: there are axis markers giving scales for the coordinates and a color-bar giving a colour scaling **[1]** (for both; 1/2 each)

- geoms: there are *point* geoms to indicate data points *and/or* there are geoms to represent *bins* **[1]** for either
- stats: the binning in the histogram is a stat **[1]**

Mark table

Question	Points	Score
1	30	
2	30	
3	30	
Total:	90	