EE 263 Homework 1

Luis A. Perez

A simple control algorithm for wireless networks

Solution:

(a) We show that the power control algorithm in the problem described above can be expressed as a linear dynamical system with constant input. We begin with the equation given for the update. We have:

$$\begin{aligned}
p_i(t+1) &= \alpha\gamma\frac{1}{S_i}(t)p_i(t) && \text{(The given algorithm)} \\
&= \alpha\gamma\frac{q_i(t)}{s_i(t)}p_i(t) && \text{(Definition of } S_i\text{)} \\
&= \alpha\gamma\frac{\sigma^2 + \sum_{j\neq i} G_{ij}p_j(t)}{G_{ii}p_i(t)}p_i(t) && \text{(Definition of } s_i(t) \text{ and } q_i(t)\text{)} \\
&= \frac{\alpha\gamma\sigma^2}{G_{ii}} + \sum_{j\neq i}(\alpha\gamma\frac{G_{ij}}{G_{ii}})p_j(t) && \text{(Distribution over sum)} \\
&= b_i + (Ap(t))_i && \text{(Definition of matrix multiplication)}
\end{aligned}$$

From the above, we can easily deduce what $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$ should look like. We have:

$$b_i = \frac{\alpha\gamma\sigma^2}{G_{ii}}$$

$$A_{ij} = \begin{cases} 0 & i = j \\ \frac{\alpha\gamma G_{ij}}{G_{ii}} & \text{otherwise} \end{cases}$$

Note that the above is well-defined since in the problem statement $G_{ii} > 0$.

(b) We plot S_i and p_i as a function of t in Figure 1, with $\gamma = 3$ and the other values as given in the problem statement. In Figure 2 we repeat the same process but not use $\gamma = 5$.

The observations are quite interesting. When $\gamma = 3.0$ (Figure 1), we see that the SINR quickly approachs our target value of $\alpha\gamma = 3.6$ (around $t = 15$). We also note that at this time, the power level for each of the three stations has stabilized.

However, when $\gamma = 5.0$ (Figure 2), we see something quite different. In this case, the SINR appears to stabalize but at a value below our target. As such, the power levels of each station appear to continue to increase.

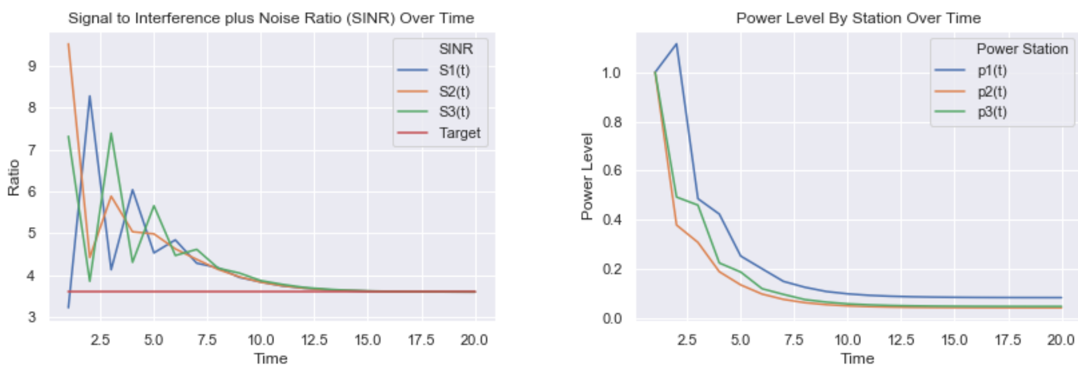


Figure 1: Plot of SINR and Power Level for each of the three station over at each timestep in our simulation. We use the given parameters with $\gamma = 3.0$.

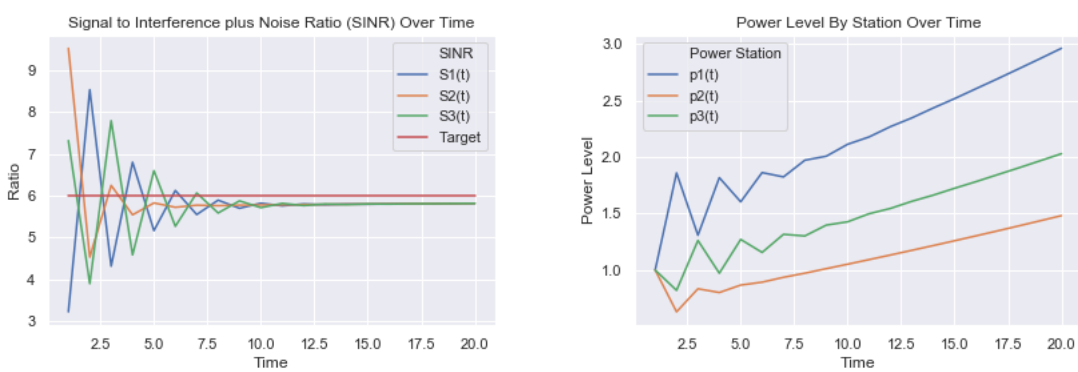


Figure 2: Plot of SINR and Power Level for each of the three station over at each timestep in our simulation. We use the given parameters with $\gamma = 5.0$.

State equations for linear meachanical system.

Solution: The problem gives us the definition of the linear mechanical system as:

$$M\ddot{q} + D\dot{q} + Kq = f$$

where $q(t) \in \mathbb{R}^k$, $f(t) \in \mathbb{R}^k$, $M, D, K \in \mathbb{R}^{k\times k}$. Assuming M is invertible, we wish to write a linear system of equations where the state $x \in \mathbb{R}^{2k}$ and consists of q on top of $\dot{q}$, $u = f$, and $y = q$.

The linear system of equations will therefore take the form:

$$\dot{x} = Ax + Bf, q = Cx + Df$$

where $A \in \mathbb{R}^{2k\times 2k}, B \in \mathbb{R}^{2k\times k}, C \in \mathbb{R}^{k\times 2k}$ and $D \in \mathbb{R}^{k\times k}$. We also define some useful notation. Let $I_k \in \mathbb{R}^{k\times k}$ be the identity matrix in $\mathbb{R}^k$, and similarly, let $0_k \in \mathbb{R}^{k\times k}$ be the zero matrix in $\mathbb{R}^{k\times k}$.

The output system is the simplest. In this case, we have:

$$D = 0_k C \qquad = [I_k \mid 0_k]$$

(The first $k \times k$ section forms the identity matrix, and the second $k \times k$ section is all zero)

With the above definitions, we have:

$$y = Cx + Du = [I_k \mid 0_k][q \mid \dot{q}]^T + 0_k f = q$$

as desired.

For the dynamics equations, we first define a bit of syntax to help with notation. We note that $A \in \mathbb{R}^{2k\times 2k}$ can be divided into four sub-matrices (for each quadrant), $A_{11}, A_{12}, A_{21}, A_{22} \in \mathbb{R}^{k\times k}$. Similarly, B can be decomposed in to two matrices $B_1, B_2 \in \mathbb{R}^{k\times k}$ where B_1 contains the top k rows and B_2 the bottom k rows. We then have:

$$\begin{aligned}
B_1 &= 0_k \\
B_2 &= M^{-1} \\
A_{11} &= 0_k \\
A_{12} &= I_k \\
A_{21} &= -M^{-1}K \\
A_{22} &= -M^{-1}D
\end{aligned}$$

This gives us:

$$
\begin{aligned}
\dot{x} &= Ax + Bu \\
&= A[q \mid \dot{q}]^T + Bf \\
&= [A_{11}q + A_{12}\dot{q} \mid A_{21}q + A_{22}\dot{q}]^T + [B_1 f + B_2 f]^T \\
&= [0_k q + I_k \dot{q} \mid -M^{-1}Kq - M^{-1}D\dot{q}]^T + [0_k f + M^{-1}f]^T \\
&= [\dot{q} \mid -M^{-1}Kq - M^{-1}D\dot{q} + M^{-1}f]^T \\
&= [\dot{q} \mid -M^{-1}(f - D\dot{q} - Kq)]^T \\
&= [\dot{q} \mid \ddot{q}] && \text{(As per problem statement)}
\end{aligned}
$$

This shows our given linear system of equations corresponds to our mechanical system.

Some standard time-series models

Solution: We assume the models are single-input, single-output for our discussion below. However, as the remark makes clear, the model we propose is readily extensible to multi-input, multi-output systems by allowing the referenced coefficients to me matrices (and expanding them as such in our models below).

The MA Model We use the state specified in the problem statement where $x(k) \in \mathbb{R}^r$ is a vector consists of $[u(k-1), \cdots, u(k-r)]^T$. We now define our matrices $A \in \mathbb{R}^{r \times r}, B \in \mathbb{R}^{r \times 1}, C \in \mathbb{R}^{1 \times r}$ and $D \in \mathbb{R}^{1 \times 1}$ as follows:

$$
\begin{aligned}
D_{11} &= a_0 \\
C_{1i} &= a_i, i = 1, \cdots, r \\
B_{1i} &= \begin{cases} 1 & i = 1 \\ 0 & \text{otherwise} \end{cases} \\
A_{ij} &= \begin{cases} 1 & i = j+1 \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

With the above, we have:

$$
\begin{aligned}
Ax(k) + Bu(k) &= A[u(k-1), \cdots, u(k-r)]^T + Bu(k) \\
&= A_1 u(k-1) + \cdots + A_r u(k-r) + B_{11} u(k) \\
&\qquad \text{(Definition of matrix multiplication)} \\
&= [u(k), u(k-1), \cdots, u(k-r+1)]^T \\
&\qquad \text{(A shifts the input down to make space for latest signal from B)} \\
&= x(k+1)
\end{aligned}
$$

as desired. And also, we have:

$$
\begin{aligned}
Cx(k) + Du(k) &= \sum_i C_{1i} u(k-i) + a_0 u(k) \qquad \text{(Definition of matrix multiplication)} \\
&= \sum_{i=0} a_i u(k-i) \\
&= y(k)
\end{aligned}
$$

which is exactly the dynamics needed for the MA model.

The AR Model We use the state specified in the problem statement where $x(k) \in \mathbb{R}^p$ is a vector consists of $[y(k-1), \cdots, y(k-r)]^T$. We now define our matrices $A \in \mathbb{R}^{p \times p}, B \in$

$\mathbb{R}^{p\times 1}, C \in \mathbb{R}^{1\times p}$ and $D \in \mathbb{R}^{1\times 1}$ as follows:

$$D_{11} = 1$$
$$C_{1i} = b_i, i = 1, \cdots, p$$
$$B_{1i} = \begin{cases} 1 & i = 1 \\ 0 & \text{otherwise} \end{cases}$$
$$A_{ij} = \begin{cases} b_j & i = 1 \\ 1 & i = j+1 \\ 0 & \text{otherwise} \end{cases}$$

With the above, we have:

$$\begin{aligned} Ax(k) + Bu(k) &= A[y(k-1), \cdots, y(k-p)]^T + Bu(k) \\ &= A_1 y(k-1) + \cdots + A_p y(k-p) + B_{11}u(k) && \text{(Definition of matrix multiplication)} \\ &= [u(k) + b_1 y(k-1) + \cdots + b_p y(k-p), y(k-1), \cdots, y(k-p+1)]^T \\ &= [y(k), y(k-1), \cdots, y(k-p+1)]^T \end{aligned}$$

The second to last line requires a bit of explanation. A shifts the input down to make space for the new output. Unlike the last model where we just needed to carry the input through, in this case we need to use the input passed by B ($u(k)$) to compute the current output $y(k)$. This is done by the first row of A which does a weighed sum over all previous $y(k-p)$ values stored in our current state which is added to the input from B. It is immediate then that the new state is $x(k+1)$ as desired.

And also, we have:

$$\begin{aligned} Cx(k) + Du(k) &= \sum_{i=1} C_{1i} y(k-i) + u(k) && \text{(Definition of matrix multiplication)} \\ &= u(k) + \sum_{i=1} b_i y(k-i) \\ &= y(k) \end{aligned}$$

which is exactly the dynamics needed for the AR model.

The ARMA Model We're going to do the simplest thing we can think of. We will use the state $x(k) \in \mathbb{R}^{p+r}$ which is a vector consisting of the previous p outputs and the previous r inputs. That is to say, we have the state:

$$x(k) = [y(k-1), \cdots y(k-p), u(k-1), \cdots, u(k-r)]^T$$

This makes defining our new linear system relatively straight-forward, since we can just combine our previous two systems. We will have four matrices, $A \in \mathbb{R}^{(p+r)\times(p+r)}, B \in$

$\mathbb{R}^{(p+r)\times 1}, C \in \mathbb{R}^{1\times(p+r)}, D \in \mathbb{R}^{1\times 1}$. These are defined as follows:

$$D_{11} = a_0$$
$$C_{1k} = \begin{cases} b_k & 1 \leq k \leq p \\ a_{k-p} & p+1 \leq k \leq p+r \end{cases}$$
$$B_{1k} = \begin{cases} a_0 & k = 1 \\ 1 & k = (p+1) \\ 0 & \text{otherwise} \end{cases}$$
$$A_{ij} = \begin{cases} b_j & i = 1, 1 \leq j \leq p \\ a_{j-p} & i = 1, p+1 \leq j \leq p+r \\ 1 & i = j+1, j \neq p \\ 0 & \text{otherwise} \end{cases}$$

The above might be somewhat hard to parse, so we explain each matrix in more detail. D is the same as in the MA model, since this model is idential to the MA model in that regard. C is simply the concatenation of the corresponding C matrices used in the AR and MA models, since the output of this model is nearly a sum of the output of both models (D handles the edge-case).

The matrix B perform two key purposes. It scales the input by a_0 so it can be used to compute the current timesteps' output (needed to generate the next state), and also simply passes the input as-is along the $p+1$ index, so it can be added to the new state.

The matrix A is likely the most complicated to understand, but again, this is simply composed of four smaller submatrices. The bottom-left matrix of dimension $r \times p$ is just all zero. The top-left matrix of dimensions $p \times p$ is the exact same as the matrix in the AR model. The bottom-right matrix of dimensions $r \times r$ is the exact same as the matrix in the MA model. And finally, the top-right matrix of dimension $p \times r$ is simply a zero matrix where the first row contains the a_i co-efficients. Putting it all together, the only interesting aspect is the first row, which contains all relevant co-efficients needed to compute the current timestep's output.

As such, we have:

$$\begin{aligned} Ax(k) + Bu(k) &= A[y(k-1), \cdots, y(k-p), u(k-1), \cdots, u(k-r)]^T + Bu(k) \\ &= A_1 y(k-1) + \cdots + A_p y(k-p) \\ &+ A_{p+1} u(k-1) + \cdots + A_{p+r} u(k-r) \\ &+ B_{11} u(k) + B_{(p+1)1} u(k) \qquad \text{(Definition of matrix multiplication)} \\ &= [b_1 y(k-1) + \cdots + b_p y(k-p) + a_0 u(k) + \cdots + a_r u(k-r), \\ & y(k-1), \cdots, y(k-p+1), u(k), \cdots, u(k-r+1)]^T \\ &= [y(k), y(k-1), \cdots, y(k-p+1), u(k), \cdots, u(k-r+1)]^T \end{aligned}$$

Finally, we have:

$$\begin{aligned}
Cx(k) + Du(k) &= \sum_{i=1}^{p} C_{1i} y(k-i) + \sum_{i=1}^{r} C_{1(p+i)} u(k-i) + D_{11} u(k) \\
&\qquad \text{(Definition of matrix multiplication)} \\
&= \sum_{i=1}^{p} b_i y(k-i) + \sum_{i=1}^{r} a_i u(k-i) + a_0 u(k) \\
&= y(k)
\end{aligned}$$

Therefore, this linear system is appropriate for the ARMA model.

Representing linear functions as matrix multiplication

Solution: Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be a linear function. We show that there is a matrix $A \in \mathbb{R}^{m\times n}$ such that for all $x \in \mathbb{R}^n, f(x) = Ax$.

To do this, we begin with some notation. Let $e_i \in \mathbb{R}^n$ be the vector containing a 1 for the i-th dimension, and zero otherwise. Now, define A as follows:

$$A_{ij} = f(e_j)_i \qquad \text{(The } ij\text{-th entry of } A \text{ is the } i\text{-th entry of the result of } f(e_j))$$

Stated another way, the n columns of A consist of f evaluated on each unit basis vector.

We now show that with this A, we indeed have $f(x) = Ax$ for all $x \in \mathbb{R}^n$. Let us first evaluate the LHS of this equation.

$$\begin{aligned} f(x) &= f(x_1e_1 + \cdots + x_ne_n) \\ &\qquad \text{(Decompose any } x \text{ as a linear combination of basis vectors } e_j) \\ &= x_1f(e_1) + \cdots + x_nf(e_n) && \text{(Linearity of } f) \end{aligned}$$

We now evaluated the RHS. Let A_k correspond to the k-th column of A.

$$\begin{aligned} Ax &= x_1A_1 + x_2A_2 + \cdots + x_nA_n \\ &\qquad \text{(Defintion of matrix multiplication as a linear combination of matrix columns)} \\ &= x_1f(e_1) + x_2f(e_2) + \cdots + x_nf(e_n) && \text{(Construction of } A \text{ as described above)} \end{aligned}$$

From the above, we see that the LHS and RHS are equivalent for arbitrary choice of $x \in \mathbb{R}^n$.

The next question is whether A is unique. The answer is yes. We prove by contradiction. Let us suppose that a matrix $\tilde{A}$ satisfying the above properties exists, and that $\tilde{A} \neq A$.

However, by our assumption, $\tilde{A} \neq A$, so there is some $A_{ij} \neq \tilde{A}_{ij}$. As such, we have that $A_j \neq \tilde{A}_j$ (the j-th columns differ). Now take $x = e_j$ (the unit vector with a 1 in the j-th dimension, zero everywhere else).

Then note that:

$$f(e_j) = Ae_j = A_j \neq \tilde{A}_j = \bar{A}e_j \implies f(e_j) \neq \bar{A}e_j$$

This is a contradiction on the fact that $f(x) = \tilde{A}x$ for all $x \in \mathbb{R}^n$. As such, our assumption that $A \neq \bar{A}$ must be wrong.

Counting sequences in language code

Solution:

(a) $B_{ij} = A^r_{ij}$ is simply the number of valid sequences in our language of length $(r+1)$ whose first symbol is j and whose last symbol is i.

We now proof the above claim using induction.

Base Case For the base case, we take $r = 1$. so we have $B = A$. As given in the problem statement, $A_{ij} = 1$ if symbol i is allowed to follow symbol j. As such, B_{ij} is 1 when the 2-letter sequence ji is a valid sequence in our language. This satisfies our property, since B_{ij} is then exactly the count of valid sequences of length 2 whose first symbol is j and last symbol is i.

Inductive Step Suppose our property holds for $B'_{ij} = A^r$. We now show that it also holds for $B_{ij} = A^{r+1}$. To do this, we consider how to compute B_{ij}. This is mechanically given as:

$$\begin{aligned}
B_{ij} &= A^{r+1}_{ij} \\
&= (AA^r)_{ij} \\
&= (AB')_{ij} \\
&= \sum_k A_{ik}B'_{kj}
\end{aligned}$$

We know that A_{ik} will be 1 if i is allowed to follow k. By our inductive hypothesis, we know that B'_{kj} is the number of valid sequences in our language of length $(r+1)$ whose first symbol is j and whose last symbol is k.

As such, the summation above will simply sum together the count of all sequence of length $(r+1)$ which start with j (can end with anything) and can be extended by adding i to the end. The result is therefore just the number of sequences in our language of length $(r+2)$ which start with j and end with i. This proof our inductive case.

(b) To solve this problem, we define A as described above. In particular, we have:

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

We then compute A^9, which turns out to be:

$$A^9 = \begin{bmatrix} 41 & 49 & 24 & 113 & 37 \\ 55 & 65 & 31 & 150 & 49 \\ 42 & 49 & 23 & 113 & 37 \\ 0 & 0 & 0 & 1 & 0 \\ 31 & 37 & 18 & 86 & 28 \end{bmatrix}$$

The above matrix contains the count of all valid words of length 10 which begin and end with symbols j and i. As such, just taking the sum of all entries will give us the total number of valid sequences of length 10. This comes out to be 1079.

We could follow a similar approach (with A being a matrix of all 1s) to count all possible sequences of length 10 (where all characters are allowed). However, a simpler approach is simply to compute $5^{10} = 9,765,625$. This is because we have 10 spaces, and each space can contain any of the 5 symbols and still be valid.

As far as comparison, our set of valid sequences is actually very small. It is only about 0.01104896%.

(c) We wish to count, among all allowed sequences of length 10, what the most common value for the seventh symbol is. We can do this by noting that all allowed sequences of length 10 must consist of an allowed sequence of length 7 which overlaps by one character with an allowed sequence of length 4. For example, 125312 (allowed sequence of length 7) and 2512 (allowed sequence of length 4) when put together (with the final and first characters overlapping) form the sequence 12531**2**512 of length 10.

With this insight, what we can do is look at the matrix A^6 as well as the matrix A^4. We know that A^6_{ij} gives us the number of valid sequences in our languages of length 7 whose first symbol is j and whose last symbol is i. Similarly for $A^3_k i$, which gives us the number of valid sequences of length 4 whose first symbol is i and last symbol is k.

Then we can define:

$$Z = \left(A^6 \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \right) \times \left(\begin{bmatrix} 1 & \cdots & 1 \end{bmatrix} A^3 \right)^T$$

where $\times$ is element-wise multiplication. Then note that with the above, we have:

$$Z_i = \sum_j A^6_{ij} \times \sum_k A^3_{ki}$$

Note that the first term in the product is counting the number of valid sequences of length 7 which end with i, while the second term is counting the number of valid sequences of length 4 which start which start with i. As such, Z_i is simply the number of valid sequences of length 10 which contain i as their seventh symbol.

Computing this using Python, we have:

$$Z = \begin{bmatrix} 288 \\ 446 \\ 144 \\ 14 \\ 185 \end{bmatrix}$$

and from this, we can immediately determine that among all allowed sequences of length 10, the most common value for the 7-th symbol is 2.

Express the following statements in matrix language

Solution:

(a) $Z = TY$ where $T_{ij} = 0$ for $i > j$ and all $i > n$. Note that the last requirement is required since the problem does not specify that Y has at most n rows (and as such, could have more, which must be ignored when forming Z).

(b) $W = VP$ where P is a square of all zeros except near the main diagonal. There, we have small 2×2 matrices along the main diagonal. Each small 2×2 matrix is zero everywhere except the anti-diagonal (top right and bottom left) where it has ones. If the dimension of P is odd, the bottom-right corner has a 1 (we assume that in the case of an odd number of columns, the last column is left unmodified).

(c) We assume the angle measured is the smallest possible angle between the two vectors (eg, $\theta' = \min\{\theta, 360^\circ - \theta\}$). Then we have $P^T Q = R$ where $R_{ij} \geq 0$ for all i, j.

(d) Similarly to above, we assume the angle being measured is the smallest possible angle between the two vectors. Then we have $P^T Q = R$ where $R_{ii} \geq 0$.

(e) $A^T A = X$ where $X_{ii} = 0$ for $i = 1, \cdots k$.

Proof of the Cauchy-Schwarz inquality

Solution: The proof is rather straight forward with the given hint. We show the step-by-step process below.

$$
\begin{aligned}
&||(||y||x \pm ||x||y)||^2 \geq 0 \\
&\hspace{6em}\text{(Square of a real number is non-negative)} \\
&\implies \sum_i (||y||x_i \pm ||x||y_i)^2 \geq 0 \quad \text{(Definition of vector norm)} \\
&\implies \sum_i (||y||^2 x_i^2 + ||x||^2 y_i^2 \pm 2||x|| \cdot ||y|| x_i y_i \geq 0 \quad \text{(Squaring inner sum term)} \\
&\implies ||y||^2 \sum_i x_i^2 + ||x||^2 \sum_i y_i^2 \pm 2||x|| \cdot ||y|| \sum_i x_i y_i \geq 0 \quad \text{(Distributing the sum)} \\
&\implies ||y||^2||x||^2 + ||x||^2||y||^2 \pm 2||x|| \cdot ||y|| x^T y \geq 0 \\
&\hspace{6em}\text{(Definition of vector norm and dot product)} \\
&\implies 2||x||^2||y||^2 \geq \mp 2||x|| \cdot ||y|| x^T y \quad \text{(Some algebra)} \\
&\implies ||x|| \cdot ||y|| \geq \mp x^T y \quad \text{(When } ||x|| \neq 0 \text{ and } ||y|| \neq 0\text{)}
\end{aligned}
$$

As such, we have $x^T y \leq ||x|| \cdot ||y||$ and also $x^T y \geq -||x|| \cdot ||y||$ as long as

$$||x|| \neq 0, ||y|| \neq 0$$

. In the case where either vector is the zero vector, it's immediately clear that $|x^T y| = ||x|| \cdot ||y||$. As such, putting all of the above together, we arrive at the Cauchy-Schwarz inequality for all x, y:

$$\implies |x^T y| \leq ||x|| \cdot ||y||$$

As for when equality holds, the first case is when at least one of x or y is the zero vector. This is immediate from the definition of the inequality.

In the case where neither is the zero vector, we must have:

$$
\begin{aligned}
&|x^T y| = ||x|| \cdot ||y|| \\
&\implies \left| \left(\frac{x^T}{||x||}\right)\left(\frac{y}{||y||}\right) \right| = 1 \quad \text{(Neither } x \text{ nor } y \text{ is the zero vector)} \\
&\implies |u_x^T u_y| = 1 \quad (u_v \text{ is the unit vector in the direction of } v)
\end{aligned}
$$

The above implies that the equality holds when neither x nor y are the zero vector if and only if they are pointing in the same ($= 1$) or opposite ($= -1$) directions.

HW1

July 1, 2019

1 EE 263: HW1 Notebook

Author: Luis Perez **Last Updated**: June 30, 2019

1.1 Imports

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns; sns.set()
```

1.2 Problem 1: A simple power control algorithm for wireless networks

1.2.1 Part (b) - Simulation of Algorithm

[2]:
```
"""
The goal is to simulate the power control dynamics using our solution
in (a). We are given the following input:

         1 .2 .1
G =     .1  2 .1
        .3 .1  3
 = 3,   = 1.2,   = 0.1.
"""
G = np.array([
    [1, .2, .1],
    [.1, 2, .1],
    [.3, .1, 3]])
```

[3]:
```
def getLinearSystem(G, gamma=3, alpha=1.2, sigma=0.1):
    # This broadcasts the diagonal as columns.
    diagonal = np.stack([G.diagonal(), G.diagonal(), G.diagonal()]).T
    # With the above, this gives ij / ii
    A = alpha * gamma * G / diagonal
    # And we also zero out the diagonal.
    A[np.identity(A.shape[0]) == 1] = 0
```

```
    b = alpha * gamma * sigma**2 / G.diagonal()
    return A,b
```

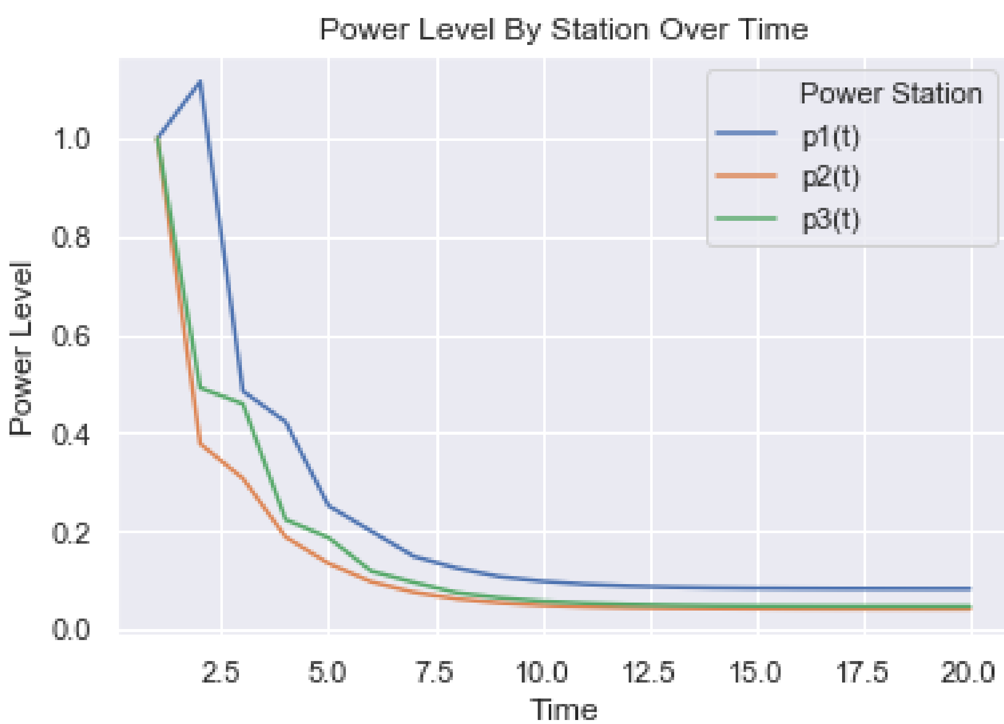
[4]:
```
# We run the simulation. We compute S(t) and p(t).
def simulation_step(G, A, b, p, sigma = 0.1):
    """One step in the simulation.
    Given current power levels p, return (p(t+1), S(t))
    """
    localG = np.copy(G)
    localG[np.identity(G.shape[0]) == 1] = 0
    q = sigma**2 + np.dot(localG, p)
    s = G.diagonal() * p
    return b + np.dot(A, p), s / q
```

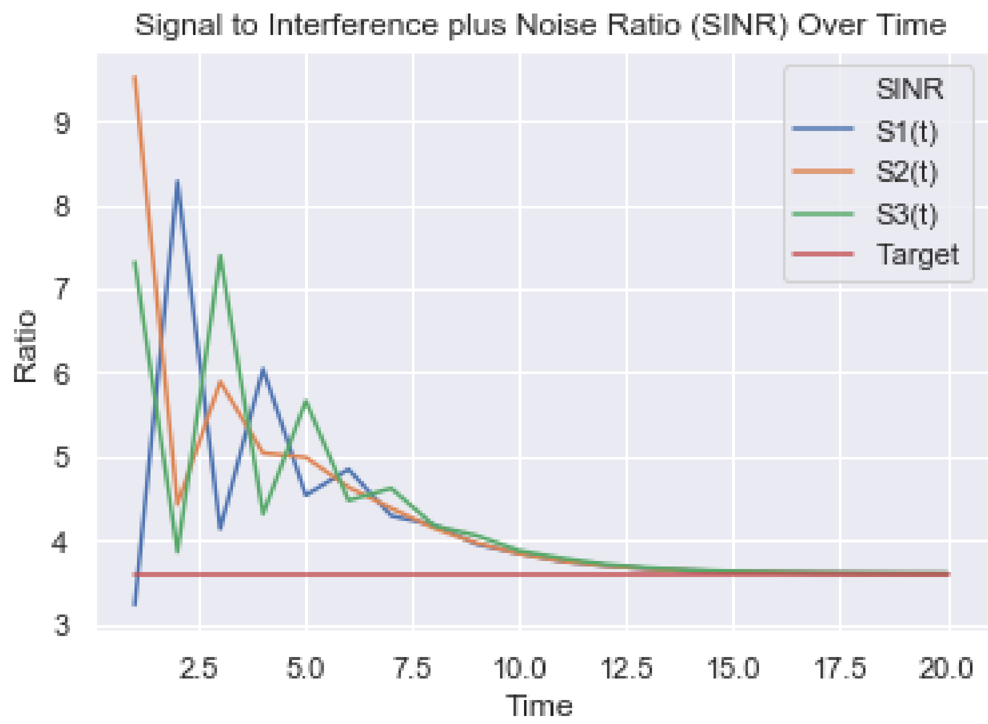
[5]:
```
def simulate(numSteps = 100, gamma=3, alpha=1.2):
    """Runs the simulation for numSteps.

    Returns list of (p(t), S(t)) for each timestep.
    """
    A, b = getLinearSystem(G, gamma=gamma, alpha=alpha)
    results = []
    prevP = np.ones(A.shape[0])
    for _ in range(numSteps):
        newP, S = simulation_step(G, A, b, prevP)
        results.append((prevP, S))
        prevP = newP
    return results
```

[6]:
```
# Plot the results. We plot S(i) and p(i) over t, as well as alpha y
def plot(numSteps=100, gamma=3, alpha=1.2, saveFig=True):
    results = simulate(numSteps=numSteps, gamma=gamma, alpha=alpha)
    pts, Sts = zip(*results)
    powerData = pd.DataFrame()
    powerData['p1(t)'], powerData['p2(t)'], powerData['p3(t)'] = zip(*pts)

    ratioData = pd.DataFrame()
    ratioData['S1(t)'], ratioData['S2(t)'], ratioData['S3(t)'] = zip(*Sts)
    ratioData['Target'] = [alpha * gamma] * len(results)

    powerData['Time'] = range(1, numSteps+1)
    ratioData['Time'] = range(1, numSteps+1)

    longPowerData = pd.melt(powerData, id_vars=['Time'],
                            var_name='Power Station')
    longRationData = pd.melt(ratioData, id_vars=['Time'],
                             var_name=['SINR'])

    ax = sns.lineplot(x='Time', y='value',
```

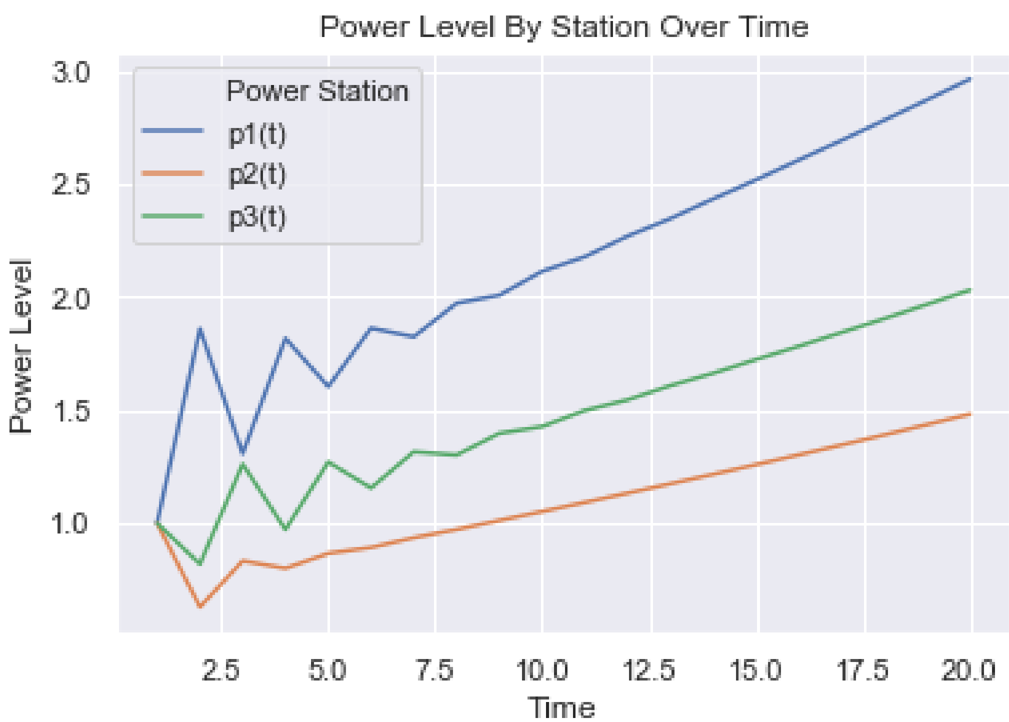
```
                          hue='Power Station', data=longPowerData)
    ax.set_title("Power Level By Station Over Time")
    ax.set_ylabel("Power Level")
    plt.show()
    if saveFig:
        ax.get_figure().savefig("power_level_steps_%s_gamma_%s.png" %␣
↪(numSteps, gamma))

    plt.close()
    ax = sns.lineplot(x='Time', y='value',
                          hue='SINR', data=longRationData)
    ax.set_title("Signal to Interference plus Noise Ratio (SINR) Over Time")
    ax.set_ylabel("Ratio")
    plt.show()
    if saveFig:
        ax.get_figure().savefig("sinr_steps_%s_gamma_%s.png" % (numSteps,␣
↪gamma))
```

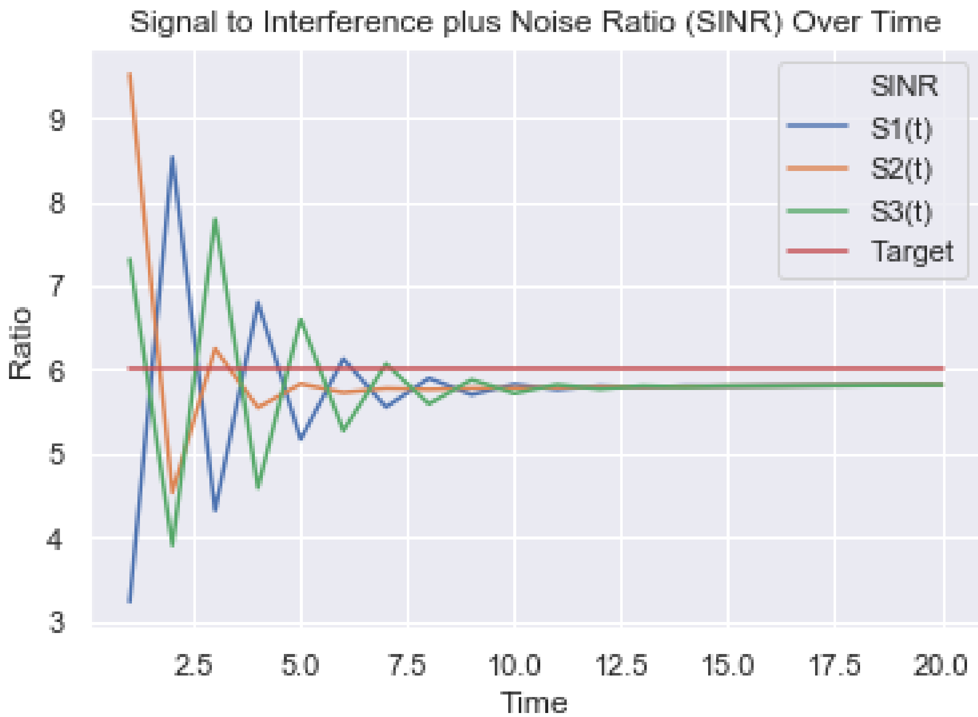
[7]:
```
# Plot with gamme = 3
plot(numSteps=20, gamma=3, saveFig=False)
```





[8]:
```
# Repeat with gamma = 5
plot(numSteps=20, gamma=5, saveFig=False)
```





1.3 Problem 5: Counting Sequences

1.3.1 Part (b) - Total number of length-10 sequences

[9]:
```
"""
The language transition model. Corresponds to the following rules:
     1 must be followed by 2 or 3
     2 must be followed by 2 or 5
     3 must be followed by 1
     4 must be followed by 4 or 2 or 5
     5 must be followed by 1 or 3

Where A_{ij} means that character j can be followed by character i.
"""

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

[10]:
```
"""
Compute A^9 which answers the question. Each entry answers the question:
    How many valid sequences of length 10 are there whose first sequence
    character is j and last sequence characnter is i.
"""
B = np.linalg.matrix_power(A, 9)
```

[11]:
```
"""
Actually sum all values to answer the question of how many
sequences of length 10 there are.
"""
print("There are %i valid sequences of length 10" % B.sum())
```

```
There are 1079 valid sequences of length 10
```

[12]:
```
"""
Quick double check. Total # of works should be 5^10
"""
assert np.linalg.matrix_power(np.ones(A.shape), 9).sum() == 5**10
```

[13]:
```
print("Valid sequences is %s%% of total posssible" % (100*B.sum() / 5**10))
```

```
Valid sequences is 0.01104896% of total posssible
```

1.3.2 Part (c) - Most Frequent 7-th Symbol in length-10 sequences

[14]:
```
"""
We look at A^6 and A^3 as explained in the homework assigment.
"""
A6 = np.linalg.matrix_power(A, 6)
A3 = np.linalg.matrix_power(A, 3)
```

[15]:
```
Z = np.dot(A6, np.ones((A6.shape[0], 1))) * np.dot(np.ones((1, A3.shape[0])),
 ↪A3).T
```

[16]:
```
print("Among all allowed sequences of length 10, "
      "the most common value for the 7-th symbol is %i." % (np.argmax(Z) + 1))
```

```
Among all allowed sequences of length 10, the most common value for the 7-th
symbol is 2.
```

[17]:
```
Z
```

[17]:
```
array([[288.],
       [448.],
       [144.],
       [ 14.],
       [185.]])
```