# Specimen

1.

a) The posterior is directly proportional to the likelihood and the prior, and is inversely proportional to the evidence.

b)

i)

Expected profit = E(f(X)) = sum P(X=x) f(x)

Profit for correct wafer = v – c. Profit for incorrect wafer = - c.

Expected profit

ii) T=0 – test is negative. T=1 – test is positive.

Assumption:

99% reliability means that:

**P(T=1|D=1)=0.99  
P(T=0|D=1)=0.99**

Then:  
P(T=1|D=0)=0.01  
P(T=0|D=1)=0.01

**Bayes rule**:

P(D=1|T=1) = P(T=1|D=1) \* P(D=1) / P(T=1)  
= 0.99 \* 0.00001 / ( P(T=1|D=1) \* P(D=1) + P(T=1|D=0) \* P(D=0) )  
= 0.00001 / (0.99 \* 0.00001 + 0.01 \* 0.99999 )

= 0.00001 / 0.01

= **0.001**

Only every **thousandth** wafer would be faulty when the test is positive, which seems very **useless**.

c)

i) The objective function is the **number of defects**. The parameters are the **temperature, dopant bias, slice thickness**. The constraint is for the temperature **not be higher than 900C**.

ii) The metaheuristic for simulated annealing sometimes accepts changes **away from local minima** in order to possibly find the **global minimum**. This provides **resistance to getting stuck**, unlike stochastic hill climbing.

iii) Convergence of optimisation visualised by a plot of objective function value (**number of defects**) against **iteration**.

iv) [Pareto optimality – not a thing in our syllabus]

2.

a) **Loops** should always be avoided in order to take full advantage of parallelised operations on vectors.

b)

n = **len(x)**  
exponent = **np.arange(n)** **/ k**  
z = **np.sum**(**x \*\*** exponent **\*** **y \*\*** exponent)

c)

i) The three values are:

* Positive infinity if the mantissa is **all zeroes** and sign bit is positive (**0**)
* Negative infinity if the mantissa is **all zeroes** and sign bit is negative (**1**)
* NaN if the mantissa is **non-zero**

ii) Because of the implementation method, the mantissa will always be between 1 (inclusive) and 2 (exclusive), so it will always start with one, but does not need to be stored every time, which would uselessly take up space.

iii) Sources:

* If a and b are **very similar**, there could be information loss in their subtraction (**cancellation error**)
* If (a-b) and c are **very different magnitudes**, there could be information loss in their addition (**magnitude error**)
* If d is very small, **division magnification** is very likely

d) Since the **inverse of a diagonal matrix** is just the **reciprocal of its entries** and the **product of a diagonal matrix with a vector** is the **elementwise product of the diagonal with that vector**, the more efficient solution would be:

**y = 1.0 / np.diag(A) \* x**

e) Linear regression?

Markov Chain Monte Carlo?

Likelihood function?

Histograms of the posterior samples (traces)?

Storing and computing the probabilities in "ordinary" form will be likely to be subject to **underflow**. **Log-probabilities** could be used to do computations with much less risk of underflow. In particular, **multiplying together many likelihood terms** can be done by **summing log-probabilities**. Display might be best converted to **odds** for user understanding.

3.

a) Nyquist limit – the limit of frequencies in a sampled signal under which it is possible to recover an original continuous time signal ­*x(t)* perfectly from this signal.

If a signal is sampled with a higher frequency than this limit, aliasing (unpleasant artifacts appearing) occurs.

b) …

# 2020/2021

1.

a) Parameters: engine tuning **settings**. Input: driver control **signal**.

b)

i) np.linalg.norm(J, 0)[:, 0]

Correct: np.linalg.norm(J[0, :])

ii) ~~A gradient descent algorithm would keep this as a variable to see which direction to continue in and how quickly.~~

**Stop** gradient descent when **l0** drops below some **threshold**.

c)

i) A: delta = 0.5. The value of the objective function changes very drastically and **erratically**, meaning that too much jumping is happening, where some good values are jumped out of.

B: delta = 0.000001. The objective function is extremely **slow** but is slowly going in the right direction.

C: delta = 0.05. A healthy middle, where no jumping is happening, however, some local minimum caused it to get stuck in it and cannot be escaped with this value. Converges

ii) ~~This time series could be reduced by sampling. To not let aliasing happen, the sampling rate needs to be higher than ???, which can also be described by the Nyquist limit.~~

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

d) By installing a **penalty function**, soft constraints can be implemented as an increased value to lambda(theta), which smoothly increases as theta exceeds limits. L(theta) = L’(theta) + lambda(theta)

e)

P(bug) = 0.005  
P(found|bug) = 0.9  
P(found|no bug) = 0.05

P(not found) = 0.05

P(no bug) = 1 – P(bug) = 0.995

**E**(optimisation) =   
1000 – 0\*P(no bug) – 5000\*P(not found|bug)\*P(bug) – 100\*P(found|bug)\*P(bug) – 1000\*P(no bug)\*P(found|no bug) =   
1000 – 5000\*0.1\*0.005 – 100\*0.9\*0.005 – 1000\*0.995\*0.05 = 947.3

Not using software:

E(optimisation) = 1000 – 5000\*P(not found|bug)\*P(bug) = 1000 – 5000\*0.05\*0.005 = 975

The profit is larger not running the software, so not worth it.

2.

a)

3000 = 500 \* 60 (every second in a minute with 500Hz)  
float64 = 8 bytes

sensor 18 = tensor[?, 17, :]

2.5 \* 500 = 1250 => tensor[1249, 17, 0]

Assumption: C order

strides: 8\*8

offset = 0 \* 8 + 17 \* 30 + ???

8/12, nice

b) Median (/order/non-linear) filter would eliminate spike noise. Low-pass filter (e.g., moving averages) would remove high-frequency noise. Median filter should be applied first to avoid spreading spikes out.

c)

i) Smallest entropy first: A (almost all 3’s) -> C (almost all 0’s or 1’s) -> B (all 3 are pretty frequent, varied outcomes).

ii) P(X=x) = n\_x / N

P(X=0) = x\_0 / 32  
P(X=1) = x\_1 / 32  
…

iii) Multiply all likelihoods for each occurrence of a symbol because they are independent.

iv) There could be a roundoff error (underflow) since multiplying many small fractions is very susceptible to this. The wrong value could be computed, which could even be 0, but this can be remedied by using log likelihoods and then summing the log of the PMF of each symbol.

3.

a) dtype = float64. Shape = (38, 1000, 18)

b)

actor\_sequence(voice\_data, from\_actor, to\_actor, k):

actor\_avg = np.mean(voice\_data, axis=1)

indices = []

for a in np.linspace(0, 1, k):

lerped = 1(-a) \* actor\_avg[from\_actor] + a \* actor\_avg[to\_actor]

ds = np.linalg.norm(lerped – actor\_avg, axis=1, ord=2)

indices.append(np.argmin(ds))

return indices

c) 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)

i)

1. Find covariance matrix (mean squared difference of each column from the mean value of each column)

2. Find eigenvector by doing power iteration on a random vector (multiply, normalise, repeat) until it stops changing.

3. Project onto vector by doing dot product of each row with x\_0 and collecting results into a vector.

ii) Many (or most) of the entries in the covariance matrix are 0.

e) Layers: 2 (histogram and points)

Guides: legend (?) axis markers for coordinate scales and colour-bar for colour scaling

geoms: point geoms for data points

stats: binning in the histogram

# 2019/2020

1.

a)

def max\_posterior(zone):

**LOG\_PRIOR\_THETA = NP.LOG(PRIOR(NP.ARANGE(112)))**

posteriors **= llik(scan\_data)** \* prior(scan\_data[:, zone]) / evidence?

**LOG\_LIK = LLIK(SCAN\_DATA)**

**POSTERIOR\_UN = NP.EXP(LOG\_PRIOR\_THETA + LOG\_LIK)**

**POSTERIOR = POSTERIOR\_UN / NP.SUM(POSTERIOR\_UN, AXIS=1)**

**MAX\_INDICES = NP.ARGMAX(POSTERIOR, AXIS=0)**

**RETURN MAX\_INDICES**

return np.max(posteriors, axis=0)

b)

Why MLE is not Bayesian:

* Does not quantify uncertainty/single point estimate
* Does not incorporate a prior
* Does not treat parameters as random variables

What would be required?

* A prior distribution over the parameters
* Some form of inference approach (e.g., Monte Carlo sampling)

Advantages:

* Would give full uncertainty on parameters

Disadvantages:

* May be harder/slower to fit
* Specifying a good prior might require additional research/expert input

c) Odds because they are very **understandable** to humans. Work well for both **very likely** and **very unlikely** values. **Very large odds** ratios may **not** be **informative**.

(Logits are better for scaling extreme events, but harder for people to interpret)

d)

i) **LOGARITHMIC Polar** coords are better for analogy of the roundness of the telescope (better for human understanding).

* Log-probabilities on a linear coord system would look the same as this representation; using a log coord would not be useful [1]
* Logits would require using signed values [1] which are hard to represent on a polar plot without an obvious “zero x axis”
* The same applies to odds, which would probably require two “sides” broken at 1:1 and would scale [1]
* The plot shows raw probabilities with a log scale, giving the advantages of log-probability representation without a change of units; a linear scale would make it very hard to distinguish smaller probabilities [1]

2.

a)

Power iteration. Illuminate a random point x\_0 on the project. Computer the inverse of B.

1. Measure z\_0 on the camera
2. Computer corresponding point y\_0 via y\_0 = B^(-1)z\_0
3. Normalise y\_0 via its infinity norm (or any other norm)
4. Illuminate a point x\_1=y\_0\_normal (or with some fixed scaling)

* Repeat steps 1-4 until x\_n close to x\_(n-1) using a vector norm to compute similarity
* Take x\_n and compute y\_n via B^(-1)z\_n
* Compute lambda = y\_n/z\_n to get the eigenvalue (assumes value is real)

b)

i)

Cons:

* A linear model can be fitted very quickly using standard algorithms which will converge
* Fitting a nonlinear model may require expensive optimisation which may/may not converge

Pros:

* Powerful analysis tools like eigendecomposition can be applied to the model
* Linear transforms of coordinates cannot represent distortions like perspective or curved changes

ii) This problem is a sum of simpler problems [+1] and stochastic gradient descent could be applied [+1]. This would allow for better resource management (e.g., smaller memory requirements) [+1] as well as potentially providing some form of stochastic relaxation [+1]. This would require some way of estimating derivatives efficiently [+1], for example, automatic differentiation [+1]

3.

a) **Microphone A**, since the **Nyquist limit** should be above **300Hz** and is such with this microphone (half of 600Hz sampling rate is 300, which is just above the urban noise limit). **Aliasing** if microphone D, **too expensive** with microphones B and C when A gets the job done.

b) The Dirac delta function is the **identity** with respect to **convolution** [+1]. An approximate delta impulse could be produced by **tapping**/striking/etc. near the glass [+1] and the corresponding signal **recorded** [+1]. This would be the convolution kernel desired. This might be difficult both because of **background noise** [+1] and because generating a perfect delta function is **not feasible** by physical means [+1]. Also, the glass response might be **nonlinear** [+1] and thus not representable as a **convolution kernel** [+1].

c)

i)

* The guides (ticks) are both **ugly** and **poorly spaced**; they should be spaced evenly and span the data range
* The plot is faceted but would **better** be **layered**, or **at least** faceted with **consistent coords**
* The mapping is the wrong way around: **frequency** should be on the **x-axis** and the **amplitude** on the **y-axis**
* The **coords** are very badly **scaled** and **inconsistent**. A single set of coords covering the data should be chosen.
* There are **no guides** **to** **label** the axes. **Labels** with **units** should be present. (only 0.5 if units not mentioned!)
* There are **no geoms to represent uncertainty**, which is present in the table. Error bars or ribbon plots could show this.
* There are **no geoms to represent measurement points**. Point geoms could provide this.
* There is **no guide** **to identify the lines**/facets. A **legend** or **titles** could provide this. A good plot would look like:



# 2019/2020 Resit

1.

a) Point geoms **too big** (obscure/occlude). No **label on x axis**. No guide (**legend**) to **distinguish between the lines**. No **title** guide. Wrong type chosen – should not be scatterplot, maybe a median. **NO UNITS, AXES WRONG WAY ROUND, NO ERROR BARS.**

b)

i) Faceting – multiple different views of the same data next to each other. Coords could be **different**, but should be kept consistent. Layering – multiple views in one graph, **same** coords; thus, the **same mapping and scaling** of attributes must be applied to thee data attributes.

ii) **Size** could represent the level or **colour map**.

c)

i) Diagonal matrix (0 in the off-diagonal entries)

ii) Inverting A, computing y = A^(-1)v

d) Perceptually uniform – the same increase in value of data leads to perceptually the same increase in value of colour. This is not possible with a “rainbow” map since small changes are very hard to perceive for some colours, like cyan and blue/green. Additionally, the human brain is not conditioned well to understand which colour follows which in a progression, which makes the mapping ambiguous

Monotonic brightness – an increase in data will always increase brightness. This is not true for rainbow mapping, where brightness does not change sometimes, but seems to because of the varied colours, e.g., blue might be inherently darker than red in people’s perceptions.

**FALSE CONTOURS**

e)

i) The data must be reinterpolated [#2] onto a regular grid / evenly spaced values [#1] before convolution is applied, or the results will be meaningless.

ii) Moving average (with constant kernel)

iii) Nonlinear filter (like median) would be better at rejecting spikers

2.

a) Machine precision defines the *relative* error between a real number and its floating point representation. It is defined for a real number x as

IEEE754 guarantee ensures that the relative error in storing computations with a floating point number in the representable range will not exceed .

b) Floating point exceptions (excluding inexact floating point):

* Invalid operation: caused by 0/0, sqrt(-1.0), inf-inf, etc
* Division by zero, caused by x/0 for x not 0, NaN or inf
* Overflow, caused by a computation having a result (absolutely) larger than the largest representable value
* Underflow, caused by a computation having a result (absolutely) smaller than the smallest representable value

c) Objective function: flow rate of oil through the pipeline (to be maximised). Parameters: diameter of the pipe, inner diameter of the flow valves, pipeline pressure. Constraints: pipe diameter 900mm, pipe pressure 7.0MPa.

A heuristic local search algorithm (like hillclimbing), or 0th-order algorithm (random search, grid search)