Math087 - Review for Quiz 2 *Solutions*

George McNinch

quiz-date 2025-04-09

1 quiz 2

1. `max-flow / min-cut`

   Consider a (finite) weighted, directed graph G with source node `s` and terminal node `t`. To find the `min-cut` for the graph, we must consider all possible partitions of the nodes into an s-group and a t-group. For each such partition, we compute the `cut-value`, namely the sum of the labels on each of the directed edges connecting a node in the `s`-group with a node in the `t`-group.

   The smallest such `cut-value` is the `min-cut` for G.

   It turns out that `min-cut` is the solution to a linear program which is dual to the `max-flow` linear program for G.

   a. What is the objective function for the `max-flow` linear program?

   Solution:
   The value of the objective function for the `max-flow` linear program is given by the sum of the weights on those edges which originate at the `source node`.

   b. What does *strong duality* say about the relationship between `max-flow` and `min-cut`?

   Solution:
   According to the *strong duality Theorem*, the maximum value of the objective function for `max-flow` coincides with the minimal value of the objective function for `min-cut`.

2. *Bi-partite graphs and matching*

   Consider a bipartite graph G with vertices $U \cup V$ and edges E.

   a. What is meant by a *matching* for G? What does it mean to say that a matching is *maximal*? *perfect*?

   Solution:
   Recall that we denote an edge connecting $u \in U$ to $v \in V$ symbolically as $[u, v]$. A *matching* is a subset M of the edges E with the property that no two edges in M have

a common vertex. In symbols, if $[u, v]$ and $[u', v']$ are in M then $\{u, v\} \cap \{u', v'\} = \emptyset$ (or equivalently: $u \neq u'$ and $v \neq v'$).

A matching M is *perfect* provided that every vertex in the graph is adjacent to exactly one edge in M.

b. For a subset $X \subseteq U$ of vertices, the neighborhood $N(X)$ is the subset of V defined by

$$N(X) = \{v \in V \mid [u, v] \in E \quad \text{for some } u \in U\}$$

If $X = \{a, b\}$ and $N(X) = \{c\}$, can G have a perfect matching? Why or why not?

Solution:
No, G can have no perfect matching. Since $N(X) = N(\{a, b\}) = \{c\}$ there are edges $[a, c]$ and $[b, c]$ in E. Moreover, $[a, c]$ is the only edge in E adjacent to the vertex a and $[b, c]$ is the only edge in E adjacent to the vertex b. If M were a perfect matching, then M must contain $[a, c]$ in order to have an edge adjacent to a. But then M can't contain the edge $[b, c]$ since c is already adjacent to the edge $[a, c] \in M$. Since $[b, c]$ is the only edge adjacent to b, there can be no perfect matching.

c. Give the statement of *Hall's Matching Theorem* (aka *Hall's Marriage Theorem*).

Solution:
Hall's Matching Theorem states: If G is a bi-partite graph with finite sets of vertices $U \cup V$ and edges E, then G has a perfect matching if and only if for every subset $X \subseteq U$, the inequality $|X| \leq |N(X)|$ holds, where $|Y|$ denotes the number of elements in the finite set Y.

3. *state machines*

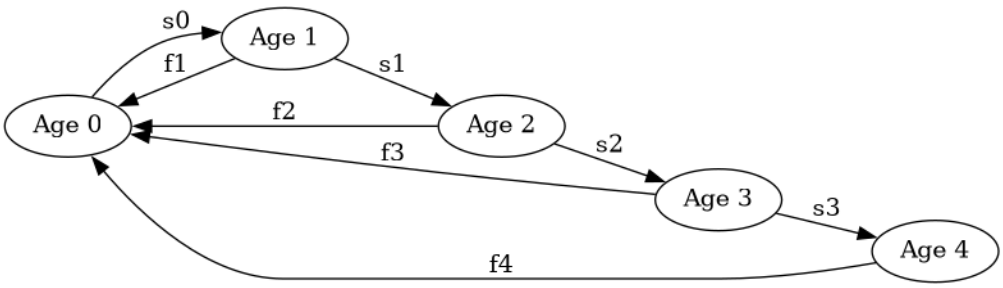
Let's recall our population model. We'll model a certain species of insects with a maximum life span of 4 years, and we'll suppose for `i=0,1,2,3` that for an individual insect of age `i` the probability of survival is given by the number `s_i`, and for `i=1,2,3,4` that for an individual of age `i` the probability of reproduction is given by the number `f_i`.

We represent the population of these insects by a vector

```
import numpy as np
p = np.array([ p0, p1, p2, p3, p4 ])
```

where `pi` represents the population of these insects having age `i`.

a. The following directed graph represents a finite state machine modeling the population.



Find a `5 x 5` matrix `M` so that if `p` represents the population in a given year, then `M @ p` represents the population in the subsequent year.
(Remember that `M @ p` is the `numpy` syntax for the product of `M` and the column vector `p`).

Solution:

$$M = \begin{pmatrix} 0 & f_1 & f_2 & f_3 & f_4 \\ s_0 & 0 & 0 & 0 & 0 \\ 0 & s_1 & 0 & 0 & 0 \\ 0 & 0 & s_2 & 0 & 0 \\ 0 & 0 & 0 & s_3 & 0 \end{pmatrix}$$

or in python notation

```
M = np.array([[0,f1,f2,f3,f4],
              [s0,0,0,0,0],
              [0,s1,0,0,0],
              [0,0,s2,0,0],
              [0,0,0,s3,0]])
```

b. If `p` represents the population in a given year, what matrix calculation must be carried out to find the population after 10 subsequent years?

Solution:
The total population after 10 years should be computed (approximated) by

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \end{pmatrix} \cdot M^{10} \cdot p$$

or – in python –

```
np.ones(5) @ np.linalg.matrix_power(M,10) @ p
```

c. If

```
ones = np.array([ 1, 1, 1, 1, 1 ])
```

explain what is computed by the limit as $n \to \infty$ of the expression

```
ones @ np.linalg.matrix_power(M,n) @ p
```

Solution:
The expression `ones @ np.linalg.matrix_power(M,n) @ p` computes the estimated total population after `n` years.
So the limit – if it exists – describes the long-term equilibrium size of the population.

4. *Markov processes*

   a. What is meant by a *stochastic matrix*?

Solution:
An $n \times n$ matrix is *stochastic* provided that the values in each of its columns sum to 1. In symbols, M is stochastic if

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \end{pmatrix} \cdot M = \begin{pmatrix} 1 & 1 & \cdots & 1 \end{pmatrix}$$

In `python` this can be written as follows:

```
np.ones(n) @ M == np.ones(n)
```

   b. A Markov process is described by a transition diagram (which we assume to be finite). The rows and columns of the corresponding matrix `P` are labeled by the vertices of the diagram, and for two vertices `a,b` the entry `P[a][b]` is given by the edge label of the transition diagram, which represents the probability that the system will change from state `a` to state `b`.
   If `vert` is the full list of vertices of the transition diagram, explain why

```
sum([ P[a][v] for v in vert])
```

   is equal to 1. Must

```
sum([ P[v][a] for v in vert])
```

   be equal to 1?

Solution:
Since `P[a][v]` represents the probability of a transition from state `a` to state `v`, the expression

```
sum([ P[a][v] for v in vert])
```

is the sum of all transition probabilities with initial state `a`. If the state is currently `a`, with probability `1.0` there will be some subsequent state, so for fixed `a` the probabilities `P[a][v]` must indeed sum to 1.

On the other hand, consider a transition diagram with two vertices `a` and `b` with three edges `[a,a]`, `[b,b]` and `[b,a]` where the edge labels are respectively `1, 0.5, 0.5`. The corresponding transition matrix is

$$P = \begin{pmatrix} 1 & 0.5 \\ 0 & 0.5 \end{pmatrix}$$

or in `python`

```
P=np.array([[.5, 1],[.5,0]])
```

In this case,

```
sum([ P[v][a] for v in [0,1] ]) == 1.5
```

So in general

```
sum([ P[v][a] for v in vert])
```

is not equal to 1.

c. Assume that the transition diagram is strongly connected and aperiodic, and let `P` be the corresponding stochastic matrix. According to the Perron-Frobenius Theorem, which of the following statements is correct:

- every eigenvalue of `P` is smaller than 1.
- `P` has `lambda = 1` as an eigenvalue, all other eigenvalues `mu` of `P` have the property that `|mu| < 1`, and the matrix `P - I` has rank `n -1` if `P` is an `n x n` matrix.
- `P` has `lambda = 1` as an eigenvalue, all other eigenvalues `mu` of `P` have the property that `|mu| < 1`, and the dimension of the 1-eigenspace is equal to 1.

Solution:

The first condition is not true.

The latter two conditions are both true:

- `P` has `lambda = 1` as an eigenvalue, all other eigenvalues `mu` of `P` have the property that `|mu| < 1`, and the matrix `P - I` has rank `n -1` if `P` is an `n x n` matrix.
- `P` has `lambda = 1` as an eigenvalue, all other eigenvalues `mu` of `P` have the property that `|mu| < 1`, and the dimension of the 1-eigenspace is equal to 1.

Both of these statements amount to the assertion that "the eigenvalue 1 has multiplicity 1".

To see that these conditions are the same as one another, recall that the dimension of the λ-eigenspace of P is equal to the dimension of the null space of $P - \lambda\mathbf{I}$.

And the dimension of the null space of any $n \times n$ matrix B is equal to n minus the rank of B.

d. Assume that the transition matrix is strongly connected and aperiodic, and let `v` be a 1-eigenvector normalized so that

```
ones = np.vector(n*[1]) # = [1,1,...,1]

ones @ v == 1
```

If `a` is a vertex of the transition diagram, explain why the long-term probability of the system being in the state corresponding to `a` is given by the value `v[a]`.

Solution:
The conditions of the Perron-Frobenius Theorem guarantee that `v` is the only 1-eigenvector. Thus each column of the matrix

$$\lim_{n\to\infty} P^n$$

coincides with `v` and in particular, `v` represents the long-term state, in the sense that the entries of `v` represent the probabilities that the system will be in the various states.

5. *Monte Carlo methods*

Suppose you have to catch a bus early every weekday morning. On a good day, you catch the fast bus that arrives first. Sometimes you miss the fast bus and must take the slow bus. And sometimes you miss both busses. After some observations, you determine that on average, you catch the fast bus twice a week, you catch the slow bus twice a week, and you miss the bus once a week.

a. What is the probability that you miss the bus on any given weekday?

Solution:
There are 5 weekdays in a week; the probability that you catch a bus is thus `2/5 + 2/5 = 4/5` and the probability that you miss the bus is `1/5`.

b. What is the probability that you catch the fast bus three days in a row?

Solution:
The probability that you catch the fast bus on a given day is `2/5`, and so the probability that you do so for three days in a row is $(2/5)^3 = 8/125$

Consider the following code which yields a list of events corresponding to 3 weeks of weekdays; on each weekday, you either catch the fast bus – marked as 'f' –, you catch the slow bus – marked as 's' – or you 'miss' the bus – marked as 'm').

```
import numpy as np
from pprint import pprint
rng = np.random.default_rng()
```

```
def wait_for_bus(fast = 2./5,slow = 2./5):
    return str(rng.choice(['f','s','m'],p=[fast,slow,1-fast-slow]))

pprint([ wait_for_bus() for _ in range(15) ])
```

```
['s', 's', 'f', 's', 'm', 's', 'f', 's', 'f', 's', 'm', 'm', 's', 'f', 's']
```

c. In the given output, do we see the expected number of *misses*?

Solution:
The list of outcomes has length 15, or 3 weeks (of weekdays). There are 3 `misses` in the list, which indeed matches the prediction ("one miss each week").

d. In the given output, do we catch the fast bus the expected number of times?

Solution:
We expect to catch the fast bus twice a week, so we'd expect to catch the fast bus 6 times in three weeks. The list only has 4 `fast-bus` catches, so it doesn't quite reflect the expected value.